8051 多功能加法器

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……而這是上半部分，下半部分則開始進入較難的時鐘信號、以及中斷服務程式、串列通訊…等等，雖然在正課上有學過，實驗課也有用過，但若並非自己實際去逐一撰寫的話，很難去理解其實際的運作方式，因此期末專題的設計幾乎都與中斷服務程式有所相關。

在這次的多功能加法器上，最難的地方就是你要隨著現在 Megawin 所處的模式去開啟以及初始化特定的 Timer，因此對於 Timer 相關的各種設定都必須相當了解才行，也要避免同時啟動到不需要的 Timer，這是這次專題的最大難處，而加法器的製作原理其實就和當初期中次系統專題的邏輯相像，只是扣除了小數的運算，因此加法器的部分難度並不會比期中次系統專題來得高，但必須將輸入法改為滾輪數字輸入，並加上串列通訊之功能才行。

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：

我的加法是用 packed 的寫法去寫的，會選擇用這個寫法一部分的原因是想嘗試，另一部分的原因則是因為在相加後，當我要把 16 進制轉成 10 進制時，有現成的 DA A 可以使用，不會像當初減法器一樣，存一個數字卻用了 2byte 的空間，相較起來也比較省空間，但邏輯上的撰寫會需要格外的小心，因為程式撰寫上會不斷地使用到 SWAP A 或是 ANL，且需要從最高位開始做左移，需要不斷檢查是否有錯，避免產生旋轉錯誤。

若輸入 1234，我的記憶體會是這樣儲存(紅字為記憶體初始化的部分)：

	57h	58h
First_num	**00**	**01**
	00	**12**
	01	**23**
	12	**34**

↪ 每按下一次 Enter，就會呼叫 Save_num 進行儲存，藉由 SWAP 和 AND 產生數字左移效果

在相加上，每相加完 2byte 都要使用一次 DA A 將數值做轉換，而下一次的相加必須用 ADDC，必須連 Carry 一起運算。

若為 1234+5678(紅字為記憶體初始化的部分)：

First_num	57h	58h
	12	**34**
Sec_num	59h	5Ah
	56	**78**
Answer (第 1 次相加 ADD)	5Bh	5Ch
	00	**BC**
DA A	**00**	**12**
Answer (第 2 次相加 ADDC)	**69**	**12**
DA A	**69**	**12**

↪ 此表為加法運算過程

這樣即完成加法運算。

這次有許多的 Timer 要使用，因此必須要弄清楚，其中各 Timer 的功用如下：

	模式	功用
Timer 0	Wave	顯示自己的學號頻率(755)方波信號
	Meas	顯示 1s 亮 1s 暗的方波信號
Timer 1	串列通訊	設定 baud rate = 4800，SOMD = 1
External Interrupt 1	Meas	負責數測量到的學號頻率為多少
Serial	串列通訊	檢查為 TI 觸發或是 RI，藉由 SBUF 傳送或接收

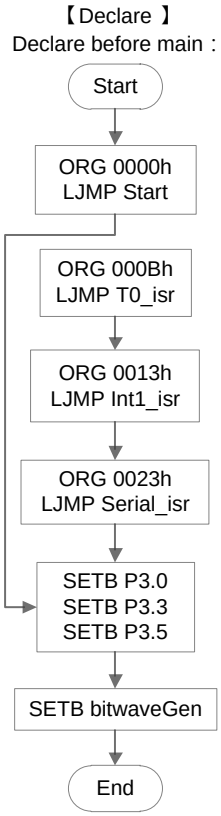
↪ 此表為各個 Timer 功用闡述

副程式 Bcd_daa、Bcd2display、Hex2bcd 主要是輔助學號頻率測量，在 Timer0 不斷閃爍時，從 P3.5 接收到的訊號有一個 low → high 的 pass 會啟動 External Interrupt 1，而當該訊號有一個 high → low 的 pass 將 16 進制數值轉換成 10 進制後，直接呼叫 Bcd2display 做顯示輸出，即完成學號頻率測量。

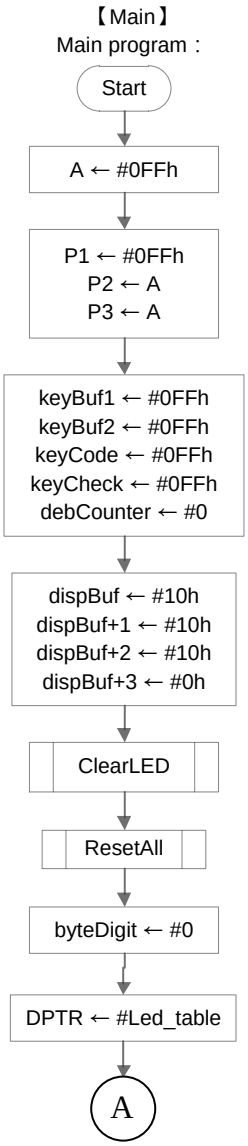
記憶體定義：

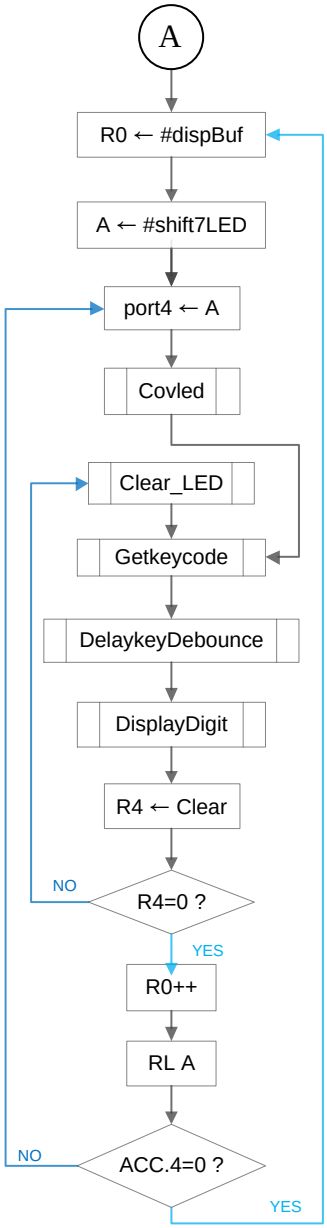
```
;-------------------------------------------------------------
;【Define】 pseudo name
//用來自定義新的名稱或位置去存取,紀錄其他東西
;-------------------------------------------------------------
port4           EQU     0E8h    //定義 port4 為位置 0E8h
shift7LED       EQU     0FEh    //目的使七字節每次只有一個點亮
dispBuf         EQU     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
t0Buf           EQU     49h     //用來存取 t0Count,執行 DJNZ 造成 Delay
t0Count         EQU     40      //40*0.025sec=1sec,使 Timer0 產生 1s
byteBCD0        EQU     4Dh     //存取要轉換後之數值的個位.十位十進制)
byteBCD1        EQU     4Eh     //存取要轉換後之數值的百位.千位(十進制)
byteBCD2        EQU     4Fh     //存取要轉換後之數值的萬位十萬位(十進制)
byteHex1        EQU     50h     //存取要轉換之數值的最高兩位(十六進制)
byteHex0        EQU     51h     //存取要轉換之數值的最低兩位(十六進制)
byteTime        EQU     52h     //控制 Timer0 時間,1 時為傳送頻率,40 為接收
byteTH0         EQU     53h     //在 Wave 和 Meas 模式下 TH0 不同
byteTL0         EQU     54h     //在 Wave 和 Meas 模式下 TL0 不同
                                //用來存不同模式時 TH0.TL0 該給多少
bitwaveGen      BIT     P1.5    //表示現在是否為 Wave 模式(傳送自己學號頻率)
bitTimingGate   BIT     P1.0    //用來產生 1s 亮 1s 暗,或是亮自己的學號頻率
byteDigit       EQU     55h     //用來數現在滾輪數字按到多少
byteTemp        EQU     56h     //在 DisplayDigit 中協助做 P1.1~P1.4 顯示
First_num       EQU     57h     //57h~58h 用來存輸入的第一個數值
Sec_num         EQU     59h     //59h~5Ah 用來存輸入的第二個數值
Answer          EQU     5Bh     //5Bh~5Ch 用來存相加之值
Save_whom       EQU     5Dh     //用來記住現在該存取數值 1 還是數值 2
Clear           EQU     5Eh     //判斷是否要執行 clear
```

完整流程圖：

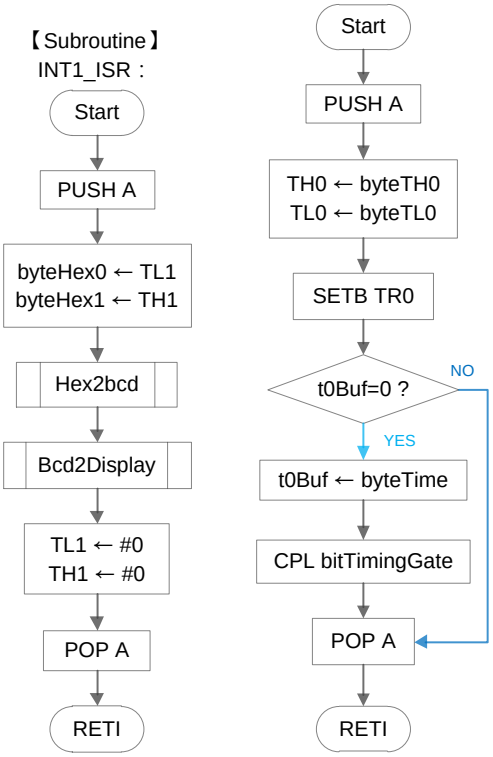


▲Declare 流程圖



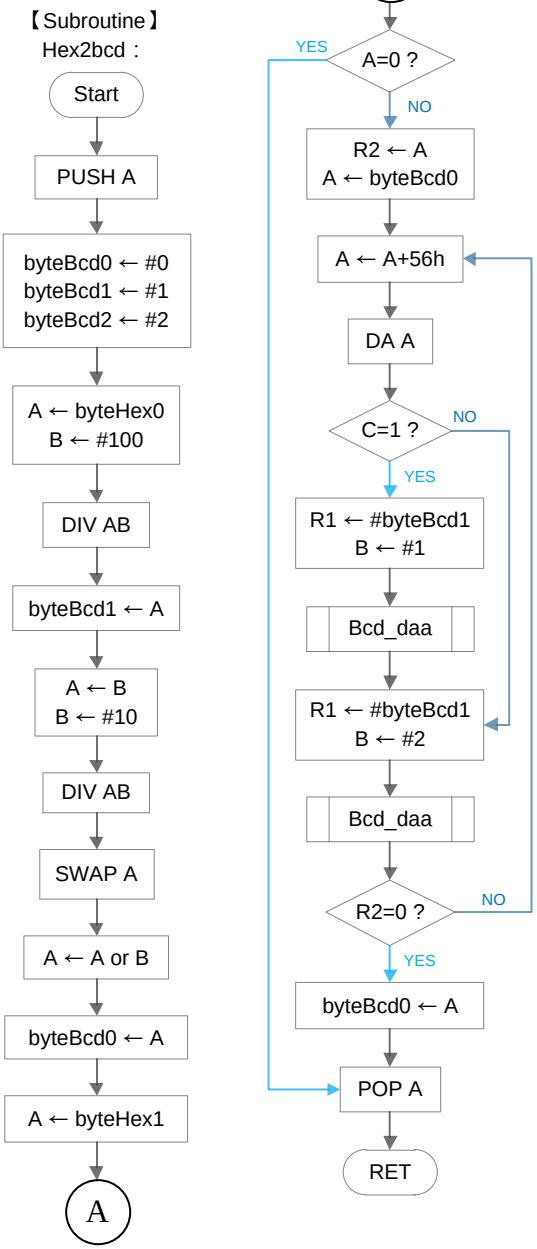


▲Main 流程圖

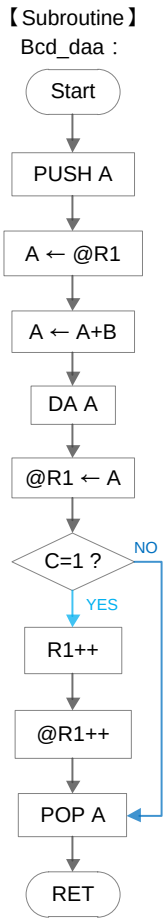


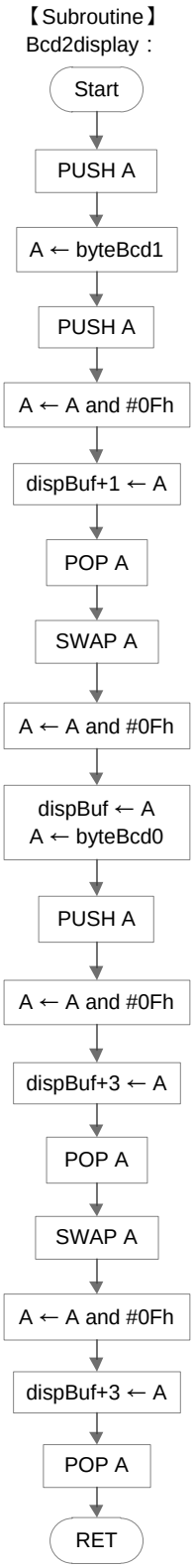
▲INT1_ISR 流程圖

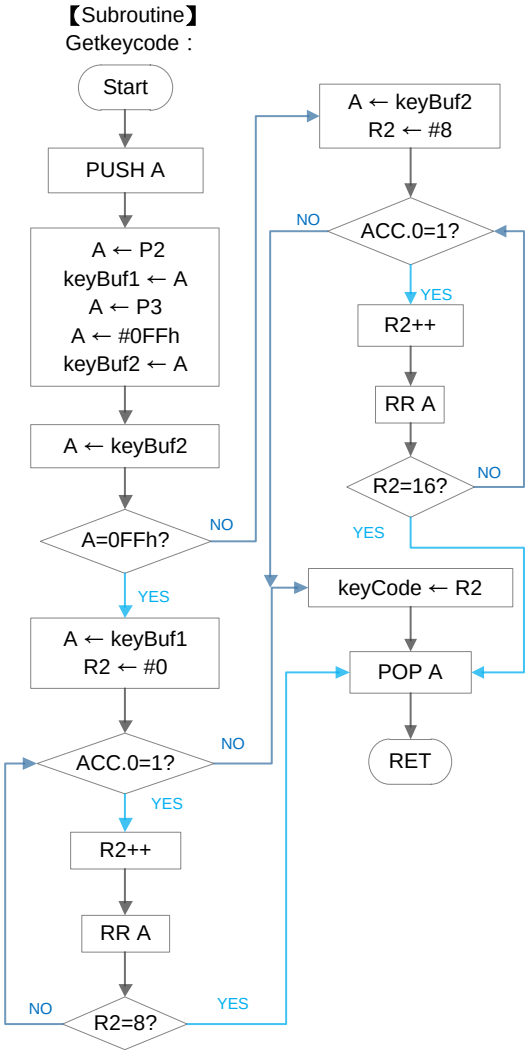
▲INT0_ISR 流程圖



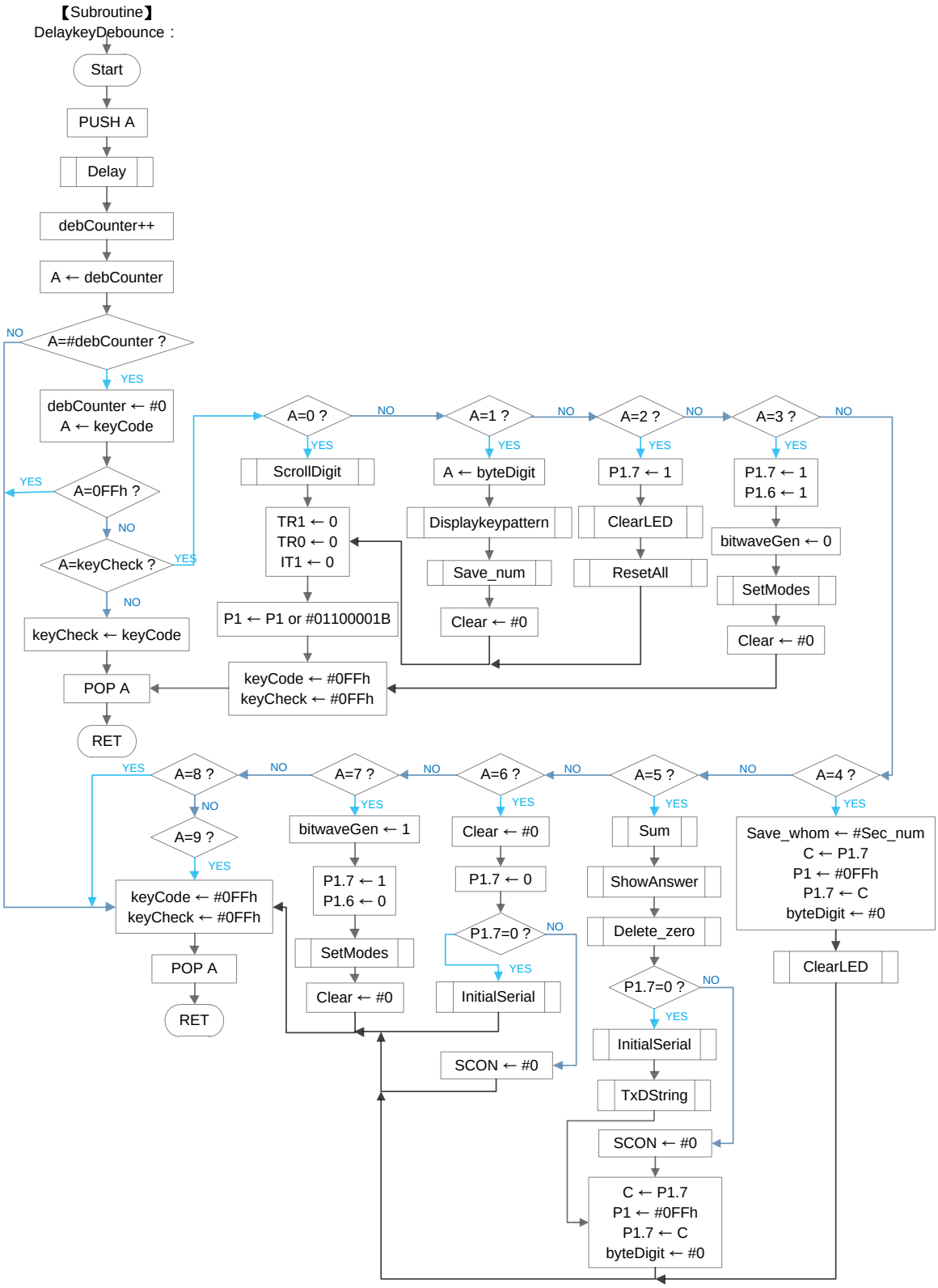
▲Hex2bcd 流程圖



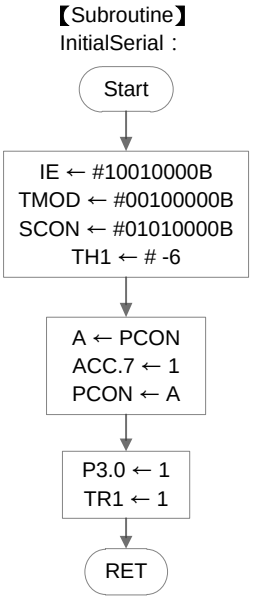




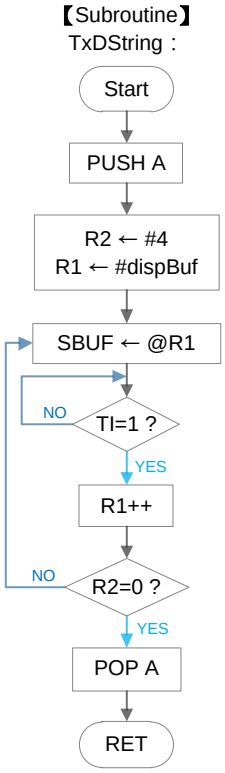
▲Bcd_daa 流程圖 **▲Bcd2diaplay 流程圖** **▲Getkeyvode 流程圖**



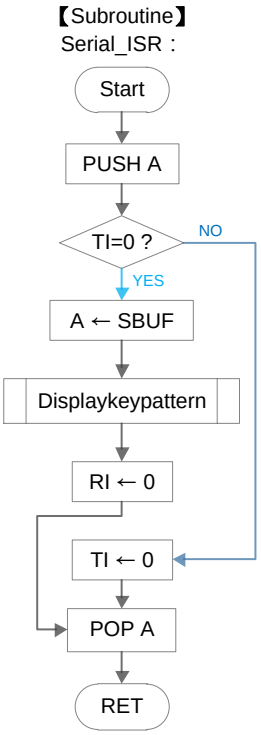
▲DelaykeyDebounce 流程圖



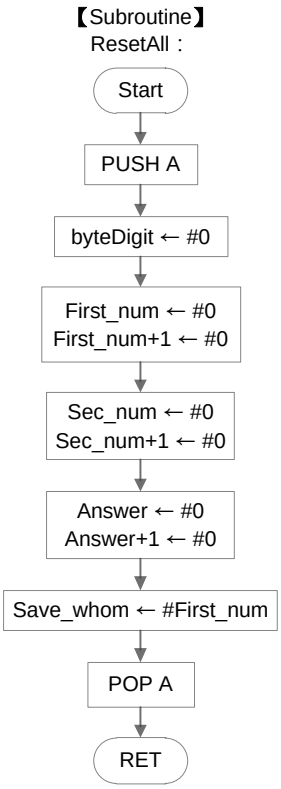
▲InitialSerial 流程圖



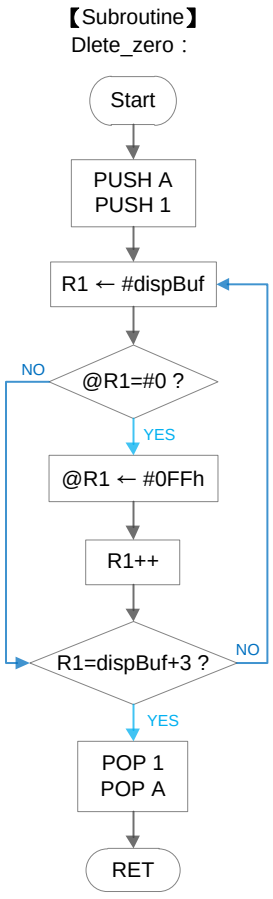
▲TxDString 流程圖



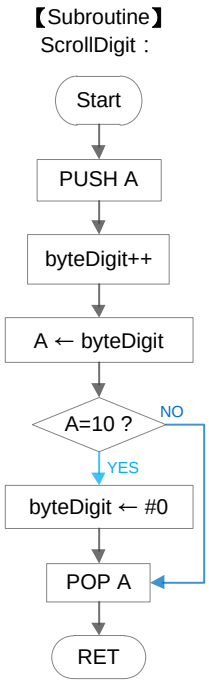
▲Serial_ISR 流程圖



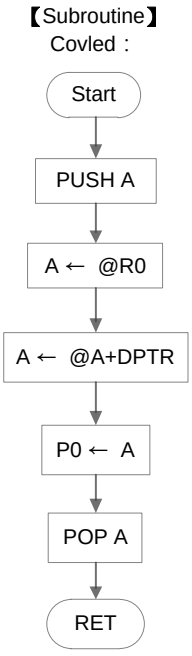
▲ResetAll 流程圖



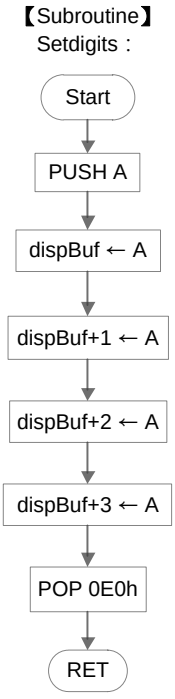
▲Delete_zero 流程圖



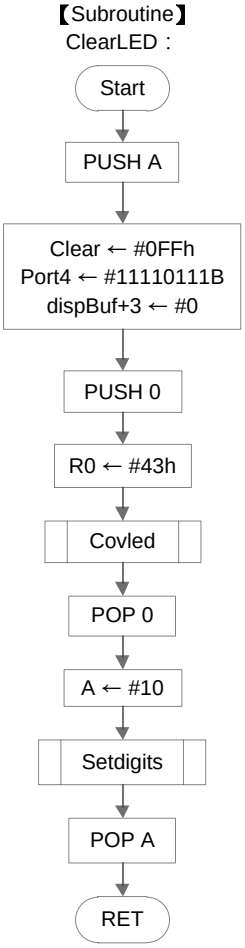
▲ScrollDigit 流程圖



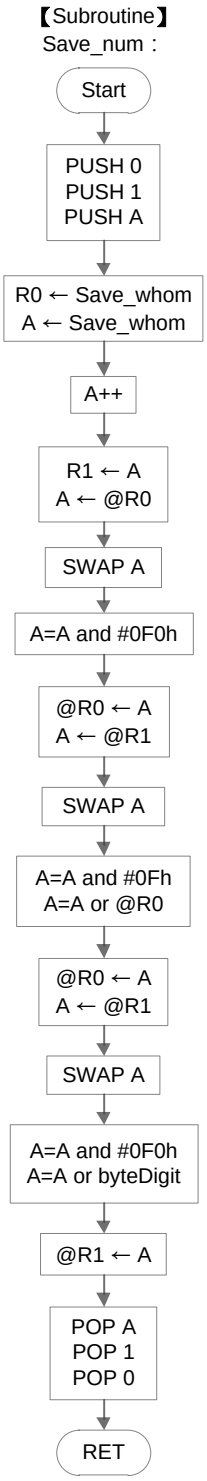
▲Covled 流程圖



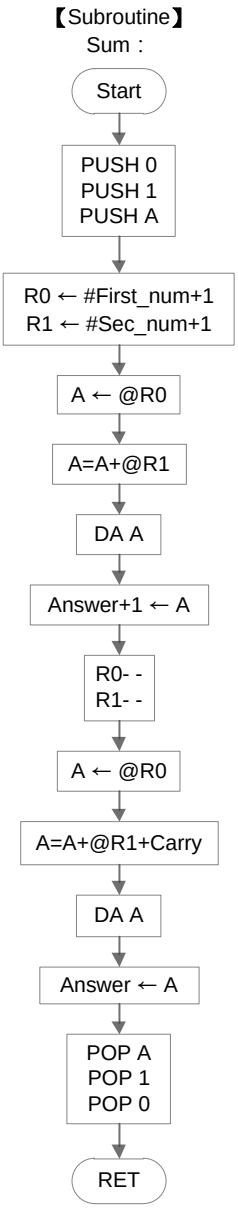
▲Setdigits 流程圖



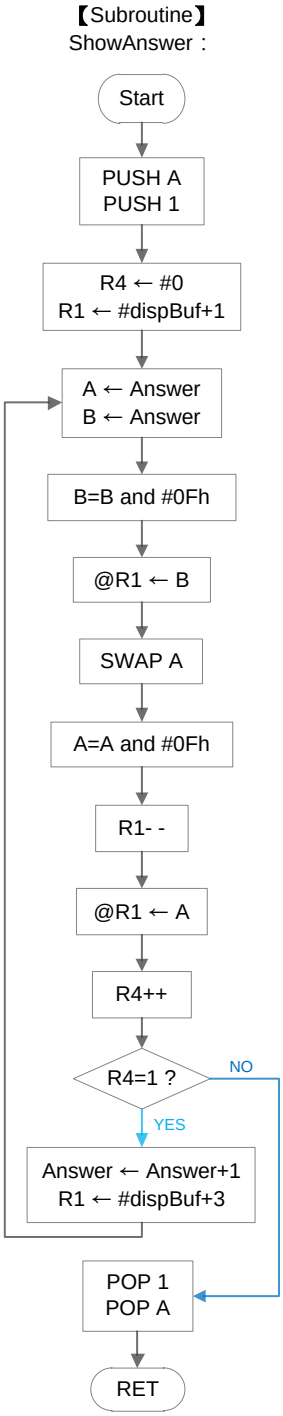
▲ClearLED 流程圖



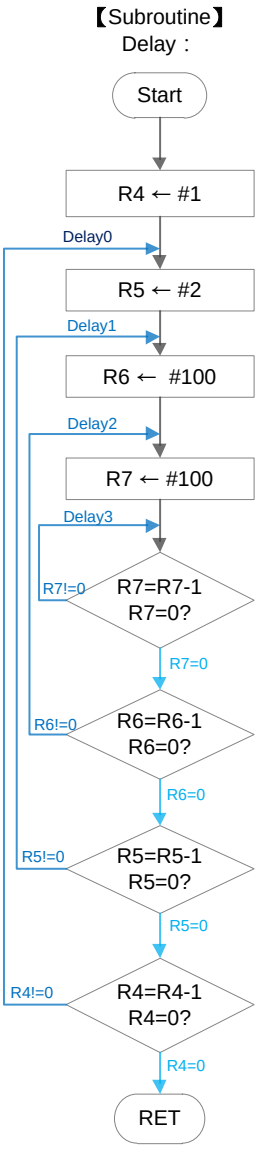
▲Save_num 流程圖



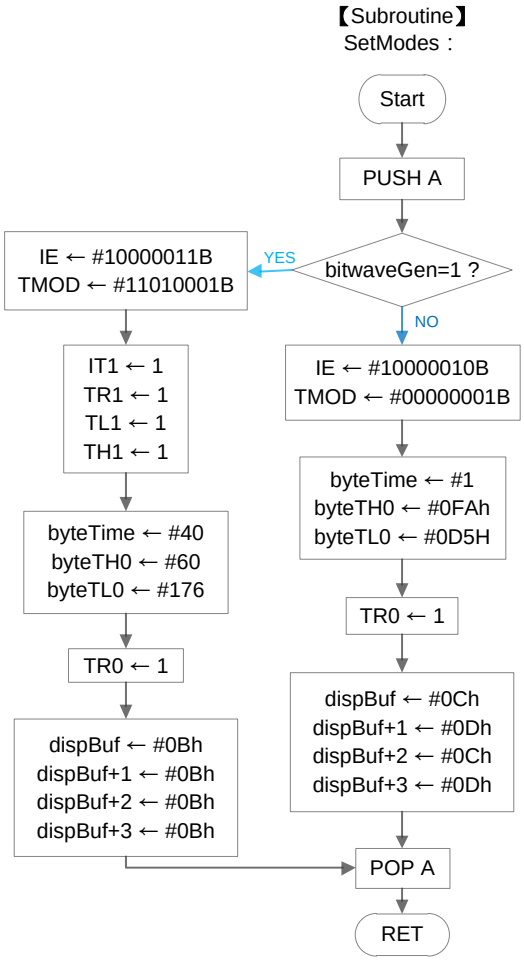
▲Sum 流程圖



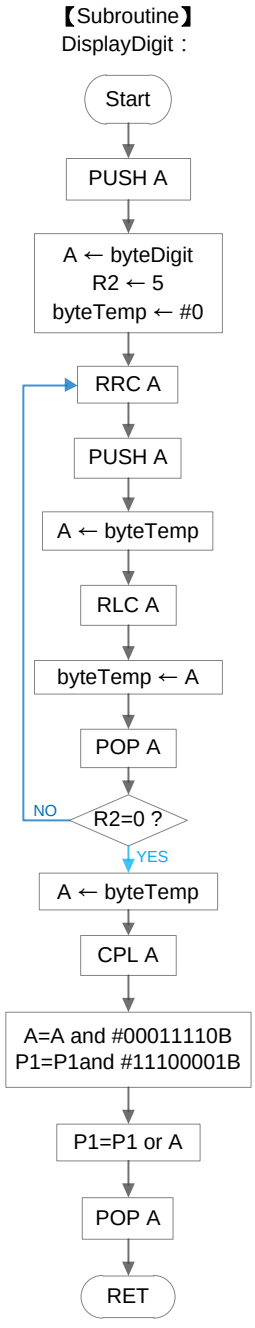
▲ShowAnswer 流程圖



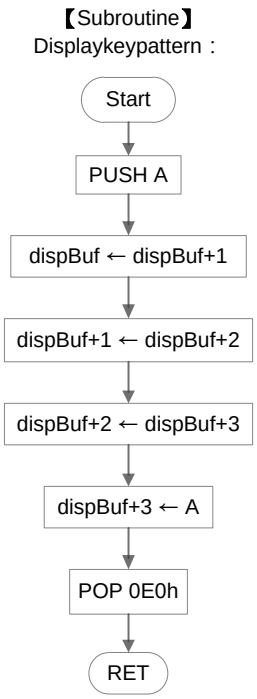
▲Delay 流程圖



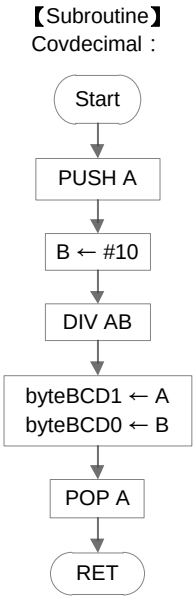
▲SetMode 流程圖



▲DisplayDigit 流程圖



▲Displaykeypattern



▲Covdecimal 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 0FFh  ;無字型
           DB 0BFh    ;-
           DB 0F7h    ;_
          DB 0C8h    ;冂
```

▲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
t0Buf           EQU     49h     //用來存取 t0Count,執行 DJNZ 造成 Delay
t0Count         EQU     40      //40*0.025sec=1sec,使 Timer0 產生 1s
byteBCD0        EQU     4Dh     //存取要轉換後之數值的個位.十位十進制)
byteBCD1        EQU     4Eh     //存取要轉換後之數值的百位.千位(十進制)
byteBCD2        EQU     4Fh     //存取要轉換後之數值的萬位十萬位(十進制)
byteHex1        EQU     50h     //存取要轉換之數值的最高兩位(十六進制)
byteHex0        EQU     51h     //存取要轉換之數值的最低兩位(十六進制)
byteTime        EQU     52h     //控制 Timer0 時間,1 時為傳送頻率,40 為接收
byteTH0         EQU     53h     //在 Wave 和 Meas 模式下 TH0 不同
byteTL0         EQU     54h     //在 Wave 和 Meas 模式下 TL0 不同
                                //用來存不同模式時 TH0.TL0 該給多少
bitwaveGen      BIT     P1.5    //表示現在是否為 Wave 模式(傳送自己學號頻率)
bitTimingGate   BIT     P1.0    //用來產生 1s 亮 1s 暗,或是亮自己的學號頻率
byteDigit       EQU     55h     //用來數現在滾輪數字按到多少
byteTemp        EQU     56h     //在 DisplayDigit 中協助做 P1.1~P1.4 顯示
First_num       EQU     57h     //57h~58h 用來存輸入的第一個數值
Sec_num         EQU     59h     //59h~5Ah 用來存輸入的第二個數值
Answer          EQU     5Bh     //5Bh~5Ch 用來存相加之值
Save_whom       EQU     5Dh     //用來記住現在該存取數值 1 還是數值 2
Clear           EQU     5Eh     //判斷是否要執行 clear
```

▲**Table 1** 此為記憶體定義。

```
ORG 0000h                          //從位置 0000h 開始
;===========================================
          LJMP start               //跳至標籤 Start 繼續
          ORG 000bh                //從位置 000Bh 開始
          LJMP T0_isr              //跳至標籤 T0_isr 繼續
          ORG 0013h                //從位置 0013h 開始
          LJMP int1_isr            //跳至標籤 Int1_isr 繼續
          ORG 0023h                //從位置 0023h 開始
          LJMP Serial_ISR          //跳至標籤 Serial_ISR 繼續
Start:
          SETB P3.0                //將 P3.0 設為 Input Mode
          SETB P3.3                //將 P3.3 設為 Input Mode
          SETB P3.5                //將 P3.5 設為 Input Mode

          SETB bitwaveGen          //關閉 P1.5 的燈
```

▲**Table 2** 此為 Main 前的宣告。

```
;-------------------------------------------------------------------
;【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             //將 A 初始化為 0FFh
          MOV P1,#0FFh             //將 P1 初始化為 0FFh(無亮燈)
          MOV P2, A                //將 P2 初始化為 0FFh(無輸入)
          MOV P3, A                //將 P3 初始化為 0FFh(無輸入)

          MOV keyBuf1, #0ffh       //將 keyBuf1(暫存 P2)初始化為 0FFh
          MOV keyBuf2, #0ffh       //將 keyBuf2(暫存 P3)初始化為 0FFh
          MOV keyCode, #0ffh       //初始化 keyCode 為 0FFh
          MOV keyCheck, #0ffh      //初始化 keyCheck 為 0FFh
          MOV debCounter, #0       //初始化 debCounter 為 0FFh
                                   //(檢查跳躍信號)
          MOV dispBuf, #10         //給 dispBuf 為偏移量 10(無字型)
          MOV dispBuf+1, #10       //給 dispBuf+1 為偏移量 10(無字型)
          MOV dispBuf+2, #10       //給 dispBuf+2 為偏移量 10(無字型)
          MOV dispBuf+3, #0        //給 dispBuf+3 為偏移量 0(字型 0)
```

▲**Table 3** 此為主程式 Main(上半)。

```
            ACALL ClearLED        //呼叫副程式 ClearLED 清空 Buffer
            ACALL ResetAll        //呼叫副程式 ResetAll 初始化其他數值

            MOV byteDigit,#0      //初始化 byteDigit 為 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 ClearLED          //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?
            ACALL DisplayDigit    //將 ByteDigit 轉為燈號 P1.1~P1.4 顯示

            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 4** 此為主程式 Main(下半)。

```
;-------------------------------------------------------------
;【Subroutine】 INT1_ISR
//外部中斷 1 用量測方波頻率
//當 0→1 時啟動,1→0 時停止計數
//之後再將得到之 16 進制數值做轉換得到 10 進制的方波頻率
;-------------------------------------------------------------
INT1_ISR:
          PUSH 0e0h                 //進入副程式前先 PUSH A
          MOV byteHex0, TL1         //將得到之 TL1 給 byteHex0 做存取
          MOV byteHex1, TH1         //將得到之 TH1 給 byteHex1 做存取
          ACALL hex2bcd             //呼叫 hex2bcd 將 16 進制數值轉 10 進制
          ACALL bcd2display         //將得到之 10 進制數值做輸出顯示
          MOV TL1, #0               //重置 TL1 為 0
          MOV TH1, #0               //重置 TH1 為 0
          POP 0e0h                  //因先前有 PUSH A,所以要 POP A
          RETI                      //返回
```

▲**Table 5** 此為副程式 INT1_ISR。

```
;-------------------------------------------------------------
;【Subroutine】 T0_isr
//用來產生特定的方波頻率
//當是 Meas 模式時,產生 1s 亮 1s 暗的方波頻率
//當是 Wave 模式時,產生自己的學號方波頻率
;-------------------------------------------------------------
T0_isr:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          CLR TR0                   //停止 Timer0 運作
          MOV TH0, byteTH0          //將 byteTH0 之值給 TH0
          MOV TL0, byteTL0          //將 byteTL0 之值給 TL0
          SETB TR0                  //啟動 Timer0
          DJNZ t0Buf, t0isr_exit    //若 T0Buf!=0,表尚未執行完,離開
          MOV t0Buf, byteTime       //若=0,重置 t0Buf 為 byteTime
          ;====================== 時間到，Do something
          CPL bitTimingGate         //將 Signal 顯示做 CPL
T0isr_exit:
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RETI                      //返回
```

▲**Table 6** 此為副程式 T0_ISR。

```
;-------------------------------------------------------------
;【Subroutine】Hex2bcd
//BCD(byteBcd2, byteBcd1, byteBcd0) ← Hex(byteHex1, byteHex0)
//將 16 進制轉換成 10 進制
;-------------------------------------------------------------
Hex2bcd:
          PUSH 0e0h                 //進入副程式前先 PUSH A
          MOV byteBcd0, #0          //先初始化 byteBcd0 為 0
          MOV byteBcd1, #0          //先初始化 byteBcd1 為 0
          MOV byteBcd2, #0          //先初始化 byteBcd2 為 0
          ;-------------------------------------------------
          ; byteHex0 -> BCD -> (byteBcd1, byteBcd0)
          ;-------------------------------------------------
          MOV A, byteHex0           //byteHex0 值給 A(16 進制)
          MOV B, #100               //將 B 給值為 100
          DIV AB                    //A 除 B 等於 A 餘 B(用除法轉成 10 進制)
          MOV byteBcd1, a           //將百位數給 bytrBcd1 存
          MOV A, B                  //將餘數 B 給 A
          MOV B, #10                //將 B 給值為 10
          DIV AB                    //A 除 B 等於 A 餘 B(用除法轉成 10 進制)
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ORL A, B                  //A 和 B 去做 or,結果存於 A
          MOV byteBcd0, a           //將十位數.個位數給 bytrBcd0 存
          ;-------------------------------------------------
          ; byteHex1 -> BCD ->(byteBcd2, byteBcd1, byteBcd0)
          ; 處理 (56 + 200) ，留意 BCD
          ;-------------------------------------------------
          MOV A, byteHex1           //byteHex1 值給 A(16 進制)
          JZ Hex2bcd_exit           //若 A=0,表示無百位數,跳至離開
          MOV R2, A                 //將 A 值給 R2
          MOV A, byteBcd0           //再將 byteBCD0 之值給 A 存取
Hex2bcd_loop1:
          ADD A, #56h               //A=A+56
          DA A                      //將 A 值結果轉換成 10 進制
          JNC hex2bcd_skip1         //檢查是否有 carry 的產生,若無,跳
          MOV R1, #byteBcd1         //若有 carry,R1 存取 byteBcd1
```

▲**Table 7** 此為副程式 Hex2bcd(上半)。

```
            MOV B, #1                   //將 B 給值為 1
            ACALL bcd_daa               //呼叫副程式 Bcd_daa 做進位處理
Hex2bcd_skip1:
            MOV R1, #byteBcd1           //將 R1 給值為 byteBcd1 的地址
            MOV B, #2                   //將 B 給值為 2(256 的百位數)
            ACALL bcd_daa               //呼叫副程式 Bcd_daa 做進位處理
            DJNZ R2, hex2bcd_loop1      //若 R2!=0,跳至 hex2bcd_loop1 繼續
            MOV byteBcd0, A             //將最後得到的 A 值給 byteBcd0
Hex2bcd_exit:
            POP 0E0h                    //因先前有 PUSH A,所以要 POP A
            RET                         //返回
```

▲**Table 8** 此為副程式 Hex2bcd(下半)。

```
;-------------------------------------------------------------------
;【Subroutine】 Bcd_daa
//利用@R1 去暫存位置做百位數(200+56 的 200)
;-------------------------------------------------------------------
Bcd_daa:
            PUSH 0e0h                   //進入副程式前先 PUSH A
            MOV A, @R1                  //取 R1 的內容為地址,該地址的內容給 A
            ADD A, B                    //A=A(原數值)+B(進位)
            DA A                        //將 A 值結果轉換成 10 進制
            MOV @R1, A                  //將 A 值存到 R1 的內容當地址
            JNC bcd_daa_skip            //若沒有 carry 的產生,跳至離開
            INC R1                      //R1++(換處理下個暫存器)
            INC @R1                     //換處理下個暫存器
bcd_daa_skip:
            POP 0e0h                    //因先前有 PUSH A,所以要 POP A
            RET                         //返回
```

▲**Table 9** 此為副程式 Bcd_daa。

```
;-----------------------------------------------------------------
;【Subroutine】Bcd2display
//將最後得到的十進位值在這裡做輸出顯示
//(dispBuf, dispBuf+1)←byteBcd1, (dispBuf+2, dispBuf+3)←byteBcd0
;-----------------------------------------------------------------
Bcd2display:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          MOV A, byteBcd1           //將 byteBcd1 之值給 A
          PUSH 0E0h                 //因為會再需要所以 PUSH A
          ANL A, #0fh               //留下 low byte 先處理百位
          MOV dispBuf+1, A          //將百位數字給 dispBuf+1 存取
          POP 0E0h                  //將先前的 PUSH 去 POP 出來
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ANL A, #0fh               //留下 low byte 先處理千位
          MOV dispBuf, A            //將千位數字給 dispBuf 存取

          MOV A, byteBcd0           //將 byteBcd0 之值給 A
          PUSH 0e0h                 //因為會再需要所以 PUSH A
          ANL A, #0fh               //留下 low byte 先處理個位
          MOV dispBuf+3, A          //將個位數字給 dispBuf+3 存取
          POP 0e0h                  //將先前的 PUSH 去 POP 出來
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ANL A, #0fh               //留下 low byte 先處理十位
          MOV dispBuf+2, A          //將十位數字給 dispBuf+2 存取

          POP 0e0h                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 10** 此為副程式 Bcd2display。

```
;-----------------------------------------------------------------
;【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 0E0h             //進入副程式前先 PUSH A
          MOV A, P2             //初始化 TL1 為 1
          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 A, #0FFh          //將 A 給值為 0FFh
          MOV keyBuf2, A        //將 keyBuf2 存取 P3(8~F)給值為 0FFh
;============================
          MOV A, keyBuf2        //將 keyBuf2(存取 8~F 輸入)給 A 做以下處理
          CJNE A,#0FFh,nextcode0//若 A 不是 FF(零輸入),跳至 nextcode0

          MOV A, keyBuf1        //將 keyBuf1(存取 0~7 輸入)給 A 做以下處理
          MOV R2, #0            // R2 用來數按鍵碼為何
nextcode1:
          JNB 0E0h, 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,nextcode1//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
nextcode0:
          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
nextcode2:
          JNB 0E0h, 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,nextcode2//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
```

▲**Table 11** 此為副程式 Getkeycode(上半)。

```
exitgetkeycode:
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 12** 此為副程式 Getkeycode(下半)。

```
;-----------------------------------------------------------------
;【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,判斷按鍵碼為何,並按照該按鍵碼的功能去撰寫內容
;-----------------------------------------------------------------
Delaykeydebounce:
          PUSH 0e0h                 //進入副程式前先 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,#debTimes,delayexitt//若 A 還未達到#debTimes,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, havekey    //若 A!=#0FFh 跳至 Here
Delayexitt:
          LJMP delayexit            //跳至 delayexit 離開副程式
havekey:
          CJNE A,keyCheck,unmatchkeyy//若 A!=keycheck,表示取到跳躍信號
                                     //跳至 unmatchkeyy

          CJNE A,#0,NOT_0           //按鍵碼 A 若!=0 跳至 NOT_0 (按鍵 Input)
          ACALL ScrollDigit         //呼叫副程式 ScrollDigit 做滾輪數字顯示
          LJMP Turn_off             //跳至 Turn_off 關閉 Timer 並離開副程式
Unmatchkeyy:
          LJMP unmatchkey           //跳至 unmatchkey 離開此副程式

NOT_0:    CJNE A,#1,NOT_1           //按鍵碼 A 若!=1 跳至 NOT_1 (按鍵 Enter)
          MOV A,byteDigit           //將 byteDigit 之值給 A 做使用
          ACALL Displaykeypattern //呼叫副程式 Displaykeypattern
          ACALL Save_num            //呼叫副程式 Save_num 去做數值存取
```

▲**Table 14** 此為副程式 DelatkeyDebounce - 1。

```
            MOV Clear,#0              //給 Clear 值為 0,表不啟動 Clear 功能
            LJMP Turn_off             //跳至 Turn_off 關閉 Timer 並離開副程式

NOT_1:      CJNE A,#2,NOT_2           //按鍵碼 A 若!=2 跳至 NOT_2 (按鍵 CLR)
            SETB P1.7                 //將 P1.7 的燈關閉
            ACALL ClearLED            //呼叫 Clear 副程式做 LED 顯示空空空 0
            ACALL ResetAll            //呼叫副程式 ResetAll 去初始化其它數值
            LJMP Turn_off             //跳至 Turn_off 關閉 Timer 並離開副程式

NOT_2:      CJNE A, #3, NOT_3         //按鍵碼 A 若!=3 跳至 NOT_3 (按鍵 Wave)
            SETB P1.7                 //將 P1.7 的燈關閉
            CLR bitwaveGen            //使 P1.5 亮起 (Wave 燈號)
            SETB P1.6                 //將 P1.6 的燈關閉
            ACALL SetModes            //呼叫副程式 SetMode 處理 Wave 模式
            MOV Clear,#0              //給 Clear 值為 0,表不啟動 Clear 功能
            LJMP key_reset            //跳至 key_reset 重置 key 並離開此副程式

NOT_3:      CJNE A,#4,NOT_4           //按鍵碼 A 若!=4 跳至 NOT_4 (按鍵 OP)
            MOV Save_whom,#Sec_num    //將 Save_whom 給值為
            MOV C,P1.7                //將 P1.7 之值給 C 去暫存
            MOV P1,#0FFh              //將 P1 的 LED 燈顯示全關
            MOV P1.7,C                //再將 C 之值還給 P1.7 去做顯示
            MOV byteDigit,#0          //重置 byteDigit 值為 0
            ACALL ClearLED            //呼叫 Clear 副程式做 LED 顯示空空空 0
            SJMP key_reset            //跳至 key_reset 重置 key 並離開此副程式

NOT_4:      CJNE A,#5,NOT_5           //按鍵碼 A 若!=5 跳至 NOT_5 (按鍵=)
            ACALL Sum                 //呼叫副程式 Sum 去做輸入兩數值的相加
            ACALL ShowAnswer          //呼叫副程式 ShowAnswer 去做結果顯示
            ACALL Delete_zero         //呼叫副程式 Delete_zero 清除高位的 0
            JB P1.7,Of                //若 P1.7 為關,跳至 Of
            ACALL InitialSerial       //呼叫 InitialSerial 開啟串列通訊
            ACALL TxDString           //呼叫副程式 TxDString 做傳送訊號
            SJMP Reseting             //跳至 Reseting 繼續其它初始化
Of:         MOV SCON,#0               //將 SCON 給值為 0,關閉串列通訊
```

▲**Table 15** 此為副程式 DelatkeyDebounce - 2。

```
Reseting: MOV C,P1.7              //將 P1.7 之值給 C 去暫存
          MOV P1,#0FFh            //將 P1 的 LED 燈顯示全關
          MOV P1.7,C              //再將 C 之值還給 P1.7 去做顯示
          MOV byteDigit,#0        //重置 byteDigit 值為 0
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式

NOT_5:    CJNE A, #6, NOT_6       //按鍵碼 A 若!=6 跳至 NOT_6(按鍵 COM)
          MOV Clear,#0            //給 Clear 值為 0,表不啟動 Clear 功能
          CPL P1.7                //將 P1.7 做 CPL
          JB P1.7,Off             //若 P1.7 為關,跳至 Off
          ACALL InitialSerial     //呼叫 InitialSerial 開啟串列通訊
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式
Off:      MOV SCON,#0             //將 SCON 給值為 0,關閉串列通訊
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式

NOT_6:    CJNE A, #7, NOT_7       //按鍵碼 A 若!=7 跳至 NOT_7(按鍵 Meas)
          SETB bitwaveGen         //將 P1.5 的燈關閉(Wave 燈號)
          CLR P1.6                //讓 P1.6 的燈亮起
          SETB P1.7               //將 P1.7 的燈關閉
          ACALL setModes          //呼叫副程式 SetMode 處理 Meas 模式
          MOV Clear,#0            //給 Clear 值為 0,表不啟動 Clear 功能
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式

NOT_7:    CJNE A, #8, NOT_8       //按鍵碼 A 若!=8 跳至 NOT_8
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式

NOT_8:    CJNE A, #9, key_reset   //按鍵碼 A 若!=9 跳至 NOT_9
          SJMP key_reset          //跳至 key_reset 重置 key 並離開此副程式

Turn_off: CLR TR1                 //關閉 Timer1
          CLR TR0                 //關閉 Timer0
          CLR IT1                 //關閉外部中斷 1
          ORL P1,#01100001B       //維持原本滾輪數字和訊號燈,其它關閉
Key_reset:
          MOV keyCode, #0ffh      //重設 keyCode 值為 0FFh
```

▲**Table 16** 此為副程式 DelatkeyDebounce - 3。

```
            MOV keyCheck, #0ffh     //重設 keyCheck 值為 0FFh
            SJMP delayexit          //跳至 delayexit 離開此副程式
unmatchkey:
            MOV keyCheck, keyCode   //若按鍵為跳躍信號,keyCode 給 keyCheck
delayexit:
            POP 0e0h                //因先前有 PUSH A,出副程式前必須 POP A
            RET                     //返回
```

▲**Table 17** 此為副程式 DelatkeyDebounce - 4。

```
;-----------------------------------------------------------------
;【Subroutine】InitialSerial
//為開啟串列通訊做設定
//其中 Timer1 負責處理 baud rate
;-----------------------------------------------------------------
InitialSerial:
            MOV IE,#10010000B       //開啟串列通訊
            MOV TMOD ,#00100000B    //開啟 Timer1 (用來處理 baud rate)
            MOV SCON ,#01010000B    //將 SCON 設定為可以接收.傳送
            MOV TH1,#-6             //設置 baud rate 為 4800
            MOV A,PCON              //將 PCON 之值給 A 暫存
            SETB ACC.7              //為了產生兩倍 baud rate
            MOV PCON,A              //將 A 還給 PCON
            SETB P3.0               //將 P3.0 設為 Input Mode
            SETB TR1                //啟用 Timer1
            RET                     //返回
```

▲**Table 18** 此為副程式 InitialSerial。

```
;-------------------------------------------------------------------
;【Subroutine】 TxDString
//此副程式負責做傳送訊號
//其中 TI 的用意為檢查是否完成傳送,避免資料覆蓋
;-------------------------------------------------------------------
TxDString:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          MOV R2,#4                 //將 R2 給值為 4
          MOV R1,#dispBuf           //R1 用來記 dispBuf 的地址
NotOver:  MOV SBUF,@R1              //將 dispBuf 裡面之值傳送到 SBUF
Zero:     JNB TI,Zero               //檢查是否傳送完成(若完成,TI=1)
          INC R1                    //換下一個 dispBuf 做傳送
          DJNZ R2,NotOver           //若尚未傳完 4 個 dispBuf,跳至 NotOver
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 19** 此為副程式 TxDString。

```
;-------------------------------------------------------------------
;【Subroutine】 Serial_ISR
//進入這個副程式時,要先檢查是 TI 觸發還是 RI 觸發
//若是 TI 觸發,重製 TI 出迴圈,這裡主要負責接收
;-------------------------------------------------------------------
Serial_ISR:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          JB TI,Hereee              //若 TI=1,跳到 Hereee
          MOV A,SBUF                //將 SBUF 所得到的內容給 A
          ACALL Displaykeypattern   //呼叫 Displaykeypattern 做顯示輸出
          CLR RI                    //清除 RI 為 0
          SJMP ExitSerial           //跳至 ExitSerial 離開此程式
Hereee:   CLR TI                    //清除 TI 為 0
ExitSerial:
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RETI                      //返回
```

▲**Table 20** 此為副程式 Serial_ISR。

```
;-------------------------------------------------------------
;【Subroutine】ClearLED
//主要負責將 dispBuf 顯示空空空 0,並設法讓第一次輸入不會將 0 推過去
;-------------------------------------------------------------
ClearLED: PUSH 0E0h                  //進入副程式前先 PUSH A
          MOV Clear, #0FFh           //將 Clear 給值為 0FFh,表示啟動 Clear
          MOV Port4,#11110111B       //設定只讓最右 LED 亮
          MOV dispBuf+3, #0          //將 diapBuf+3(最右 LED)給值為 0
          PUSH 0                     //因為會用到 R0,所以先 PUSH 0
          MOV R0,#43h                //將 R0 給值為 43h
          ACALL Covled               //呼叫副程式 Covled 做輸出顯示
          POP 0                      //將 R0 的值 POP 回來
          MOV A,#10                  //將 A 給值為 10,字型為空
          ACALL Setdigits            //呼叫 Setdigits 將該值給每個 dispBuf
          POP 0E0h                   //因先前有 PUSH A,所以要 POP A
          RET                        //返回
```

▲**Table 21** 此為副程式 ClearLED。

```
;-------------------------------------------------------------
;【Subroutine】ResetAll
//這裡主要負責將存數值相關的暫存器做初始化,為下次運算做準備
;-------------------------------------------------------------
ResetAll: PUSH 0E0h                  //進入副程式前先 PUSH A
          MOV byteDigit,#0           //初始化 byteDigit 給值為 0
          MOV First_num,#0           //初始化 First_num 給值為 0
          MOV First_num+1,#0         //初始化 First_num+1 給值為 0
          MOV Sec_num,#0             //初始化 Sec_num 給值為 0
          MOV Sec_num+1,#0           //初始化 Sec_num+1 給值為 0
          MOV Answer,#0              //初始化 Answer 給值為 0
          MOV Answer+1,#0            //初始化 Answer+1 給值為 0
          MOV Save_whom,#First_num//將 Save_whom 給值為 First_num
          POP 0E0h                   //因先前有 PUSH A,所以要 POP A
          RET                        //返回
```

▲**Table 22** 此為副程式 ResetAll。

```
;-------------------------------------------------------------
;【Subroutine】Save_num
//此副程式主要做數值的存取(利用 packed 做存取)
//由 Save_num 記住現在該存取的為數值 1 還是數值 2
//數值 1 存於 57h.58h, 數值 2 存於 59h.5Ah
;-------------------------------------------------------------
Save_num:
          PUSH 0                    //因為會用到 R0,因此先 PUSH R0
          PUSH 1                    //因為會用到 R1,因此先 PUSH R1
          PUSH 0E0h                 //進入副程式前先 PUSH A
          MOV R0,Save_whom          //將 Save_whom 之值給 R0 做處理
          MOV A,Save_whom           //將 Save_whom 之值給 A 做處理
          INC A                     //A=A+1(先處理較高位元)
          MOV R1,A                  //將 R1 給值為 A(值 1 的 58h 或值 2 的 5Ah)
          MOV A,@R0                 //取 R0 之內容當地址的內容
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ANL A,#0F0h               //留下 A 的 high byte
          MOV @R0,A                 //將左移後的值放回去
          MOV A,@R1                 //取 R1 之內容當地址的內容給 A
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ANL A,#0Fh                //留下 A 的 low byte
          ORL A,@R0                 //A 去 or R0 之內容當地址的內容
          MOV @R0,A                 //將得到的 A 還給 R0 之內容當地址的位置
          MOV A,@R1                 //取 R1 之內容當地址的內容給 A
          SWAP A                    //將 A 的 high byte 和 low byte 互換
          ANL A,#0F0h               //留下 A 的 high byte(此時處理個位.十位)
          ORL A,bytedigit           //A 去 or byteDigit
          MOV @R1,A                 //將左移後的值放回去
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          POP 1                     //因先前有 R1,所以要 POP R1
          POP 0                     //因先前有 R0,所以要 POP R0
          RET                       //返回
```

▲**Table 23** 此為副程式 Save_num。

```
;-------------------------------------------------------------------
;【Subroutine】Sum
//因為是用 packed 的方式做存取,因此兩數值可以直接做相加
//之後再數值用 DA A 轉乘 10 進制即可
;-------------------------------------------------------------------
Sum:        PUSH 0                  //因為會用到 R0,因此先 PUSH R0
            PUSH 1                  //因為會用到 R1,因此先 PUSH R1
            PUSH 0E0h               //進入副程式前先 PUSH A
            MOV R0,#First_num+1     //將 R0 給值為 First_num+1 的地址
            MOV R1,#Sec_num+1       //將 R1 給值為 Sec_num+1 的地址
            MOV A,@R0               //取 R0 之內容當地址的內容給 A
            ADD A,@R1               //將兩數值的最低位做相加
            DA A                    //轉換成 10 進制
            MOV Answer+1,A          //將結果存在 Answer+1
            DEC R0                  //R0=0-1
            DEC R1                  //R1=R1-1
            MOV A,@R0               //取 R0 之內容當地址的內容給 A
            ADDC A,@R1              //將兩數值的最高位連同 carry 做相加
            DA A                    //轉換成 10 進制
            MOV Answer,A            //將結果存在 Answer
            POP 0E0h                //因先前有 PUSH A,所以要 POP A
            POP 1                   //因先前有 R1,所以要 POP R1
            POP 0                   //因先前有 R0,所以要 POP R0
            RET                     //返回
```

▲**Table 24** 此為副程式 Sum。

```
;-----------------------------------------------------------------
;【Define】 Delete_zero
//將高位沒用的 0 刪除
//Ex0012 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,#dispBuf            //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,#dispBuf+3,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 25** 此為副程式 Delete_zero。

```
;-----------------------------------------------------------------
;【Subroutine】ScrollDigit
//數 byteDigit 值為多少
//因為最大只能輸入到 9,若超過 9 須重置 byteDigit 為 0
;-----------------------------------------------------------------
ScrollDigit:
          PUSH 0E0h                  //進入副程式前先 PUSH A
          INC byteDigit              //byteDigit++
          MOV A,byteDigit            //將 byteDigit 值給 A
          CJNE A,#10,Go              //若 A!=10,跳至 Go 離開此副程式
          MOV byteDigit,#0           //若 A=10,重置 byteDigit 為 0
Go:       POP 0E0h                   //因先前有 PUSH A,所以要 POP A
          RET                        //返回
```

▲**Table 26** 此為副程式 ScrollDigit。

```
;-------------------------------------------------------------
;【Subroutine】ShowAnswer
//使用 packed 去做存取
//Ex 43h.42h 存 Answer+1,41h.40h 存 Answer
;-------------------------------------------------------------
ShowAnswer:
           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 R4,#0                 //將 R4 給值為 0,數做完傳送了沒
           MOV R1,#dispBuf+1         //R1 記住 dipBuf+1 的位置
Not_over:  MOV A,Answer              //將 Answer 目前存的值給 A
           MOV B,Answer              //將 Answer 目前存的值給 B
           ANL B,#0Fh                //只留下 B 的 low byte
           MOV @R1,B                 //將 low byte 轉給 41h 或 43h
           SWAP A                    //將 A 的 high byte 與 low byte 互換
           ANL A,#0Fh                //只留下 A 的 low byte
           DEC R1                    //R1=R1-1,此時存為 40h 或 42h
           MOV @R1,A                 //將 A 值存到取 R1 之內容當地址的位置
           INC R4                    //R4-R4+1,數處理完 dispBuf 了沒
           CJNE R4,#1, OK            //R4!=1,表完成 dispBuf 的存值,跳至 OK
           MOV Answer,Answer+1       //將 Answer+1 之值給 Answer
           MOV R1,#dispBuf+3         // R1 記住 dipBuf+3 的位置
           SJMP Not_over             //跳至 Not_over 繼續處理
OK:        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 27** 此為副程式 ShowAnswer。

```
;-------------------------------------------------------------------
;【Subroutine】DisplayDigit
//此程式用來處理滾輪數字之 Binary 顯示(P1.1~P1.4)
;-------------------------------------------------------------------
DisplayDigit:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          MOV A,byteDigit           //將 byteDigit 之值給 A 做存取
          MOV R2,#5                 //將 R2 給值為 5
          MOV byteTemp,#0           //初始化 byteTemp 為 0
R2_NOT_0: RRC A                     //利用 carry 做暫存,將 A 向右旋
          PUSH 0E0h                 //先將 A 值 PUSH 暫存起來
          MOV A,byteTemp            //將 byteTemp 值給 A
          RLC A                     //利用 carry 做暫存,將 A 向左旋進去
          MOV byteTemp,A            //將 A 值給給 byteTemp 做儲存
          POP 0E0h                  //將原數值 POP 出來
          DJNZ R2,R2_NOT_0          //R2!=0,表尚未處理完 P1,較至 R2_NOT_0
          MOV A,byteTemp            //將 byteTemp 之值給 A 做處理
          CPL A                     //將 A 值做 CPL
          ANL A,#00011110B          //留 P1.1~P1.4 以外的燈號顯示
          ANL P1,#11100001B         //留 P1.1~P1.4 以外的燈號顯示
          ORL P1,A                  //將 P1 和 A 做 or 的結果存於 1
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 28** 此為副程式 DisplayDigit。

```
;-------------------------------------------------------------------
;【Subroutine】Covdecimal
//將數值轉換成十進制(藉由 DIV AB)
//並將高位給 byteBCD1 存取, 低位給 byteBCD1 存取
;-------------------------------------------------------------------
Covdecimal:
          PUSH 0E0h                 //進入副程式前先 PUSH A
          MOV B, #10                //將 B 給值為 10
          DIV AB                    //進行 A/B,商數存在 A,餘數存在 B
          MOV byteBCD1, A           //將 A 值給 byteBCD1 記住
          MOV byteBCD0, B           //將 B 值給 byteBCD0 記住
          POP 0E0h                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 29** 此為副程式 Covdecimal。

```
;-----------------------------------------------------------------
;【Subroutine】 SetModes
//檢查現在是傳送學號頻率模式或是接收學號模式
//若是傳送學號頻率模式需顯示_ㄇ_ㄇ
//若是接收學號頻率模式須顯示----
;-----------------------------------------------------------------
SetModes:
          PUSH 0e0h              //進入副程式前先 PUSH A
          JB bitwaveGen, here    //若 bitwaveGen=1,跳至 here
          MOV IE, #10000010B     //此時 Wave 燈亮,啟用 Timer0
          MOV TMOD, #00000001B   //開啟 Timer0,Mode1
          MOV byteTime, #1       //將 byteTime 給值為 1
          MOV byteTH0, #0FAh     //將 byteTH0 給值為 FAh(學號計算結果)
          MOV byteTL0, #0D5h     //將 byteTL0 給值為 D5h(學號計算結果)
          SETB tr0               //啟動 Timer0

          MOV dispbuf, #0Ch      //給 dispBuf 為偏移量 C(_字型)
          MOV dispBuf+1, #0Dh    //給 dispBuf+1 為偏移量 D(ㄇ字型)
          MOV dispBuf+2, #0Ch    //給 dispBuf+2 為偏移量 C(_字型)
          MOV dispBuf+3, #0Dh    //給 dispBuf+4 為偏移量 D(ㄇ字型)
          SJMP exitsetmodes      //跳至 exitsetmodes 離開

Here:     MOV IE, #10000110B     //此時 Meas 燈亮,啟用 Timer0 和外部中斷 1
          MOV TMOD, #11010001B   //設置外部中斷 1 和 Timer0,Mode1
          SETB IT1               //啟動外部中斷 1
          SETB TR1               //啟動 Timer1
          MOV Tl1, #1            // 初始化 TL1 為 1
          MOV TH1, #1            //初始化 TH1 為 1
          MOV byteTime, #40      //將 byteTime 給值為 40(因 40*0.025s=1s)
          MOV byteTH0, #60       //將 byteTH0 給值為 60
          MOV byteTL0, #176      //將 byteTL0 給值為 176
          SETB TR0               //啟動 Timer0(產生 1s High 1s Low)

          MOV dispbuf, #0Bh      //將 dispBuf 給值為 0Bh(字型表第 B 個)
          MOV dispBuf+1, #0Bh    //將 dispBuf+1 給值為 0Bh(字型表第 B 個)
          MOV dispBuf+2, #0Bh    //將 dispBuf+2 給值為 0Bh(字型表第 B 個)
```

▲**Table 30** 此為副程式 SetModes(上半)。

```
          MOV dispBuf+3, #0Bh   //將 dispBuf+3 給值為 0Bh(字型表第 B 個)
Exitsetmodes:
          POP 0e0h              //因先前有 PUSH A,所以要 POP A
          RET                   //返回
```

▲**Table 31** 此為副程式 SetModes(下半)。

```
;-----------------------------------------------------------------
;【Subroutine】Displaykeypattern
//將 LED 向左傳遞做顯示
;-----------------------------------------------------------------
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, A           //將取得之按鍵碼給右一的 LED
           POP 0E0h                   //因先前有 PUSH A,所以要 POP A
           RET                        //返回
```

▲**Table 32** 此為副程式 Displaykeypattern。

```
;-----------------------------------------------------------------
;【Subroutine】Setdigits
//將 A 值給 dispBuf 至 dispBuf+3
;-----------------------------------------------------------------
Setdigits:
           PUSH 0E0h                  //進入副程式前先 PUSH A
           MOV dispBuf, A             //將 dispBuf 給值為 A
           MOV dispBuf+1, A           //將 dispBuf+1 給值為 A
           MOV dispBuf+2, A           //將 dispBuf+2 給值為 A
           MOV dispBuf+3, A           //將 dispBuf+3 給值為 A
           POP 0E0h                   //因先前有 PUSH A,所以要 POP A
           RET                        //返回
```

▲**Table 33** 此為副程式 Setdigits。

```
;-------------------------------------------------------------------
;【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 34** 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 35** 此為副程式 Covled。

```
;-------------------------------------------------------------
;【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  0FFh          //當偏移值為 10 時,取 0FFh,字型無,回去給 A
          DB  0BFh          //當偏移值為 11 時,取 0BFh,字型-,回去給 A
          DB  0F7h          //當偏移值為 12 時,取 0F7h,字型_,回去給 A
          DB  0C8h          //當偏移值為 13 時,取 0C8h,字型冂,回去給 A
;-------------------------------------------
          END               //結束程式
```

▲**Table 36** 此為副程式 LED_table。

四、 實作過程、心得、與結論（200 字以上）

8051 多功能加法器

實作過程：

在實作過程中，因為這次其實很多設計都是課堂上做過的實驗，因此也可以說很多的程式碼可以用拼湊的拼出來，雖然聽起來很簡單，但實際操作上有許多困難。我最一開始就是卡在我未注意到 EQU 有位置是重複定義的，造成程式在執行時產生不正常的現象。在撰寫加法時，因為我利用 packed 的方法做存取，因此會不斷用到 SWAP 以及 ANL，其實很容易旋到自己都不知道轉到哪了，後來我就用 R 暫存器去監看各數值，甚至還發現自己忘了初始化，怪不得加法出來總是錯的。

在充滿活動的大二生活中，要擠出數小時出來寫程式其實並不簡單，我大多都還是在半夜寫到凌晨才去睡，有時又會因為精神不好，容易心浮氣躁沉不住氣去找錯，在時間規劃上我發現我還有很大的進步空間，

心得&結論：

經歷上次的期中專題，我學到大致的 Debug，但卻也不算是完全學會，因為我總是將自己卡在迴圈或是不知道何時該按下 Megawin 的按鍵，原來其實紅點不該設在按鍵碼的判別式，否則每當自己每次按一個按鍵時都要不斷地也在電腦按 F10、F11 繼續執行，應該將紅點改設在會出問題的按鍵判別式下方，這樣就可以慢慢觀看當點下那個按鍵時那些數值會更動、會跑進哪些副程式……等，提升 Debug 效率，而在查詢地址那欄，不能單純的只打上 0x0040，而是要改成 D：0x0040，否則查到的並非地址，在學會這些以後，我的 Debug 效率也有了大大的進步，甚至也能夠幫同學 Debug，覺得很有收穫。

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，課堂剛教到時鐘信號時，我還覺得一個頭兩個大，覺得它很模糊也不好理解，但我想老師會要我們每行打上註解的原因也是這個吧！畢竟這次程式有一些現成的資源可以取用，很多人都會貼到不知道自己在做甚麼，再加上註解後，一瞬間覺得自己完全理解程式走每一行、每個副程式、和哪個 Timer 的用意了，這次的專題我得到的不單單只是課堂上的成績、Debug 的用法，也還有大大的成就感。

五、 參考文獻

無。