LAB - 5

**AIM of the LAB**: Analyse and evaluate different Array operations using ARM processor.

**OBJECTIVE - (1)** : find the largest / smallest number in each array of size N.

**(1) PSEUDOCODE:**

1) Load the address of 'count' into R0
2) Load the value at that address (count = 7) into R1 (R1 = N)
3) Load the address of the array into R2
4) Load the 1st element of the array into R3 (current max / current min = 1st element)
5) Repeat until all the N elements are processed.:
   (a) Decrement the R1 by 1
   (b) if R1 == 0 , jump to fwd (end)
   (c) Load the next element from the array into R4.
   (d) Compare R3 and R4
** (e) if R3 > R4, go back to step a [skip update] } for largest Number
** (e) if R3 < R4 , go back to step a [skip update] } for Smallest Number
   (f) Else, update R3 = R4 (new max / new min)
   (g) Repeat

6) At 'fwd':
(a) Store 'R3' (Smallest/Largest Number) into memory labelled as "Result".
(b) Exit.

## (2) ASSEMBLY CODE (with descriptions):

```
Mov R0, #address of Count   ; R0 ← address of Count
LDR R1, [R0]                ; R1 ← Count (eg - 7)
mov R2, #address of array   ; R2 ← base address of array
LDR R3, [R2], #4            ; R3 ← first element of array,
                              R2 += 4

Loop:
SUBS R1, R1, #1             ; R1 = R1 - 1, updating flags
BEQ END                     ; if R1 == 0, exit loop

LDR R4, [R2], #4            ; R4 ← next element of array
                              R2 += 4
CMP R3, R4                  ; compare R3 and R4
BGT Loop *****              ; if R3 > R4, go back to Loop
BLT Loop *****              ; if R3 < R4, keep R3
                            ; 1st one for largest Number
                              and 2nd one for smallest
                              Number
```

```
MOV R3, R4      ; False, update R3 = R4 (new smaller version)
                ;// update max

B LOOP          ; Repeat Loop

END:
MOV R5, #address_of_smaller   ; R5 ← address to store result
STR R3, [R5]                  ; store smaller number in memory
                              ;// largest
B END                         ; infinite loop to end execution.
```

## ③ INPUT / OUTPUT ANALYSIS :

### (a) for Largest Number :

Array : [21, 53, 50, 69, 16, 79, 52]

Now, R3 = 21

- 53 > 21 → R3 = 53
- 50 < 53 → skip
- 69 > 53 → R3 = 69
- 16 < 69 → skip
- 79 > 69 → R3 = 79
- 52 < 79 → skip

So, final largest number : 79 (0x4F)

stored at :

**0x40000000 : 0x4F000000**

Data stored :

0x00000038 → 0x15
3C → 0x35
40 → 0x32
44 → 0x45
48 → 0x10
4C → 0x4F
50 → 0x34

Result stored :

0x40000000 → ?

(b) for Smallest Number:

Array: [21, 53, 50, 69, 16, 79, 52]

Now, R3 = 21

- 53 > 21 → Skip
- 50 > 21 → Skip
- 69 > 21 → Skip
- 16 < 21 → R3 = 16
- 79 > 16 → Skip
- 52 > 16 → Skip

So, final Smallest Number:
16 (0x10)

So, output will be at:
0x40000000 → 0x10000000

OBJECTIVE - ②: Separet Even and Odd Numbers in an array of size N.

① PSEUDOCODE:

1) Start
2) Load the Count of Numbers into R1
3) Set Pointer R3 to base of input array
4) Set Pointer R4 to base of even Number reserve location
5) Set Pointed R5 to base of odd numbes reserve location
6) Repeat until the Counter becomes 0:
(a) Load Next Number from array using R3
(b) Check if No. is even (& with 1):
if (No. & 1 == 0) ; Even

Store the number at even location, increment the even pointer (R4)

Else ; odd

Store the No. at odd location, increment the odd Pointer (R5)

(c) Decrement Count

(d) End Repeat

7) Stop.

## (2) ASSEMBLY CODE (with description):

```
AREA prog2, CODE, READONLY
ENTRY
START
    ldr r0, =Count ; R0 → address of Count variables
    ldr r1, [r0]   ; R1 ← Count = 7
    ldr r3, =array ; R3 ← base address of input array
    ldr r4, =even  ; R4 ← base address to store even numbers
    ldr r5, =odd   ; R5 ← base address to store odd numbers

back
    ldr r6, [r3], #4 ; R6 ← current element, R3 += 4
    ands r7, r6, #1  ; R7 ← R6 & 1 (check LSB)
    beq fwd          ; if Even, jump to fwd
```

```
str r6, [r5], #4    ; if ODD → Store in an array, R5+=4
b. fwd1             ; Skip even storage

fwd
str r6, [r6], #4    ; Store even number in R4, R4+=4

fwd1
subs r1, r1, #1     ; Decrement Count
bne back            ; if count ≠0, repeat

exit
b exit              ; infinite loop to stop execution
```

## ③ INPUT / OUTPUT ANALYSIS :

Array : [0x 15, 0x 35, 0x 32, 0x 45, 0x 10, 0x 4f, 0x 34]
→ in Decimal : [12, 53, 50, 64, 16, 79, 52] Stored in 0x00000038

Now, in Even Numbers data stored is:

50, 16, 52 → 0x32, 0x10, 0x34

Stored in 0x40000000

Now, in odd Numbers data stored is:

21, 53, 64, 79 → 0x15, 0x35, 0x45, 0x4f

Stored in 0x4000000C

**CONCLUSION:** This lab focused on implementing ARMv7 assembly programs to find the largest, smallest & separate even and odd numbers from an array. By using conditional branches & memory addressing, I learned how simple instructions can perform powerful data operations. The experience enhanced my understanding of register manipulation, flag usage, & memory management in low level programming.

**POST LAB:**

(Q1) Explain briefly condition codes (flags) of ARM processor.

Soln) The ARM processor uses 4 main condition flags in the CPSR:

→ N (Negative): Set if the result the of an operation is negative

→ Z (Zero): Set if the result is zero.

→ C (Carry): Set if there is a carry out from an addition or a borrow from an subtraction

→ V (Overflow): Set if there is a signed overflow.

(Q2) Which ~~instructions~~ Condition flags (flags) is considered for the following branch instruction?

(a) B label → No label are checked

(b) BEQ label → N flag, Zero flag

(c) BLT label → Z flag only
→ N flag & V flag, where V ≠ N.