利用組合語言撰寫加法計算器

班級 電機 2B 學號 1050642 姓名 李鍇灝

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：李鍇灝（請打字輸入，以示負責）

一、 專題目的

利用 MAGAWIN 開發板子 MPC82G516 去實現加法計算器，必須了解 MAGAWIN-8051 單晶片的 Pin 連到 I/O 的架構，按鍵編排如圖 Figure 1，OP 為加號按鍵，CLR 為清除按鍵。而一開始進入計算器介面時 4 個 7-segment 應該為 blank/blank/blank/0，並藉由按鍵輸入第一個數字，然後按加號按鍵後可以輸入第二個數字，最後按下等於按鍵即可運算結果，如果結果溢位需要將 P1.0 的 LED 亮起來，相反的沒有溢位不需要亮，中途按下 CLR 會變成 blank/blank/blank/0。

7	8	9	OP
4	5	6	
1	2	3	
0		CLR	=

Figure 1 計算器按鍵配置圖

一開始可利用穩壓按鍵的程式與四個 7-segment 視覺暫留的程式為基礎下去更改，更改方式從一開始的定義記憶體位置，主程式初始值設定以及四個 7-segment 視覺暫留，然後穩壓按鍵後的按鍵功能判斷，設定每個按鍵功能，最後利用 Keil uVision3 開發環境傳入程式，最後 Debug 直到完成程式目的。

二、 詳細設計原理、記憶體定義、與完整流程圖

I/O 的架構如圖 Figure 2，Pin 4 是控制四個 7-segment 的共陽端，Pin 0 為控制 7-segment 的 8 個 LED，Pin 1 為控制 8 顆 LED 的，Pin 2 為控制編號 0-7 按鍵，Pin 3 為控制 8-F 按鍵。也就是把 P0/P1/P4 當作 output，P2/P3 當作 input，讀鍵盤的值經過運算給 7-segment 以及 LED 適當的值。

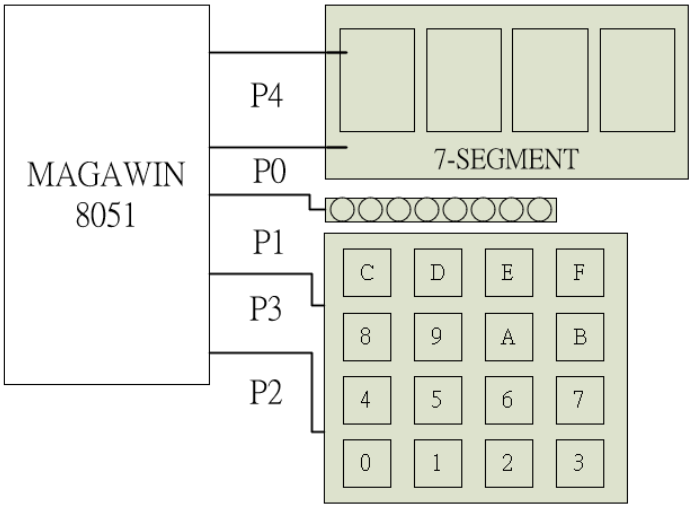

Figure 2 8051 Pin I/O 的架構

一開始必須定義記憶體的位值或是值給予使用，如圖 Table 1，總共定義 13 個記憶體位置(40H~48H 和 50H~53H)，2 個旗標用以及 3 個值用以使用，在一開始進入主程式必須將定義的記憶體位置以及做初始值設定包刮 Port2/Port3/Keybuf1/Keybuf2/Keycode/Keycheck 給定#0FFH 表示一開始沒有按鍵被 Push 的狀態，Debcounter 設成 0 從頭開始計數 Addbit 加號旗標做清除，把 Dispbuf 的 4 個記憶體設成初始狀態所以分別給 blank/blank/blank/0 然後 Addbuf 給定 0/0/0/0，並把 Carrybit 設成 1，最後也要將 DPTR 的地址給定 LED_TABLE 的地址初始化，如圖 Figure 2。

Table 1 記憶體位置以及值的定義

名稱 (Label)	位置或值	主要功能
PORT4	E8H	P4 原位置設置 Label
DISPBUF (+0~+3)	40H~43H	用來儲存 7-segment 4 個顯示值用以查表
KEYBUF1	44H	用來處理 0~7 按鍵 Input 的記憶體
KEYBUF2	45H	用來處理 8~F 按鍵 Input 的記憶體
KEYCODE	46H	用來儲存經由判斷完後哪個按鍵被 Push
KEYCHECK	47H	用來 double-check 是否為這個按鍵被 Push
DEBCOUNTER	48H	計數現在為第 N 個 Delay
ADDBUF (+0~+3)	50H~53H	用來儲存 4 個被加數用以做加法
CARRYBIT	P1.0	溢位時的旗標
ADDBIT	01H	當加號 OP 被按時的旗標
SHIFTLED	FEH	定義一開始 P4 的值為 0FEH，作為初始左旋顯示碼的作用
BLANK	FFH	定義 Blank 的值為 0FFH，作為給 P0 時 7-sgment Blank
DEBTIMES	60	定義每經過 N 個 Delay double-check 的時間，N=60

初始值設定完後，在主程式需做控制顯示 7-segment 循環顯示造成視覺暫留以一次顯示 4 個數字，

流程圖如 Figure 3，由 Reset 開始，重設 7-segment 顯示的值由第一個開始給以及 7-segment 由第一個開始亮，把依序啟動 7-segment 碼給 PORT4，呼叫點亮 7-segment 和取得按鍵是否被 PUSH，以及按鍵的功能，做完後，點亮下一個 7-segment，如果沒有超出第 4 個 7-segment 就迴圈向左移，如果超出就 Reset。

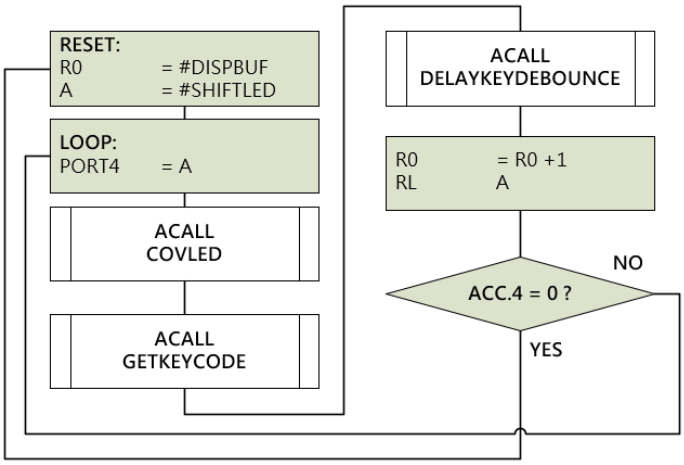

Figure 3 主程式 7-segment 向依序迴圈顯示

主程式完畢後由第一個副程式 Covled 開始寫，這個副程式只有不同的邏輯，如果需要顯示 Blank 那麼就直接給值，如果不是顯示 Blank 就使用查表之後給值，如圖 Figure 4，而下一個副程式須取得按鍵碼才可以做按鍵功能處理，Gotkeycode 流程圖如圖 Figure 5，分成 Port2 和 Port3 兩邊做處理(按鍵 0~7 以及 8~F)最後如果有讀到按鍵 Keycode 裡就會有值，否則直接離開程式。

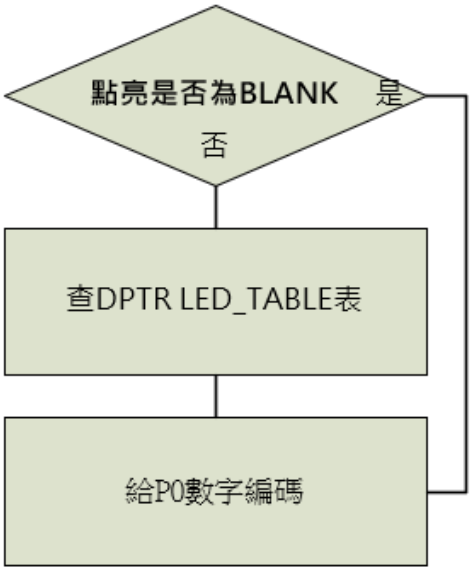

Figure 4 副程式 Covled 流程圖

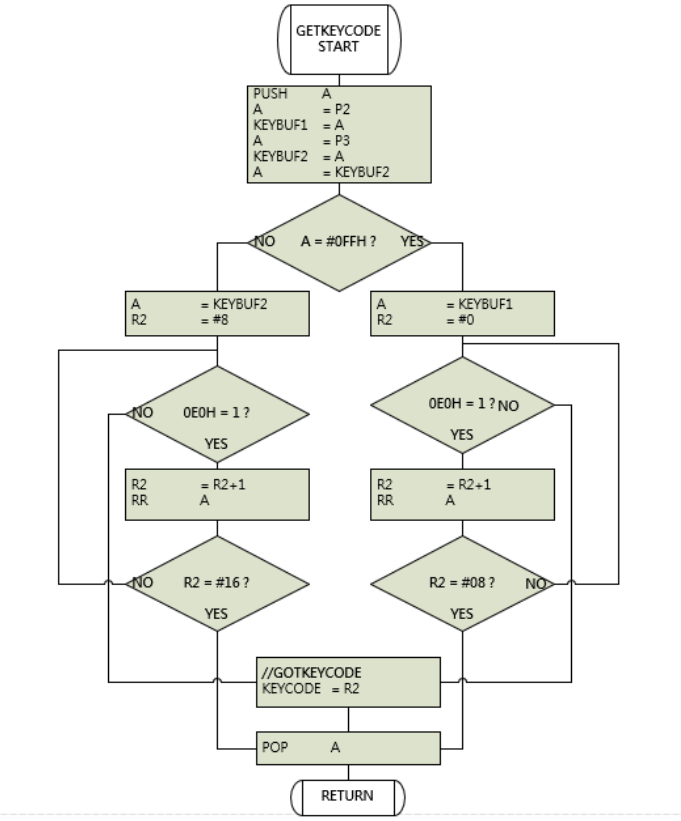

Figure 5 Gotkeycode 流程圖

接下來就是主要的按鍵功能設定，由一開始的穩定按鍵判斷後，到功能判斷，執行完功能必須重設按鍵，按鍵穩態流程圖如 Figure 6，經過一段時間之後讀 Keycode 的值如果沒有被 Push 則直接離開，如果有則判斷是否和上個時間的 Keycheck 值一樣，一樣的話表示穩態，不一樣的話給 Keycheck 現在這個時間點 Keycode 的值離開並等待下個時間再次檢查。穩態後進入按鍵功能判斷如圖 Figure 7，判斷完如果是數字鍵給定值後呼叫位移 7-sgment 程式如圖 Figure 8，位移程式 Displaykeydebounce 如果 7-segment 的顯示狀況為 blank/blank/blank/0 的話把最低 7-segment 設定為 Keycode，其餘的可能性做推移的動作。

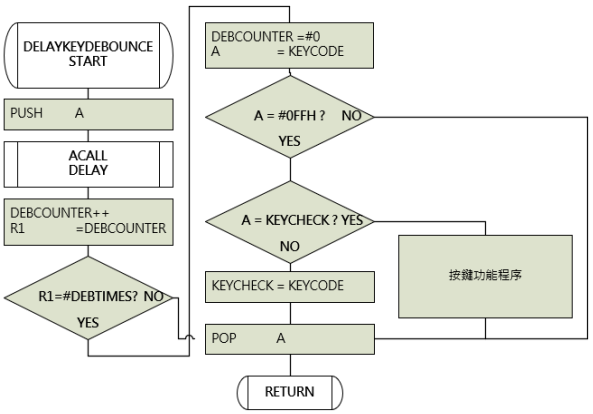

Figure 6 按鍵穩態流程圖

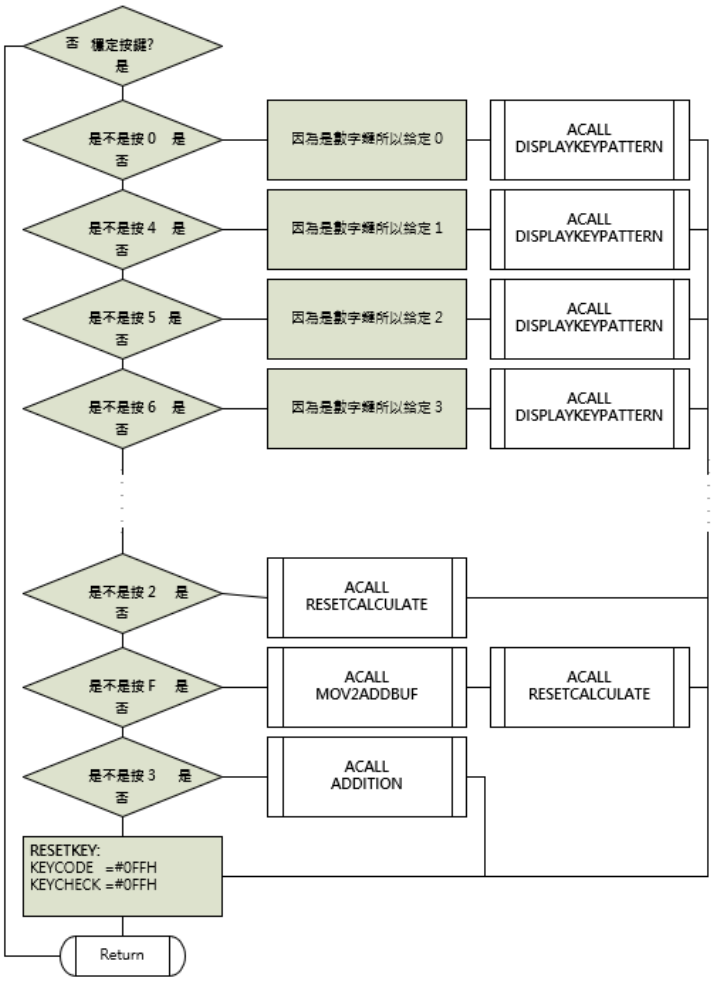

Figure 7 穩態後的按鍵判斷流程圖

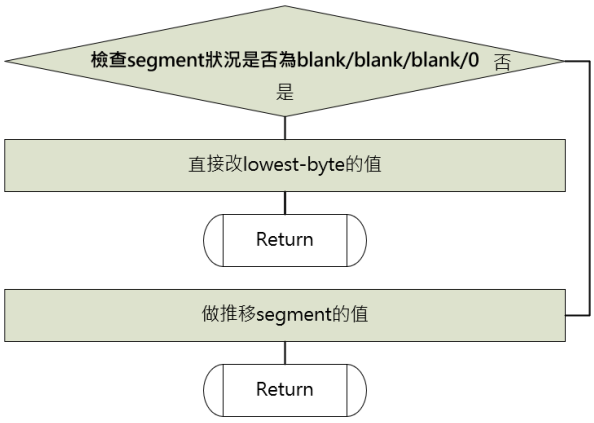

Figure 8 推移 7-segment 流程圖

而按下 Clear 時會 Resetcalculate 將值設定為 Blank/Blank/Blank/0，如果是按下加號則先將 4 個 Dispbuf 的值存入 4 個 Addbuf 做一個被分的動作，然後 Resetcalculate，而等於的如圖 Figure9 先將 4 個 Dispbuf 和另外四個 Addbuf 如果有 Blank 先轉零，之後將兩數相加，加完的數字利用再加上 246 判斷是否有溢位

如果有的話把 P1.0 設定 1，如果沒有的話減去 246 用以表示原來的值然後再把 P1.0 設成 0，最完判斷後把無用的 0 轉回 Blank，最後 P1.0 做 complement(因為 0 為亮 1 為不亮)。

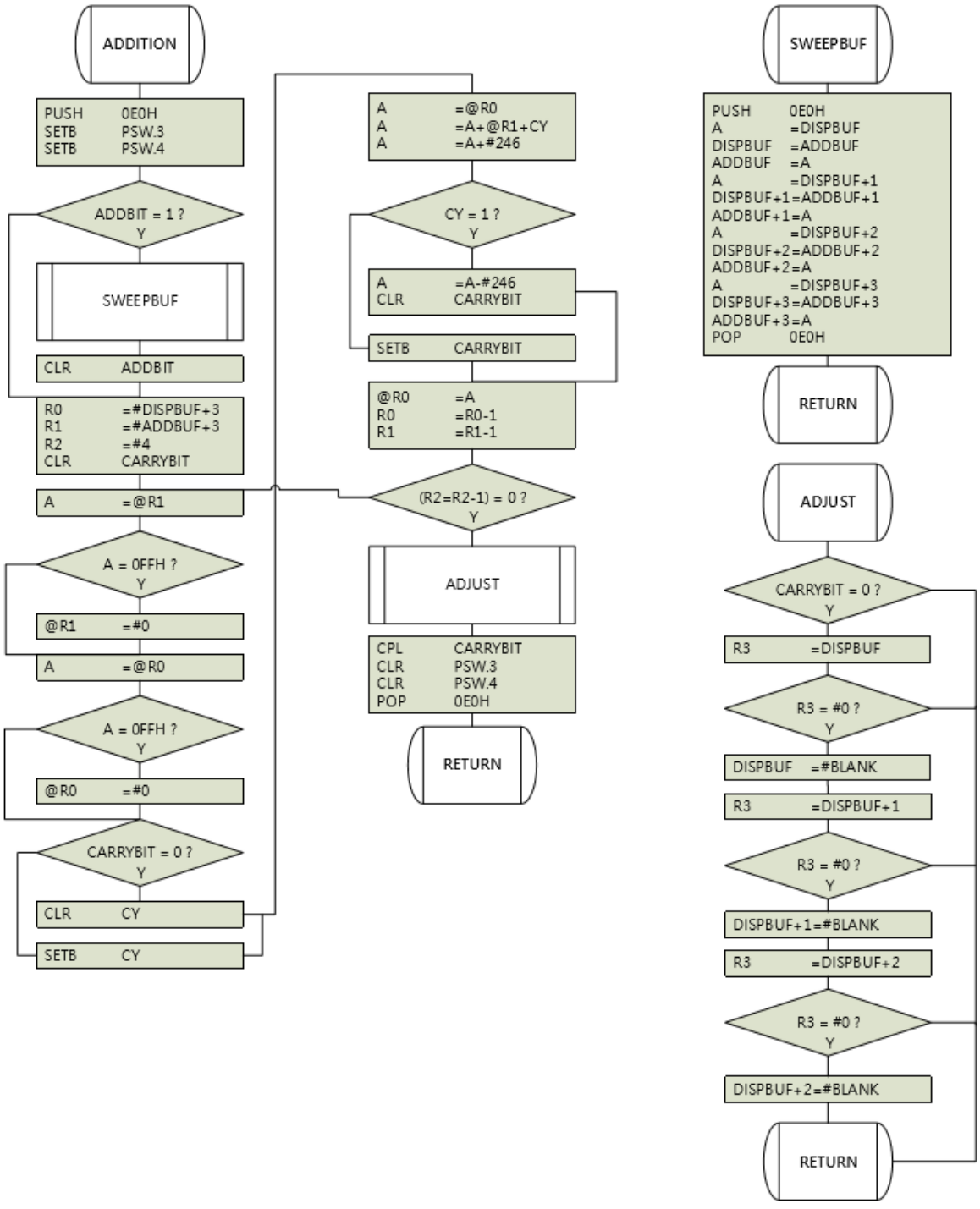

Figure 9Addition 相加副程式流程圖

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）

```
;-------------------------------------定義
//定義 byte 地址
PORT4       EQU     0E8H            // P4 原位置設置 Label
DISPBUF     EQU     40H             //用來儲存 7-segment 4 個顯示值用以查表
KEYBUF1     EQU     44H             //用來處理 0~7 按鍵 Input 的記憶體
KEYBUF2     EQU     45H             //用來處理 8~F 按鍵 Input 的記憶體
KEYCODE     EQU     46H             //用來儲存經由判斷完後哪個按鍵被 Push
KEYCHECK    EQU     47H             //用來 double-check 是否為這個按鍵被 Push
DEBCOUNTER  EQU     48H             //計數現在為第 N 個 Delay
ADDBUF      EQU     50H             //用來儲存 4 個被加數用以做加法
//定義 bit 地址
CARRYBIT    BIT     P1.0            //溢位時的旗標
ADDBIT      BIT     01H             //當加號 OP 被按時的旗標
//定義常用值
SHIFTLED    EQU     0FEH            //作為初始左旋顯示碼的作用
BLANK       EQU     0FFH            //作為給 P0 時 7-sgment Blank
DEBTIMES    EQU     60              //定義每經過 N 個 Delay double-check 的時間
;-------------------------------------主程式
            ORG     0000H           //程式開始
//初始值設定
            MOV     A,#0FFH         //給定 A 初始值#0FFH
            MOV     P2,A            //把 Port2 初始#0FFH
            MOV     P3,A            //把 Port3 初始#0FFH
            MOV     KEYBUF1,A       //把 KEYBUF1 初始#0FFH
            MOV     KEYBUF2,A       //把 KEYBUF2 初始#0FFH
            MOV     KEYCODE,A       //把 KEYCODE 初始#0FFH
            MOV     KEYCHECK,A      //把 KEYCHECK 初始#0FFH
            MOV     DEBCOUNTER,#0   //把 DEBCOUNTER 初始#0
            CLR     ADDBIT          //把 Addbit 做清除的動作
            ACALL   RESETCALCULATE  //重設 7-segment
            MOV     ADDBUF,#0       //Addbuf 設成 0
            MOV     ADDBUF+1,#0     //Addbuf+1 設成 0
            MOV     ADDBUF+2,#0     //Addbuf+2 設成 0
            MOV     ADDBUF+3,#0     //Addbuf+3 設成 0
            MOV     DPTR,#LED_TABLE //初始 DPTR 的位置 LABEL(LED_TABLE)的位置
//控制顯示 7-segment 循環顯示造成視覺暫留程式
RESET:
```

```
                MOV     R0,#DISPBUF             //重設 7-segment 顯示的值由第一個開始給
                MOV     A,#SHIFTLED             //重設 7-segment 由第一個開始亮
LOOP:
                MOV     PORT4, A                //啟動哪個 7-segment 碼給 PORT4
                ACALL   COVLED                  //7-segment 做顯示的動作
                ACALL   GETKEYCODE              //取得按鍵碼
                ACALL   DELAYKEYDEBOUNCE        //做甚麼按鍵需做甚麼事情
                INC     R0                      //指向下一個值
                RL      A                       //左移啟動 7-segment 碼
                JB      ACC.4,LOOP              //如果沒有超出第 4 個就跳 Loop
                SJMP    RESET                   //如果有超出就 Reset
;---------------------------------------
DISPLAYKEYPATTERN:
                PUSH    0E0H                    //保護 A 的值被改變
//判別是否為 blank/blank/blank/0
                MOV     R3,DISPBUF+3            //判斷第一位是否為零
                CJNE    R3,#0,GENERALCASE       //不是零就做一般推移
                MOV     R3,DISPBUF+2            //判斷第二位是否為 blank
                CJNE    R3,#BLANK,GENERALCASE   //不是 blank 就做一般推移
                MOV     R3,DISPBUF+1            //判斷第三位是否為 blank
                CJNE    R3,#BLANK,GENERALCASE   //不是 blank 就做一般推移
                MOV     R3,DISPBUF              //判斷第四位是否為 blank
                CJNE    R3,#BLANK,GENERALCASE   //不是 blank 就做一般推移
                MOV     DISPBUF+3,A             //是 blank/blank/blank/0 所以改第一個值為被按的
                SJMP    CASENDL                 //不做推移
//不是 blank/blank/blank/0 的話
GENERALCASE:
                MOV     DISPBUF,DISPBUF+1       //推移第 3 個到第 4 個
                MOV     DISPBUF+1,DISPBUF+2     //推移第 2 個到第 3 個
                MOV     DISPBUF+2,DISPBUF+3     //推移第 1 個到第 2 個
                MOV     DISPBUF+3,A             //推移被按的到第 1 個
CASENDL:
                POP     0E0H                    //POP A 因為被 PUSH 進了
                RET                             //return 原來 Call Function 下一行
;---------------------------------------
//取得按鍵碼
GETKEYCODE:
                PUSH    0E0H                    //保護 A 的值被改變
```

```
                MOV     A,P2                    //讀 Port2 的值
                MOV     KEYBUF1,A               //存取他在 Keybuf1
                MOV     A,P3                    //讀 Port3 的值
                MOV     KEYBUF2,A               //存取他在 Keybuf2
                MOV     A,KEYBUF2               //先判斷 Keybuf2(按鍵 8 到 F)
                CJNE    A,#0FFH,NEXTCODE0       //如果 8 到 F 被按下的話，跳至 Nextcode0
                MOV     A,KEYBUF1               //否則處理 Keybuf1(按鍵 0 到 7)
                MOV     R2,#0                   //重設 R2 鍵盤 counter 碼為 0
NEXTCODE1:
                JNB     0E0H,GOTKEYCODE         //如果讀到值了就去取得按鍵碼
                INC     R2                      //否則找下一個做 R2 鍵盤 counter 碼++
                RR      A                       //右旋用以找到下一個
                CJNE    R2,#8,NEXTCODE1         //如果還沒找到按鍵碼等於 8 就繼續 LOOP 找
                SJMP    EXITGETKEYCODE          //如果找到按鍵碼等於 8 就表示沒有按鍵被案
NEXTCODE0:
                MOV     A,KEYBUF2               //處理 Keybuf2(按鍵 8 到 F)
                MOV     R2,#8                   //重設 R2 鍵盤 counter 碼為 8
NEXTCODE2:
                JNB     0E0H,GOTKEYCODE         //如果讀到值了就去取得按鍵碼
                INC     R2                      //否則找下一個做 R2 鍵盤 counter 碼++
                RR      A                       //右旋用以找到下一個
                CJNE    R2,#16,NEXTCODE2        //如果還沒找到按鍵碼等於 16 就繼續 LOOP 找
                SJMP    EXITGETKEYCODE          //如果找到按鍵碼等於 16 就表示沒有按鍵被案
GOTKEYCODE:
                MOV     KEYCODE,R2              //取得按鍵碼
EXITGETKEYCODE:
                POP     0E0H                    //POP A 因為被 PUSH 進了
                RET                             //return 原來 Call Function 下一行
;---------------------------------------
DELAYKEYDEBOUNCE:
                PUSH    0E0H                    //保護 A 的值被改變
                ACALL   DELAY                   //Delay 時間
                INC     DEBCOUNTER              //Debcounter 做++計數 Delay
                MOV     R1,DEBCOUNTER           //R1 當作中介值
                CJNE    R1,#DEBTIMES,EXIT2DOOR//當 Debcounter 等於 60 時表示到了檢查時間
                MOV     DEBCOUNTER,#0           //重設 Debcounter
                MOV     A,KEYCODE               //把 Keycode 的值給 A
                CJNE    A,#0FFH,CHECK           //如果有按鍵碼就跳 Check
```

```
        SJMP    EXIT2DOOR           //EXIT 太遠的傳送門
CHECK:
        CJNE    A,KEYCHECK,COVER    //DoubleCheck Keycode 是否一樣
        SJMP    CASE0               //一樣的話開始判斷
COVER:
        MOV     KEYCHECK,KEYCODE    //不一樣的話就得重新判斷一次
EXIT2DOOR:
        SJMP    EXIT                //EXIT 太遠的傳送門
CASE0:
        CJNE    A,#00H,CASE1        //如果不是 0 就檢查 1
        MOV     A,#00H              //給 0
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
CASE1:
        CJNE    A,#04H,CASE2        //如果不是 1 就檢查 2
        MOV     A,#01H              //給 1
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
CASE2:
        CJNE    A,#05H,CASE3        //如果不是 2 就檢查 3
        MOV     A,#02H              //給 2
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
CASE3:
        CJNE    A,#06H,CASE4        //如果不是 3 就檢查 4
        MOV     A,#03H              //給 3
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
CASE4:
        CJNE    A,#08H,CASE5        //如果不是 4 就檢查 5
        MOV     A,#04H              //給 4
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
CASE5:
        CJNE    A,#09H,CASE6        //如果不是 5 就檢查 6
        MOV     A,#05H              //給 5
        ACALL   DISPLAYKEYPATTERN   //做推移顯示
        SJMP    RESETKEY            //重設按鍵
```

```
CASE6:
        CJNE    A,#0AH,CASE7            //如果不是 6 就檢查 7
        MOV     A,#06H                  //給 6
        ACALL   DISPLAYKEYPATTERN       //做推移顯示
        SJMP    RESETKEY                //重設按鍵
CASE7:
        CJNE    A,#0CH,CASE8            //如果不是 7 就檢查 8
        MOV     A,#07H                  //給 7
        ACALL   DISPLAYKEYPATTERN       //做推移顯示
        SJMP    RESETKEY                //重設按鍵
CASE8:
        CJNE    A,#0DH,CASE9            //如果不是 8 就檢查 9
        MOV     A,#08H                  //給 8
        ACALL   DISPLAYKEYPATTERN       //做推移顯示
        SJMP    RESETKEY                //重設按鍵
CASE9:
        CJNE    A,#0EH,CASECLR          //如果不是 9 就檢查 clear
        MOV     A,#09H                  //給 9
        ACALL   DISPLAYKEYPATTERN       //做推移顯示
        SJMP    RESETKEY                //重設按鍵
CASECLR:
        CJNE    A,#02H,CASEADD          //如果不是 clear 就檢查 add
        ACALL   RESETCALCULATE          //叫副程式做顯示器
        SJMP    RESETKEY                //重設按鍵
CASEADD:
        CJNE    A,#0FH,CASEEQUAL        //如果不是 add 就檢查 equal
        ACALL   MOV2ADDBUF              //叫副程式做移動這些可能要被加的數字
        ACALL   RESETCALCULATE          //重設顯示器
        SJMP    RESETKEY                //重設按鍵
CASEEQUAL:
        CJNE    A,#03H,RESETKEY         //如果不是 equal 就重設鍵盤
        ACALL   ADDITION                //叫副程式做相加動作
        SJMP    RESETKEY                //重設按鍵
RESETKEY:
        MOV     KEYCODE,#0FFH           //重設 keycode
        MOV     KEYCHECK,#0FFH          //重設 keycheck
EXIT:
        POP     0E0H                    //POP A 因為被 PUSH 進了
```

```
                RET                             //return 原來 Call Function 下一行
;---------------------------------------
RESETCALCULATE:
                PUSH    0E0H                    //保護 A 的值被改變
                MOV     DISPBUF,#BLANK          //設第四個 7-segment 為 Blank
                MOV     DISPBUF+1,#BLANK        //設第三個 7-segment 為 Blank
                MOV     DISPBUF+2,#BLANK        //設第二個 7-segment 為 Blank
                MOV     DISPBUF+3,#0            //設第一個 7-segment 為 0
                SETB    CARRYBIT                //重設 overflow
                POP     0E0H                    //POP A 因為被 PUSH 進了
                RET                             //return 原來 Call Function 下一行
;---------------------------------------
MOV2ADDBUF:
                PUSH    0E0H                    //保護 A 的值被改變
                SETB    ADDBIT                  //舉起 Add 旗標
                MOV     ADDBUF,DISPBUF          //把第四個顯示值存給第四個加法 buffer
                MOV     ADDBUF+1,DISPBUF+1      //把第三個顯示值存給第三個加法 buffer
                MOV     ADDBUF+2,DISPBUF+2      //把第二個顯示值存給第二個加法 buffer
                MOV     ADDBUF+3,DISPBUF+3      //把第一個顯示值存給第一個加法 buffer
                POP     0E0H                    //POP A 因為被 PUSH 進了
                RET                             //return 原來 Call Function 下一行
;---------------------------------------
ADDITION:
                PUSH    0E0H                    //保護 A 的值被改變
                SETB    PSW.3                   //使用 Bank2 設 PSW.3 為 1
                SETB    PSW.4                   //使用 Bank2 設 PSW.4 為 1
                JNB     ADDBIT,NOTADDBUF        //如果 addbit 舉起
                ACALL   SWEEPBUF                //則交換加數與被加數至正確位置
                CLR     ADDBIT                  //交換好後即可以清除
NOTADDBUF:
                MOV     R0,#DISPBUF+3           //把 R0 當作指標指向 Dispbuf+3
                MOV     R1,#ADDBUF+3            //把 R1 當作指標指向 Addbuf+3
                MOV     R2,#4                   //把 R2 當作計數 4 次
                CLR     CARRYBIT                //清除 Carrybit 用以準備相加
ADDAGAIN:
                MOV     A,@R1                   //檢查 Addbuf4 個是否有 Blank
                CJNE    A,#BLANK,CHECKANOTHER//是否為 Blank
                MOV     @R1,#0                  //如果有轉零
```

```
CHECKANOTHER:
        MOV     A,@R0               //檢查 Dispbuf4 個是否有 Blank
        CJNE    A,#BLANK,NOBLANK    //是否為 Blank
        MOV     @R0,#0              //如果有轉零
NOBLANK:
        JB      CARRYBIT,SET1       //如果有 Carry 跳 SET CY
        CLR     PSW.7               //如果沒有 CY 等於 0
        SJMP    DOADD               //做完做加法
SET1:
        SETB    PSW.7               //CY 就等於 1
//轉換 blank to 0 完後做相加
DOADD:
        MOV     A,@R0               //把 dispbuf 記憶體位置給 A
        ADDC    A,@R1               //dispbuf 和 addbuf 做相加連同 CY 一起
        ADD     A,#246              //判斷相加後是否大於等於十
        JNC     NOTCARRY            //若沒有大於十則跳 Notcarry
        SETB    CARRYBIT            //設 Carrybit 是 1
        SJMP    CARRYENDL           //超過溢位循環不用處理因此跳給值的地方
NOTCARRY:
        SUBB    A,#246              //因為沒有 carry 所以減去 246
        CLR     CARRYBIT            //然後把 Carrybit 清除
CARRYENDL:
        MOV     @R0,A               //給 dispbuf 處理後相加的值
        DEC     R0                  //處理下一個被加數
        DEC     R1                  //處理下一個加數
        DJNZ    R2,ADDAGAIN         //總共處理 4 次
        ACALL   ADJUST              //調整加完之後 0 的處理
        CPL     CARRYBIT            //清除 Carrybit
        CLR     PSW.3               //使用 Bank0 設 PSW.4 為 0
        CLR     PSW.4               //使用 Bank0 設 PSW.4 為 0
        POP     0E0H                //POP A 因為被 PUSH 進了
        RET                         //return 原來 Call Function 下一行
;-------------------------------------
SWEEPBUF:
        PUSH    0E0H                //保護 A 的值被改變
        MOV     A,DISPBUF           //sweep 第 4 個加數與被加數
        MOV     DISPBUF,ADDBUF      //用 A 當中介
        MOV     ADDBUF,A            //第 4 個 sweep 完畢
```

```
        MOV     A,DISPBUF+1         //sweep 第 3 個加數與被加數
        MOV     DISPBUF+1,ADDBUF+1  //用 A 當中介
        MOV     ADDBUF+1,A          //第 3 個 sweep 完畢
        MOV     A,DISPBUF+2         //sweep 第 2 個加數與被加數
        MOV     DISPBUF+2,ADDBUF+2  //用 A 當中介
        MOV     ADDBUF+2,A          //第 2 個 sweep 完畢
        MOV     A,DISPBUF+3         //sweep 第 1 個加數與被加數
        MOV     DISPBUF+3,ADDBUF+3  //用 A 當中介
        MOV     ADDBUF+3,A          //第 1 個 sweep 完畢
        POP     0E0H                //POP A 因為被 PUSH 進了
        RET                         //return 原來 Call Function 下一行
;--------------------------------------
//由左開始判斷如果有零就轉回來
ADJUST:
        PUSH    0E0H                //保護 A 的值被改變
        JB      CARRYBIT,NOTZERO    //overflow 時不用轉
        MOV     R3,DISPBUF          //用 R3 當中介
        CJNE    R3,#0,NOTZERO       //如果第三個為零
        MOV     DISPBUF,#BLANK      //就轉 Blank
        MOV     R3,DISPBUF+1        //用 R3 當中介
        CJNE    R3,#0,NOTZERO       //如果第二個為零
        MOV     DISPBUF+1,#BLANK    //就轉 Blank
        MOV     R3,DISPBUF+2        //用 R3 當中介
        CJNE    R3,#0,NOTZERO       //如果第一個為零
        MOV     DISPBUF+2,#BLANK    //就轉 Blank
        MOV     R3,DISPBUF+3        //用 R3 當中介
NOTZERO:
        POP     0E0H                //POP A 因為被 PUSH 進了
        RET                         //return 原來 Call Function 下一行
;--------------------------------------
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           //R7--當 R7 不等於零跳 Delay3
        DJNZ    R6,DELAY2           //R6--當 R6 不等於零跳 Delay2
        DJNZ    R5,DELAY1           //R5--當 R7 不等於零跳 Delay1
        DJNZ    R4,DELAY0           //R4--當 R7 不等於零跳 Delay0
        RET                         //return 原來 Call Function 下一行
;--------------------------------------
//點亮 7-segment 數字
COVLED:
        PUSH    0E0H                //保護 A 的值被改變
        MOV     A,@R0               //把指標 R0 的值取出
        CJNE    A,#BLANK,NOTBLANK   //如果不是 Blank 就跳查表
        SJMP    BLANK2P0            //如果是 Blank 就直接給值
NOTBLANK:
        MOVC    A,@A+DPTR           //查表後把值給 A
BLANK2P0:
        MOV     P0,A                //顯示給值的數字
        POP     0E0H                //POP A 因為被 PUSH 進了
        RET                         //return 原來 Call Function 下一行
;--------------------------------------
LED_TABLE:
        DB      0C0H                //7-segment ‘0’
        DB      0F9H                //7-segment ‘1’
        DB      0A4H                //7-segment ‘2’
        DB      0B0H                //7-segment ‘3’
        DB      99H                 //7-segment ‘4’
        DB      92H                 //7-segment ‘5’
        DB      82H                 //7-segment ‘6’
        DB      0D8H                //7-segment ‘7’
        DB      80H                 //7-segment ‘8’
        DB      90H                 //7-segment ‘9’
        DB      88H                 //7-segment ‘A’
        DB      83H                 //7-segment ‘b’
        DB      0C6H                //7-segment ‘C’
        DB      0A1H                //7-segment ‘d’
        DB      86H                 //7-segment ‘E’
        DB      8EH                 //7-segment ‘F’
;--------------------------------------
```

```
END                                    //程式結束
```

四、 實作過程、心得、與結論

7-segment 初始狀態如圖 Figure A，按下第一個數字 123 如圖 Figure B，按下加號時計算機會再次清除 7-segment 上的顯示如圖 Figure C 並把上一個值存取到 Addbuf，接著按下第二個數字 456 如圖 Figure D，接著按下等於如圖 Figure E，因為做第一次加法時會會轉換加數與被加數的位置，因此 Addbuf 是存取加數再次按下時會再加上 456 如圖 Figure F，最後觀察溢位情形按下 Clear 後測試 1235 加上 9999，結果出現 P1.0 的燈號是亮的且顯示 1234(表示值為 11234)如圖 Figure G。

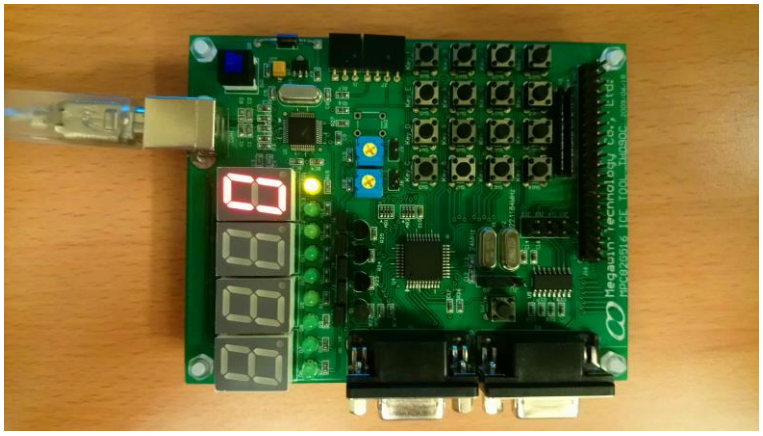

Figure A 實作初始狀態

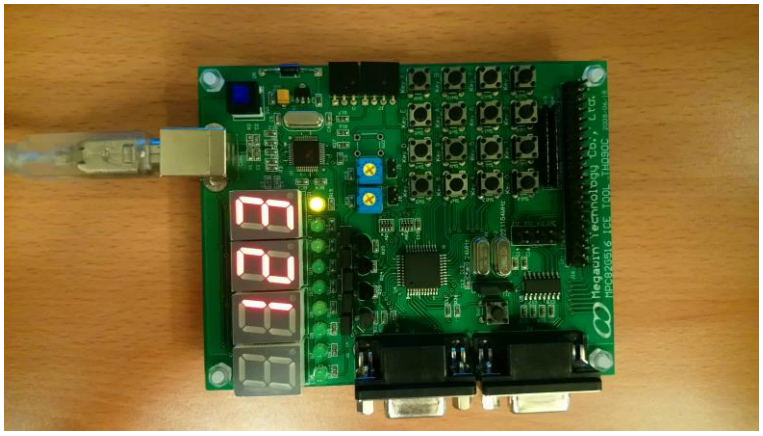

Figure B 按下第一個值 123

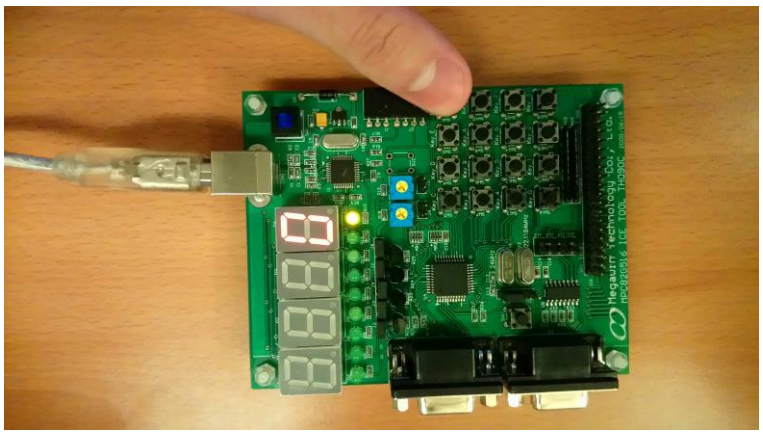

Figure C 按下加號時的情況

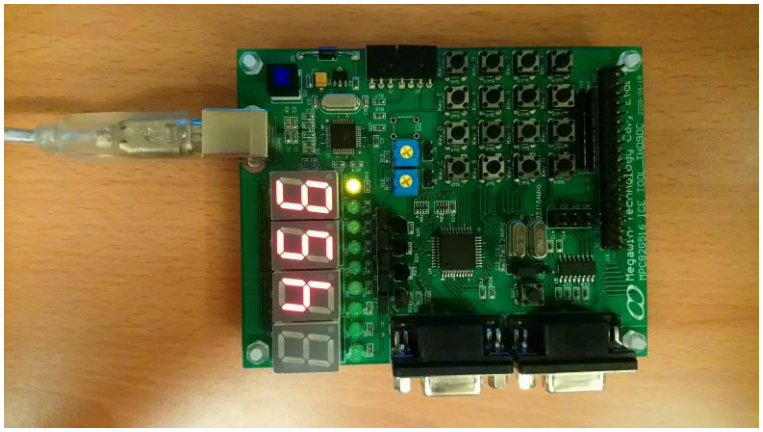

Figure D 按下第二個數字

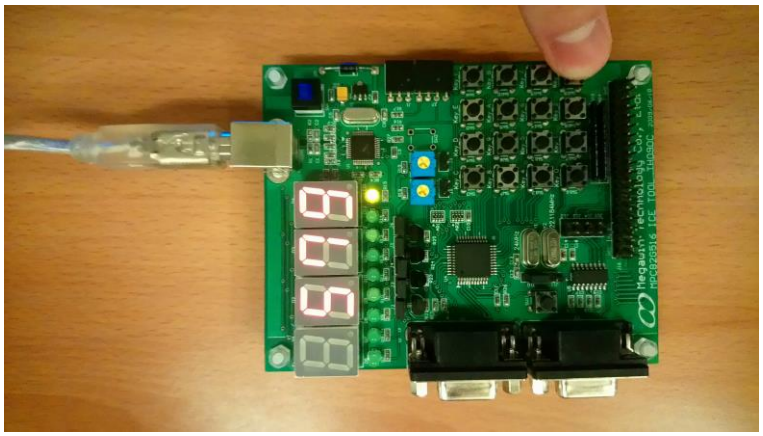

Figure E 按下等於後的結果

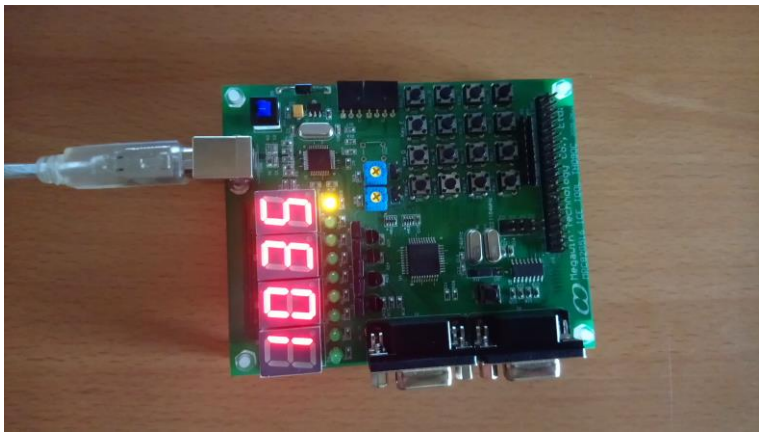

Figure F 再次按下等於後的結果(連加觀察)

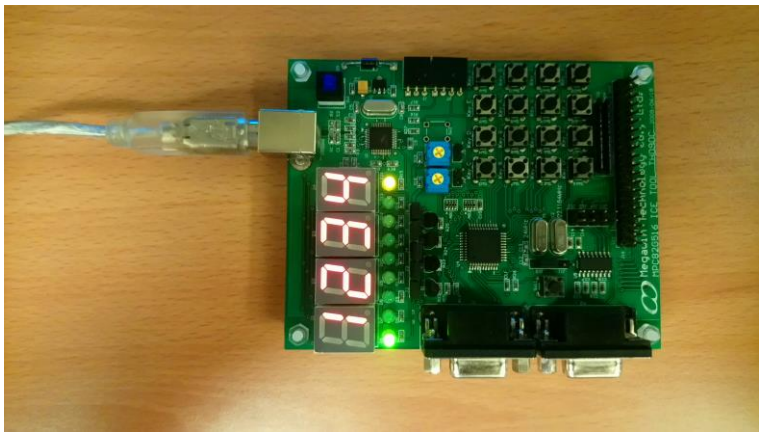

Figure G 溢位時的結果

經過這次的實作可以了解到計算機的運作方式，或許這只是加法，也了解到計算機底下的程式並非容易的，或許可以延伸加減乘除四種不同計算機的片段，以結合成一台基本的計算機，也明白到 8051 功能的強大，連結一些顯示功能結合時間像是電子時鐘或者計時器又或者結合 8*8 點矩陣利用出更多樣的變化。

五、 參考文獻

[1]Muhammad Ali Mazidi/Janice G.Mazidi/Rolin D.McKinlay，The 8051 Microcontroller and Embedded System，Internal Second Edition，2006。