**Laboratory Assignment 1**

**Purpose:**

The purpose of this laboratory assignment is to reinforce concepts covered in Chapter 2 of Computer Organization and Design: The Hardware/Software Interface ARM Edition, by D. Patterson and J. Hennessy, and to further your familiarity with the ARM instruction set architecture (ISA).

**Instructions:**

You may work individually or in small groups to complete the laboratory assignment. After completing the assignment please submit your laboratory report (one per person or group) electronically via email (to [bmichael@nps.edu](mailto:bmichael@nps.edu)) or in printed form (in class or at my office).

The ARM assembly language subroutine below implements the well-known bubble sort algorithm to solve—in a non-optimal manner—the problem of sorting a list of elements.

START STMDB r13!,{r0-r7}

MOV r3,r0

MOV r4,r1

LABEL\_1 CMP r4,r3

BEQ LABEL\_5

MOV r3,r0

LABEL\_2 CMP r4,r3

BEQ LABEL\_4

ASL r7,r3,#2

ADD r7,r7,r2

LDMIA r7,{r5,r6}

CMP r6,r5

BGE LABEL\_3

STR r6,[r7]

STR r5,[r7,#4]

LABEL\_3 ADD r3,r3,#1

B LABEL\_2

LABEL\_4 ADD r4,r4,#-1

MOV r3,r0

B LABEL\_1

LABEL\_5 LDMIA r13!{r0-r7}

MOV r15,r14

Note: STMDB sp!, reglist pushes registers onto a stack. The LDMIA instruction loads multiple registers in a single instruction. See the tutorial on subroutines attached to this laboratory assignment. See the addendum to the laboratory assignment for information about these two commands and writing subroutines for ARM processors. Another good source of reference information is available at <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489e/Babefbce.html>.

The bubble sort algorithm consists of a sequential pairwise comparison from left to right, swapping values if the left element is greater than the right element, repeating this process until all of the elements in the list are ordered in ascending numeric value. For instance, consider this list of integers

(5 60 22 319 77 49 514)

The steps taken in the bubble sort algorithm would be:

First pass:

Step 1.1 (**5 60** 22 319 77 49 514) → (**5 60** 22 319 77 49 514)

Step 1.2 (5 **60 22** 319 77 49 514) → (5 **22 60** 319 77 49 514) ; 60 > 22, swap

Step 1.3 (5 22 **60 319** 77 49 514) → (5 22 **60 319** 77 49 514)

Step 1.4 (5 22 60 **319 77** 49 514) → (5 22 60 **77 319** 49 514) ; 319 > 77, swap

Step 1.5 (5 22 60 77 **319 49** 514) → (5 22 60 77 **49 319** 514) ; 319 > 49, swap

Step 1.6 (5 22 60 77 49 **319 514**) → (5 22 60 77 49 **319 514**)

Second pass:

Step 2.1 (**5 22** 60 77 49 319 514) → (**5 22** 60 77 49 319 514)

Step 2.2 (5 **22 60** 77 49 319 514) → (5 **22 60** 77 49 319 514)

Step 2.3 (5 22 **60 77** 49 319 514) → (5 22 **60 77** 49 319 514)

Step 2.4 (5 22 60 **77 49** 319 514) → (5 22 60 **49 77** 319 514) ; 77 > 49, swap

Step 2.5 (5 22 60 49 **77 319** 514) → (5 22 60 49 **77 319** 514)

Step 2.6 (5 22 60 49 77 **319 514**) → (5 22 60 49 77 **319 514**)

Third pass:

Step 3.1 (**5 22** 60 49 77 319 514) → (**5 22** 60 49 77 319 514)

Step 3.2 (5 **22 60** 49 77 319 514) → (5 **22 60** 49 77 319 514)

Step 3.3 (5 22 **60 49** 77 319 514) → (5 22 **49 60** 77 319 514) ; 60 > 49, swap

Step 3.4 (5 22 49 **60 77** 319 514) → (5 22 49 **60 77** 319 514)

Step 3.5 (5 22 49 60 **77 319** 514) → (5 22 49 60 **77 319** 514)

Step 3.6 (5 22 49 60 77 **319 514**) → (5 22 49 60 77 **319 514**)

Fourth pass:

Step 4.1 (**5 22** 49 60 77 319 514) → (**5 22** 49 60 77 319 514)

Step 4.2 (5 **22 49** 60 77 319 514) → (5 **22 49** 60 77 319 514)

Step 4.3 (5 22 **49 60** 77 319 514) → (5 22 **49 60** 77 319 514)

Step 4.4 (5 22 49 **60 77** 319 514) → (5 22 49 **60 77** 319 514)

Step 4.5 (5 22 49 60 **77 319** 514) → (5 22 49 60 **77 319** 514)

Step 4.6 (5 22 49 60 77 **319 514**) → (5 22 49 60 77 **319 514**)

The last pass (i.e., fourth pass in this example) is necessary to determine that all of the elements are in ascending order.

Here is a pseudocode representation of the bubble sort algorithm[[1]](#footnote-1):

**procedure** bubbleSort( A : list **of** sortable items )

n = length(A)

**repeat**

swapped = **false**

**for** i = 1 **to** n-1 inclusive **do**

/\* **if** this pair **is** out **of** order \*/

**if** A[i-1] > A[i] **then**

/\* swap them **and** remember something changed \*/

swap( A[i-1], A[i] )

swapped = **true**

**end** **if**

**end** **for**

**until** **not** swapped

**end** **procedure**

*1. Which of the instructions contained in the function indicate that the subroutine was written for use on an ARM processor?*

*2. Explain how the subroutine implements the bubble sort algorithm.*

1. From https://en.wikipedia.org/wiki/Bubble\_sort [↑](#footnote-ref-1)