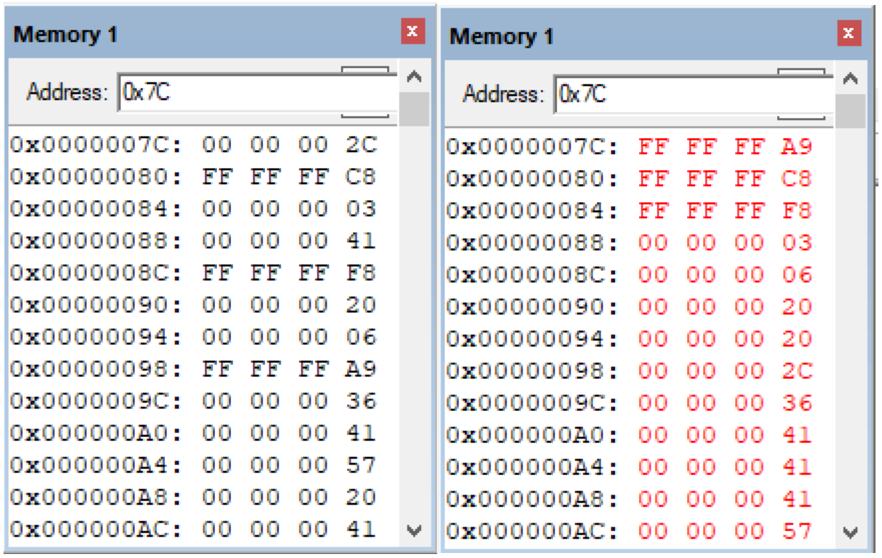
CS 2208B
Ali Al-Musawi Lab 5

## Question 2

In this question, we add the following code segment (see Assembly_Lab_5_Q2.s). Note that we used a stack that **grows down** and **points to occupied memory**.

```
sortTwo             LDR r13, =endOfArray        ;Find a zeroed block in memory
                    STR r8, [r13, #4]!          ;Free up register r8 using the stack
                    STR r9, [r13, #4]!          ;Free up register r9 using the stack
                    LDR r8, [r6]                ;r8 stores a[j]
                    LDR r9, [r7]                ;r9 stores a[j+1]
                    CMP r8, r9                  ;Compare a[j] and a[j+1]
                    STRGT r8, [r7]              ;If a[j] > a[j+1] then a[j] = *(a[j+1])
                    STRGT r9, [r6]              ;If a[j] > a[j+1] then a[j+1] = *(a[j])
                    LDR r9, [r13], #-4          ;Retrieve the old value of register r9
                    LDR r8, [r13], #-4          ;Retrieve the old value of reigster r8
                    MOV r15, r14                ;Return from sortTwo
```

After running, the program, we check the memory map. We notice that the array is now sorted. Below is a comparison of the array in memory before and after the sort.



Address: 0x7C

```
0x0000007C: 00 00 00 2C
0x00000080: FF FF FF C8
0x00000084: 00 00 00 03
0x00000088: 00 00 00 41
0x0000008C: FF FF FF F8
0x00000090: 00 00 00 20
0x00000094: 00 00 00 06
0x00000098: FF FF FF A9
0x0000009C: 00 00 00 36
0x000000A0: 00 00 00 41
0x000000A4: 00 00 00 57
0x000000A8: 00 00 00 20
0x000000AC: 00 00 00 41
```

Address: 0x7C

```
0x0000007C: FF FF FF A9
0x00000080: FF FF FF C8
0x00000084: FF FF FF F8
0x00000088: 00 00 00 03
0x0000008C: 00 00 00 06
0x00000090: 00 00 00 20
0x00000094: 00 00 00 20
0x00000098: 00 00 00 2C
0x0000009C: 00 00 00 36
0x000000A0: 00 00 00 41
0x000000A4: 00 00 00 41
0x000000A8: 00 00 00 41
0x000000AC: 00 00 00 57
```

Left: Array Memory Map Before Sort. Right: Array Memory Map After Sort.

## Question 3

In this question, we have used a stack that **grows up** and **points to occupied memory**. Here are the adjustments we made:

```
                    ADR r13, stack              ;Set up a pointer to the stack
                    STR    PC, [r13, #-4]!      ;pre-decrement the stack pointer
                    B  sortTwo                  ;call sortTwo(*a[j],*a[j+1])
; After several lines of code…
sortTwo             STR r8, [r13, #-4]!         ;Free up register r8 using the stack
                    STR r9, [r13, #-4]!         ;Free up register r9 using the stack
                    LDR r8, [r6]                ;r8 stores a[j]
                    LDR r9, [r7]                ;r9 stores a[j+1]
                    CMP r8, r9                  ;Compare a[j] and a[j+1]
                    STRGT r8, [r7]              ;If a[j] > a[j+1] then a[j] = *(a[j+1])
                    STRGT r9, [r6]              ;If a[j] > a[j+1] then a[j+1] = *(a[j])
                    LDR r9, [r13], #4           ;Retrieve the old value of register r9
                    LDR r8, [r13], #4          ;Retrieve the old value of reigster r8
                    LDR r12, [SP], #4          ;Retrieve the pipelined PC location
                    SUB PC, r12, #4            ;Adjust the pipelined PC location and save it

a                   DCD 44,-56,3,65,-8,32,6,-87,54,65,87,32,65
endOfArray          SPACE 4
                    SPACE 32                   ;reserved room for the stack to grow
stack               DCD 0x0                    ;the base of the stack
```

For the full code, see Assembly_Lab_5_Q3.s. Note that the result is identical to the memory map above.

## Question 4

Note this can be achieved using 1 line of code in assembly.  We add the following line right after the function call `B   sortTwo:`

```
BLE endOuter        ;break if no swap occurs i.e. the array is sorted.
```

The logic of this line is as follows. To do a swap, we compare the values **`a[j], a[j+1]`** using the instruction **`CMP`**, which sets the flags. Right after we exit, we check the flags. If the result of the comparison results in no swap, then we break. We test the code on a sorted array. See Assembly_Lab_5_Q4.s for the full code.