Due: 4/11/2024 3am

National Taiwan University of Science and Technology, NTUST

Design of Embedded Microprocessor Systems

EE5019701_HW2

教 授 : 王 乃 堅

班級:電機所碩一

姓名:陳俊博

學號:M11207521

目錄

一、 目的&原理

使用泡沫排序法來處理一組數字：-10, 11, 20, 50, -20, -3。首先對這些有號數進行排序，使它們按從小到大的順序排列。這一過程不僅需確保數字間的正確順序，特別是要注意負數的處理，確保它們在正確的位置。排序完成後，將這些數字進行累加，同時檢查是否存在溢出的情況，這涉及到了進位的檢測，以確保計算的準確性。將最終的排序結果儲存於 ArrayB 中，而溢出情況和總和則分別存放在 R5 和 R7 暫存器中。接著針對相同的數字序列再次進行排序，這次將它們視作無號數來處理。在無號數的排序過程中，需注意不將它們當作負數來處理，因為無號數的值範圍與有號數不同，要求在排序時必須正確理解數值的範圍以避免錯誤。排序後的無符號數同樣進行累加，並且檢查是否溢出。結果將存放於 ArrayC，而溢出情況和總和分別放在 R6 和 R8 暫存器中。

泡沫排序是一種簡單的排序演算法，通過重複遍歷需排序的數列，比較每對相鄰元素，並在順序錯誤的情況下交換它們，逐步將每個元素放到它應該在的位置。這個過程一直重複到沒有再需要交換的元素為止，即該數列已排序完成。

這個過程於數列的第一個元素，持續進行直到整個數列排序完成。在每次遍歷中，演算法從數列的起始位置開始，一直到最後一個尚未排序的元素。它比較每一對連續元素，即第 i 個和第 i+1 個元素。如果發現前者大於後者，則兩者會交換位置。這樣的比較和交換過程確保了每輪遍歷都能將該輪中最大的元素推移到其應有的位置，為數列的未排序部分的最後。隨著每輪的完成，未排序的數列長度逐漸減少，因為每完成一輪遍歷後，至少有一個元素被放置在其最終位置上。在這個過程一直持續重複，直到不再需要進行任何交換，表示數列已排序完成。

Time complexity:

1. Best case: $O(n)$
   情況為 input data 恰巧由小→大呈現
   (只需 n-1 次比較，且無 swap 發生)
2. Worst case: $O(n^2)$
   情況為 input data 恰巧由大→小呈現
   (若 n 筆，則需 swap n-1, n-2,..., 1 次→n(n-1)/2 次)
3. Average case: $O(n^2)$。

Space complexity:

- $O(n^2)$

二、 程式流程圖

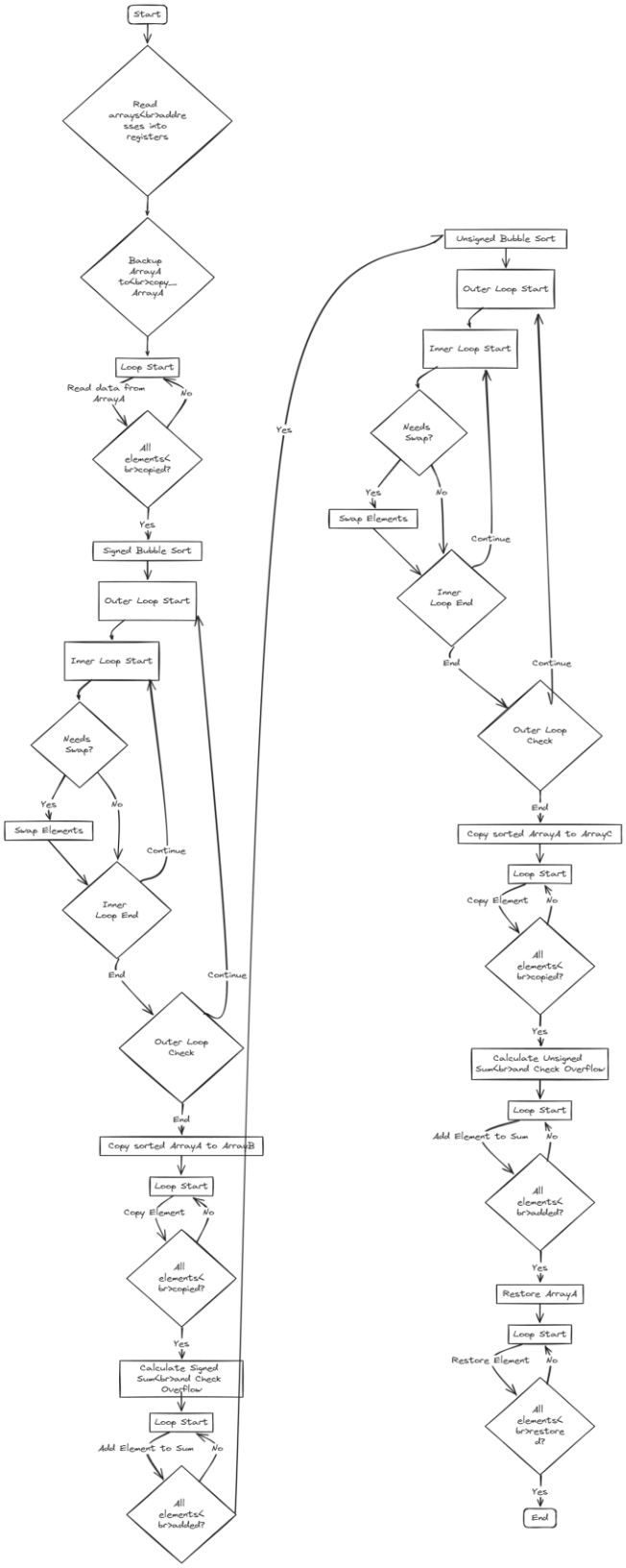

圖 1 程式流程圖

三、 程式碼(含註解)

```
    AREA Matrix, CODE, READONLY ; AREA 定義一個記憶體區域，包含指令或資料。
                                ; Matrix 為該記憶體區域的名稱。
                                ; CODE 為該區域用來存放程式碼。
                                ; READONLY 為程式在執行時這部分記憶體不能被修改。
    ENTRY                       ; ENTRY 表示程式開始執行的地方。

start                           ; 程式開始
    LDR r0, =ArrayA             ; 將 ArrayA 的地址載入至 r0
    LDR r10, =ArrayB            ; 將 ArrayA 的地址載入至 r10
    LDR r11, =ArrayC            ; 將 ArrayA 的地址載入至 r11
    LDR r12, =copy_ArrayA       ; 將 copy_ArrayA 的地址載入至 r12
    MOV r1, #5                  ; 將常數 5 移至 r1 (用於迴圈 count)
    MOV r2, #0                  ; 將常數 0 移至 r2 (用於迴圈 index)

backup_ArrayA                   ; 備份 ArrayA
    LDR r3, [r0, r2, LSL #2]    ; 將 r0 + r2 * 4 位置的值載入至 r3
    STR r3, [r12, r2, LSL #2]   ; 將 r3 中的值儲存至 r12 + r2 * 4 位置
    ADD r2, r2, #1              ; r2 加 1
    CMP r2, #6                  ; 將 r2 與 6 比較
    BLT backup_ArrayA           ; 如果 r2 小於 6，則跳回至 backup_ArrayA
                                ; 重複到 ArrayA 中的值備份至 copy_ArrayA 就結束

signed_out_loop                 ; 有號數 bubble sort 的外部迴圈
    MOV r2, #0                  ; 內部迴圈設定 r2 為 0

signed_in_loop                  ; 有號數 bubble sort 的內部迴圈
    LDR r3, [r0, r2, LSL #2]    ; 將 r0 + r2 * 4 位置的值載入至 r3
    ADD r9, r2, #1              ; 計算下一個 index(r2 + 1)並儲存至 r9
    LDR r4, [r0, r9, LSL #2]    ; 將 r0 + r9 * 4 位置的值載入至 r4
    CMP r3, r4                  ; 將 r3 與 r4 的值進行比較
    BLT signed_no_swap          ; 如果 r3 小於 r4，則不需要交換; 否則進行交換
    STR r4, [r0, r2, LSL #2]    ; 將 r4 中的值儲存至 r0 + r2 * 4 位置
    STR r3, [r0, r9, LSL #2]    ; 將 r3 中的值儲存至 r0 + r9 * 4 位置

signed_no_swap                  ; 有號數不須交換的情況
    ADD r2, r2, #1              ; index r2 = r2 + 1
    CMP r2, r1                  ; 將 index r2 和 index r1 比較
    BLT signed_in_loop          ; 如果 r2 小於 r1，返回至內部迴圈繼續比較
    SUBS r1, r1, #1             ; 如果 r2 沒有小於 r1，index = r1 - 1
    CMP r1, #0                  ; 將 index r1 與 0 進行比較
    BGT signed_out_loop         ; 如果 r1 大於 0，返回至外部迴圈; 否則代表完成排序
    MOV r1, #0                  ; 重設為 r1 為 0，下一個 function 需用到

signed_copy_ArrayA_to_ArrayB    ; 將排序好的有號數複製到 ArrayB
    LDR r3, [r0, r1, LSL #2]    ; 將 r0 + r1 * 4 位置的值載入至 r3
    STR r3, [r10, r1, LSL #2]   ; 將 r3 中的值儲存至 r10 + r1 * 4 位置
    ADD r1, r1, #1              ; r1 加 1
    CMP r1, #6                  ; 將 r1 與 6 比較
    BLT signed_copy_ArrayA_to_ArrayB    ; 如果 r1 小於 6，繼續複製值到 ArrayB

fir_initial                     ; 第一次初始化使用過的暫存器
    MOV r1, #0
    MOV r2, #0
```

```
    MOV r3, #0
    MOV r4, #0
    MOV r5, #0
    MOV r7, #0
    MOV r9, #0

signed_sum_overflow             ; 計算有號數的和與溢出次數
    LDR r4, [r0, r2, LSL #2]    ; 將 r0 + r2 * 4 位置的值載入至 r4
    ADDS r7, r7, r4             ; 將有號數總和加到 r7 中，並更新溢出
    ADC r5, r5, #0              ; 將前一加法的進位加到 r5 中
    ADD r2, r2, #1              ; index r2 = r2 + 1
    CMP r2, #6                  ; 將 r2 與 6 進行比較
    BLT signed_sum_overflow     ; 如果 r2 小於 6，則繼續加總
    MOV r1, #5                  ; 為無號數排序重設 r1 為 5

unsigned_out_loop               ; 無號數 bubble sort 的外部迴圈
    MOV r2, #0                  ; 為內部迴圈重設 r2 為 0

unsigned_in_loop                ; 無號數 bubble sort 的內部迴圈
    LDR r3, [r0, r2, LSL #2]    ; 將 r0 + r2 * 4 位置的值載入至 r3
    ADD r9, r2, #1              ; 計算下一個 index(r2 + 1)並儲存至 r9
    LDR r4, [r0, r9, LSL #2]    ; 將 r0 + r9 * 4 位置的值載入至 r4
    CMP r3, r4                  ; r3 與 r4 進行比較
    BLO unsigned_no_swap        ; 如果 r3 小於 r4，則不須交換，否則進行交換
    STR r4, [r0, r2, LSL #2]    ; 將 r4 中的值儲存至 r0 + r2 * 4 位置
    STR r3, [r0, r9, LSL #2]    ; 將 r3 中的值儲存至 r0 + r9 * 4 位置

unsigned_no_swap                ; 有號數不須交換的情況
    ADD r2, r2, #1              ; index r2 = r2 + 1
    CMP r2, r1                  ; index r2 與 index r1 進行比較
    BLT unsigned_in_loop        ; 如果 r2 小於 r1，返回內部迴圈
    SUBS r1, r1, #1             ; 如果 r2 沒有小於 r1，index r1 = r1 - 1
    CMP r1, #0                  ; r1 與 0 進行比較
    BGT unsigned_out_loop       ; 如果 r1 大於 0，返回外部迴圈; 否則代表完成排序
    MOV r1, #0                  ; 重設為 r1 為 0，下一個 function 需用到

unsigned_copy_ArrayA_to_ArrayC  ; 將排序好的無號數複製到 ArrayC
    LDR r3, [r0, r1, LSL #2]    ; 將 r0 + r1 * 4 位置的值載入至 r3
    STR r3, [r11, r1, LSL #2]   ; 將 r3 中的值儲存至 r11 + r1 * 4 位置
    ADD r1, r1, #1              ; r1 加 1
    CMP r1, #6                  ; 將 r1 與 6 進行比較
    BLT unsigned_copy_ArrayA_to_ArrayC  ;如果 r1 小於 6，繼續複製到 ArrayC

sec_initial                     ; 第二次初始化使用過的暫存器
    MOV r1, #0
    MOV r2, #0
    MOV r3, #0
    MOV r4, #0
    MOV r6, #0
    MOV r8, #0
    MOV r9, #0

unsigned_sum_overflow           ; 計算無號數的和與溢出次數
    LDR r4, [r0, r2, LSL #2]    ; 將 r0 + r2 * 4 位置的值載入至 r4
    ADDS r8, r8, r4             ; 將無號數總和加到 r8 中，並更新溢出
    ADC r6, r6, #0              ; 將前一加法的進位加到 r6 中
```

```
    ADD r2, r2, #1              ; index r2 = r2 + 1
    CMP r2, #6                  ; 將 r2 與 6 進行比較
    BLT unsigned_sum_overflow   ; 如果 r2 小於 6，則繼續加總
    MOV r1, #0                  ; 為 r1 設定為 0
    LDR r2, =copy_ArrayA        ; 將 copy_ArrayA 的地址載入至 r2

restore                         ; 還原 ArrayA 的數值
    LDR r3, [r2, r1, LSL #2]    ; 將 r2 + r1 * 4 位置的值載入至 r3
    STR r3, [r12, r1, LSL #2]   ; 將 r3 中的值儲存至 r12 + r1 * 4 位置
    ADD r1, r1, #1              ; r1 加 1
    CMP r1, #6                  ; 將 r1 與 6 進行比較
    BLT restore                 ; 如果 r1 小於 6，繼續還原 ArrayA 的值;否則還原完成
    MOV r2, #0                  ; 初始化值
    MOV r3, #0                  ; 初始化值
    MOV r4, #0                  ; 初始化值

stop                            ; 程式停止
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x123456                ; 中斷
    AREA Data, DATA, READWRITE  ; 定義名為 Data 的資料區域，可讀寫
ArrayA dcd -10, 11, 20, 50, -20, -3 ; 初始化 ArrayA
ArrayB dcd 0, 0, 0, 0, 0, 0         ; 初始化 ArrayB
ArrayC dcd 0, 0, 0, 0, 0, 0         ; 初始化 ArrayC
copy_ArrayA dcd 0, 0, 0, 0, 0, 0    ; 初始化 copy_ArrayA
    END                             ; 結束
```

四、 程式執行結果

在這次的實驗中，有號數的總和被存放在 r7 暫存器，而溢位的結果則被放置在 r5 暫存器。從圖 3 的狀態觀察，r5 暫存器的顯示值為 3，這表示在加法過程中發生了三次溢位。r7 暫存器的值為 48，這代表經過有號數排序後的累加結果。具體到運行過程，當在 Debug 模式下按下 f8（單步執行）時，可以看到 r5 暫存器的值從 0 變化到 1，這發生在將-20 與-10 相加後得到-30 的過程中，發生了第一次溢位，如圖 4 及圖 5 所示。接著在圖 6 和圖 7 中，也觀察到了類似的溢位情況，證明了總共發生了三次溢位。

至於有號數排序的結果則被放置在 ArrayB 中。在我的程式碼中，ArrayB 的地址被載入到 r10 暫存器，從圖中可以看到 r10 暫存器的第一個位置是-20，隨後每隔四個位置分別是-10、-3、11、20 和 50，如圖 8 所示。

對於無符號數的處理，排序結果被放置在 ArrayC 中，且 ArrayC 的地址載入到 r11 暫存器。從圖中可以看到 r11 暫存器的第一個位置是 11，隨後每隔四個位置依次是 20、50、-20、-10 和-3，如圖 9 所示。

無符號數的總和與溢位次數結果顯示在 r6 暫存器（溢位次數）與 r8 暫存器（無號數總和），均為 3 次溢位和總和為 48。

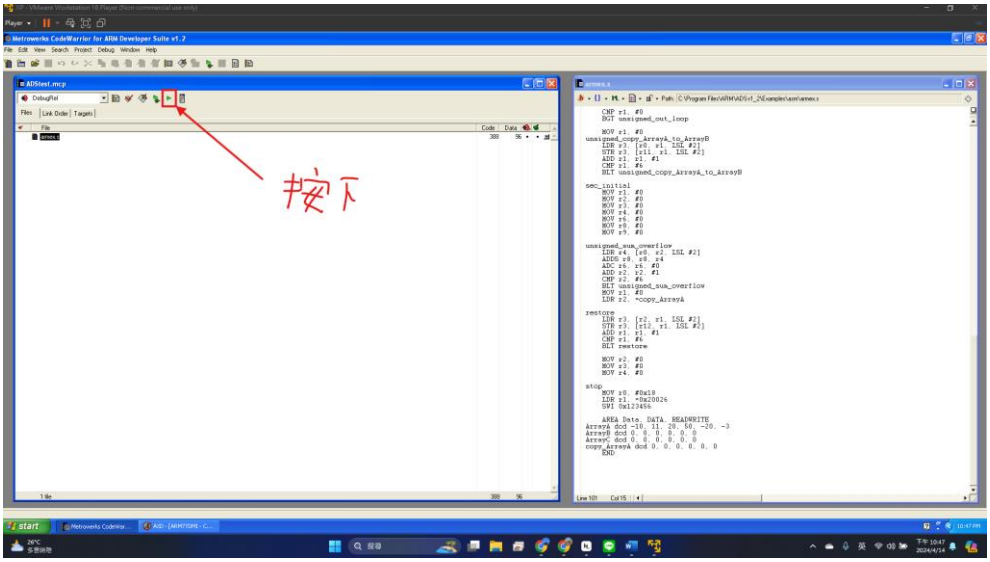

圖 2　　ARM Developer Suite v1.2 介面

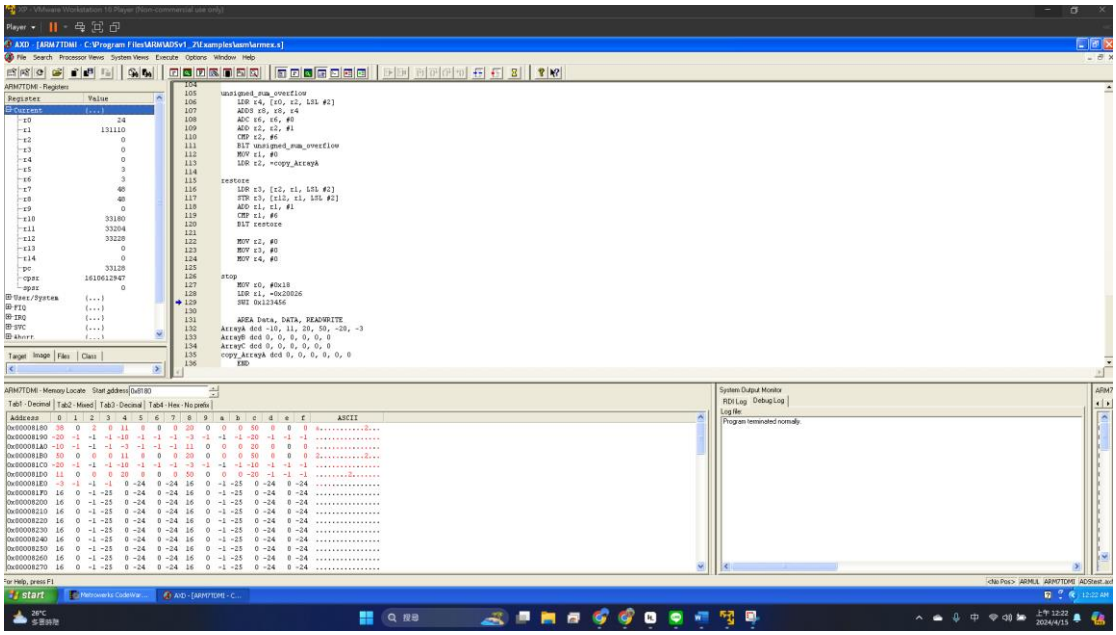

圖 3　　按下 Run 後執行結果之 AXD 介面

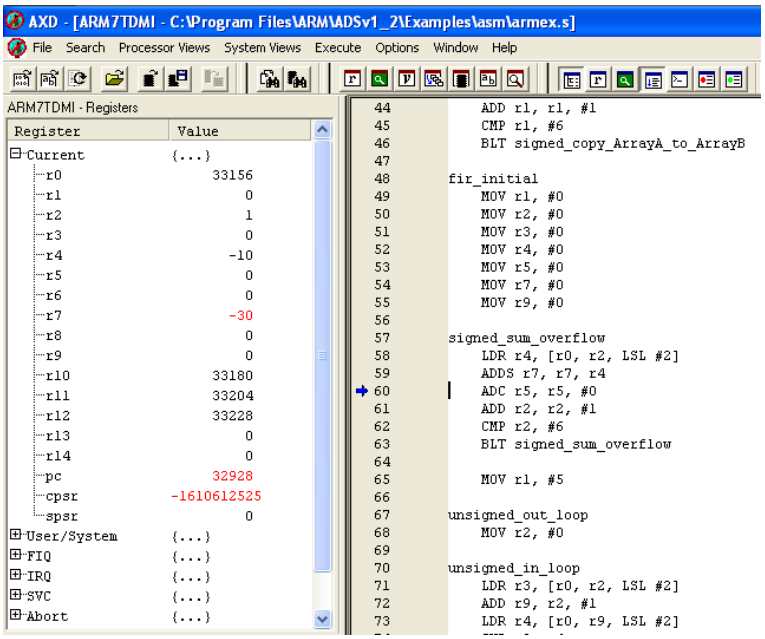

圖 4　無發生溢位

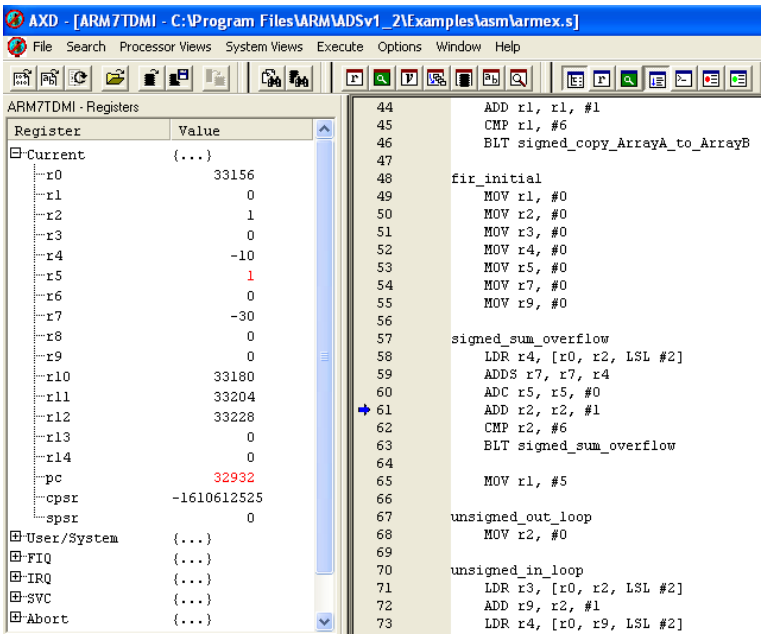

圖 5　發生第一次溢位

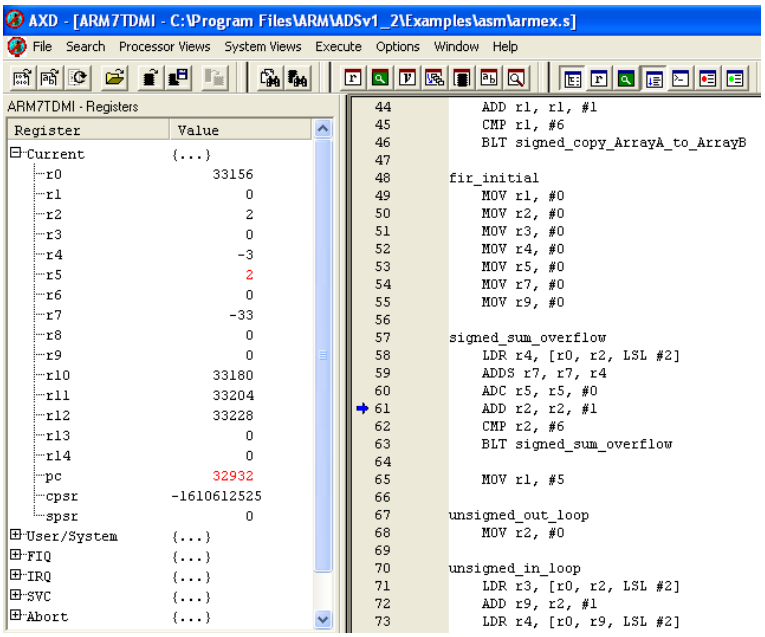

圖 6　　發生第二次溢位

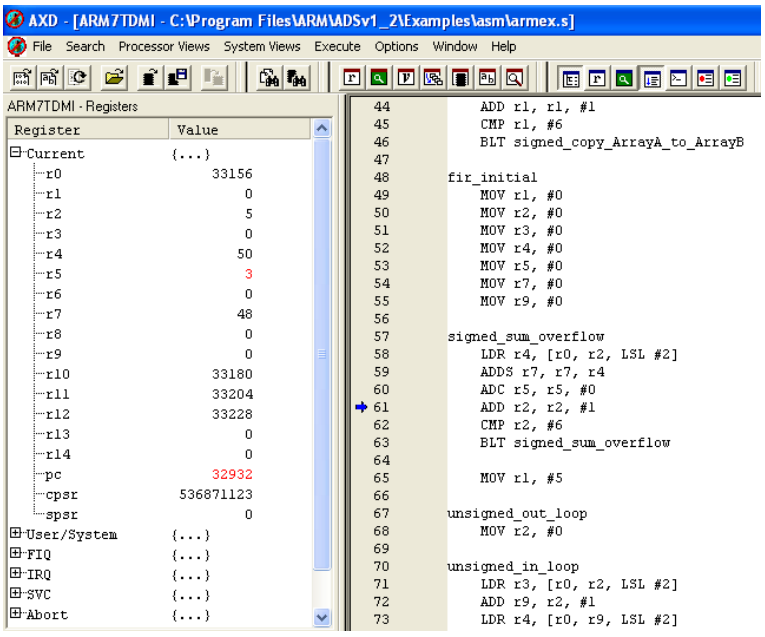

圖 7　　發生第三次溢位

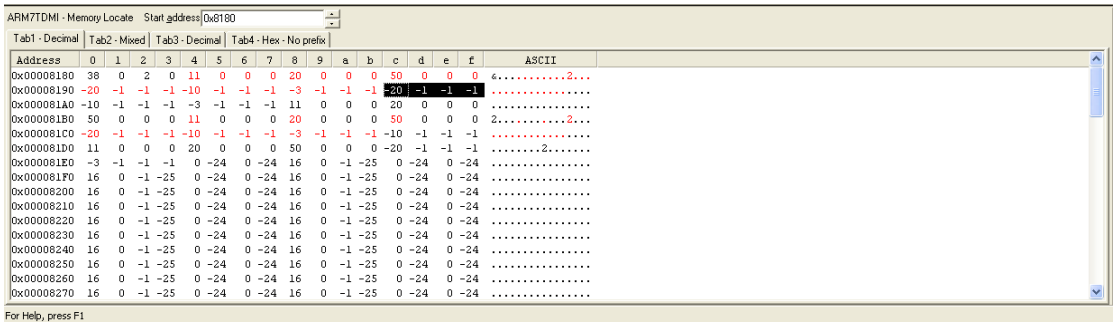

圖 8　　有號數 ArrayB 排序之結果

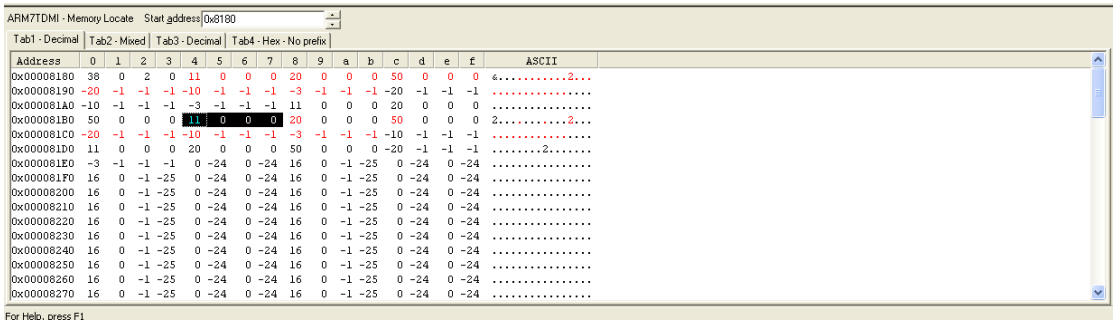

圖 9　　無號數 ArrayC 排序之結果

五、　心得

在這次的實驗中，我使用了泡沫排序法來處理有號數和無號數，並進行了不同情況下的數值累加和溢出檢查。透過這個實驗，我對嵌入式系統中資料處理和記憶體管理有了初步的認識。

在這次實驗中，針對一些數值進行了累加和溢出的檢測。使用 ARM 組合語言來進行這些操作，學習如何用 ARM 組合語言進行數學運算和溢出檢查，對我來說是非常有用的技能，這不僅提高對 ARM 組合語言的熟練度，也加深了對於底層 CPU 工作原理的理解。這種技能對於任何需要精確控制硬體運作的開發工作都是非常寶貴的。

這次的實驗增加了我的組合語言程式技巧，也加深了我對處理器運作和記憶體管理的理解。這些經驗對我來說很讚，讓我對未來開發更複雜的系統或進行最佳化有了更多的信心。但突然要我們寫排序真的讓我差點哭出來...