Problem FC-4 (2 parts) **Loops in MIPS**

Part A: Create an assembly language function that computes the largest, smallest, and average values in a variable sized array. Here the values are placed in an array in static memory using the `.word` assembler directive. The number of values is implied by the address of the next value in static memory at label `Next:`. An assembly language shell program, shown below, is provided. Place the minimum value in $4, the maximum in $5, and the average in $6.

```
# This program finds the largest, smallest, and average values for
# an integer array.

.data
Array:.word 243, 459, 896, 535, 264, 698, 268, 281, 921, 886
      .word 864, 215, 781, 151, 435, 128, 276, 336, 790, 825
      .word 501, 725, 835, 160, 300, 095, 481, 282, 515, 282
      .word 662, 770, 776, 998, 758, 447, 758, 272, 015, 398
      .word 042, 645, 565, 265, 105, 778, 739, 148, 309, 960
      .word 903, 067, 469, 126, 673, 864, 658, 333, 170, 987
      .word 565, 228, 235, 477, 568, 254, 628, 421, 788, 012
      .word 246, 170, 746, 892, 586, 875, 055, 850, 885, 828
      .word 717, 797, 971, 862, 269, 082, 824, 728, 650, 470
      .word 740, 522, 232, 648, 323
Next: .word 00

# $1 = array pointer, $2 = array size, $3 = input, $4, = min,
# $5 = max, $6 = avg, $7 = pred
.text
IN4B:       addi  $1, $0, Array     # set memory base
            addi  $2, $0, Next      # load end condition
            sub   $2, $2, $1        # compute size in bytes
            lw    $4, Array($0)     # initialize min
            add   $5, $4, $0        # initialize max
            add   $6, $4, $0        # initialize average
            addi  $1, $0, 4         # first element offset
Loop:       lw    $3, Array($1)     # load next value
            add   $6, $6, $3        # add to running sum
            slt   $7, $3, $4        # compare input to min
            beq   $7, $0, Skip1     # skip is >=
            add   $4, $3, $0        # otherwise update min
Skip1:      slt   $7, $5, $3        # compare max to input
            beq   $7, $0, Skip2     # skip if >=
            add   $5, $3, $0        # otherwise update max
Skip2:      addi  $1, $1, 4         # adjust array pointer
            bne   $1, $2, Loop      # loop until end is reached
            sra   $2, $2, 2         # compute size in words
            div   $6, $2            # compute average
            mflo  $6                # move result
            jr      $31             # return to operating system
```

Part B: Create an assembly language function that computes the averages of even and odd numbers in a variable sized array. Here the values are placed in an array in static memory using the .word assembler directive. The number of values is implied by the address of the next value in static memory at label Next:. An assembly language shell program, shown below, is provided. Assume at least one even and one odd value occurs in the list. Your program should place the average of all even numbers in $4 and the average of all odd numbers in $5.

```
# This program the average of all even numbers and the average of all odd
# numbers in an integer array.  The even number average is placed in
# $4 and the odd number average is placed in $5.
# Assumes there is at least one even and one odd number in the array.
.data
Array:.word 243, 459, 896, 535, 264, 698, 268, 281, 921, 886
      .word 864, 215, 781, 151, 435, 128, 276, 336, 790, 825
      .word 501, 725, 835, 160, 300, 095, 481, 282, 515, 282
      .word 662, 770, 776, 998, 758, 447, 758, 272, 015, 398
      .word 042, 645, 565, 265, 105, 778, 739, 148, 309, 960
      .word 903, 067, 469, 126, 673, 864, 658, 333, 170, 987
      .word 565, 228, 235, 477, 568, 254, 628, 421, 788, 012
      .word 246, 170, 746, 892, 586, 875, 055, 850, 885, 828
      .word 717, 797, 971, 862, 269, 082, 824, 728, 650, 470
      .word 740, 522, 232, 648, 323
Next: .word 00

# $1 = array index, $2 = upper limit address, $3 = current element,
# $4 = running sum of even numbers, $5 = running sum of odd numbers
# $6 = count of even numbers, $7 = count of odd numbers
# $8 = predicate register
.text
EvenOddAvgs: addi $1, $0, Array     # set memory base
             addi    $2, $0, Next    # load end address
             addi    $4, $0, 0       # init even running sum
             addi    $5, $0, 0       # init odd running sum
             addi    $6, $0, 0       # init even count
             addi    $7, $0, 0       # init odd count
Loop:        lw      $3, 0($1)       # load in current element
             andi    $8, $3, 1       # mask all but LSB and place in $8
             beq     $8, $0, Even    # If $3 is even, add to even sum and count
             add     $5, $5, $3      # else add $3 to odd running sum
             addi    $7, $7, 1       #      and inc count of odd numbers
             j       Continue        #      and continue with end of loop
Even:        add   $4, $4, $3        # $3 is even, add it to even running sum
             addi  $6, $6, 1         #      and inc count of even numbers
Continue:    addi    $1, $1, 4       # inc array index
             bne     $1, $2, Loop    # if did not hit end address, yet, loop
             div     $4, $6          # divide even running sum by even count
             mflo    $4              # put even number average in $4
             div     $5, $7          # divide odd running sum by even count
             mflo    $5              # put odd number average in $5
             jr      $31             # return to operating system
```