Computer Organization and Architecture (EET2211)

LAB V: Analyze and evaluate different Array operations using ARM processor.

Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar

Branch: CSE		**Section: 2241026**	
S. No.	**Name**	**Registration No.**	**Signature**
6	Dinanath Dash	2241004161	

Marks: ______/10

Remarks:

Teacher's Signature

I. OBJECTIVE:

1. Find the largest/smallest number in an array of size N.
2. Separate Even numbers and odds numbers in an array of size N.

II. PRE-LAB

- Explain the addressing modes involved in instructions.

For each objective in prelab describe the following points:

- Write the assembly code with description (ex. Mov ax,3000h – ax<-3000h)
- Examine & analyze the input/output of assembly code.

For Obj. 1 →

```
.data
 array:
     .word 8, 8, 2, 10, 3   @ Define an array with the elements 5,8,2,10,3
 array_size:
     .word 5                @ Define the size of array
.text
.global main
 main:
     ldr r0, =array         @ load the address of 'array' in register r0
     ldr r1, =array_size    @ load the address of array_size into r1
     ldr r1, [r1]           @ load the value at r1 into r1
     ldr r2, [r0]           @ load the first element of array into r2
     add r0, r0, #4         @ increment r0 to point to the next element in 'array'
 loop:
     cmp r1, #0             @ compare the remaining array size (r1) with 0
     beq end_loop           @ if r1 is 0, branch to end_loop
     ldr r3, [r0]           @ load the next element of array into r3
     cmp r3, r2             @ compare the value in r3 with the current maximum in r2
     movgt r2, r3           @ if r3 is greater than r2, move r3 to r2
     add r0, r0, #4         @ increment r0 to point to the next element in array
     subs r1, r1, #1        @ decrement the remaining array size in r1 by 1 & update flags
     B loop                 @ branch to loop to continue processing
 end_loop:
     my_exit: b my_exit
.end
```

For Obj. 2→

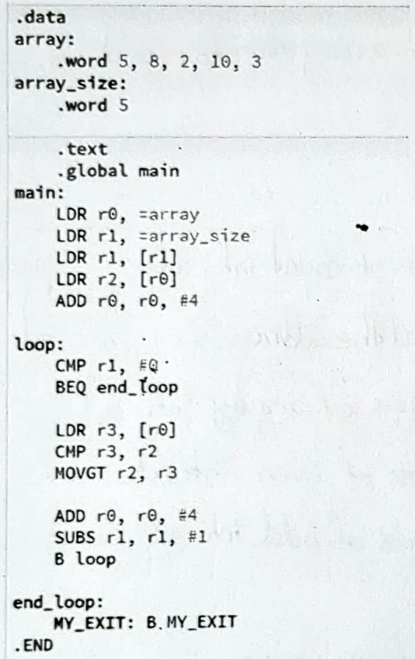
```
.data
count:
    .word 0x07          @ Define a word 'count' with value 7
array:
    .word 0x15, 0x35, 0x32, 0x48, 0x10, 0x4f, 0x34 @ Define an array with the
                                                      given elements
even:
    .word 0,0,0,0,0,0,0 @ Define an array for even ~~words~~ numbers
odd:
    .word 0,0,0,0,0,0,0 @ Define an array ~~for~~ for odd ~~no~~ numbers
.global _start
_start:
    ldr r0, =count      @ load the address of count into r0
    ldr r1, [r0]        @ load the value at the address in r0 into r1
    ldr r3, =array      @ load the address of array into r3
    ldr r4, =even       @ load the address of even into r4
    ldr r5, =odd        @ load the address of odd into r5
back:
    ldr r6, [r3], #4    @ load a word from array into r6 & increment r3 by 4
    ands r7, r6, #1     @ perform bitwise and on r6 with 1, result in r7
    beq fwd             @ if result is zero, branch to fwd
    str r6, [r5], #4    @ store r6 in odd array & increment r5 by 4
    b fwd1              @ branch to fwd1 to continue processing
fwd:
    str r6, [r4], #4    @ store r6 in even array & increment r4 by 4
fwd1:
    subs r1, r1, #1     @ subtract 1 from r1, & update flags
    bne back            @ if r1 is not zero, branch to back to continue processing
exit: b exit
.end
```

III. LAB

Note: For each objective do the following job and assessment:

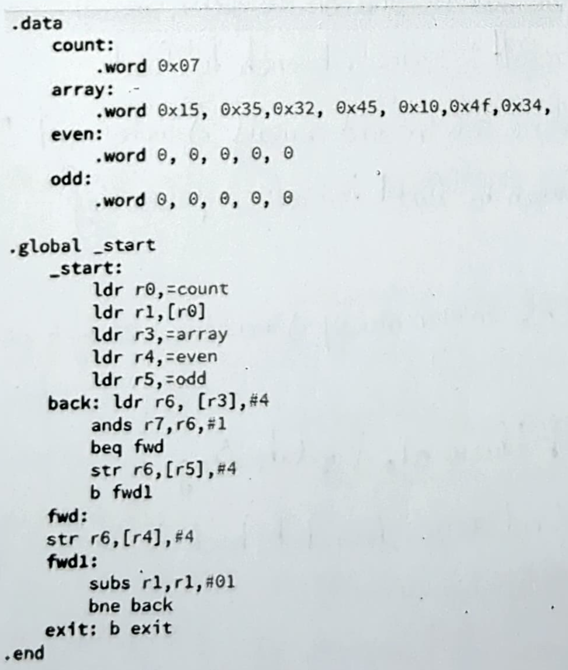
- Screen shots of the Assembly language program (ALP)

For Obj. 1:



```
.data
array:
    .word 5, 8, 2, 10, 3
array_size:
    .word 5

    .text
    .global main
main:
    LDR r0, =array
    LDR r1, =array_size
    LDR r1, [r1]
    LDR r2, [r0]
    ADD r0, r0, #4

loop:
    CMP r1, #0
    BEQ end_loop

    LDR r3, [r0]
    CMP r3, r2
    MOVGT r2, r3

    ADD r0, r0, #4
    SUBS r1, r1, #1
    B loop

end_loop:
    MY_EXIT: B MY_EXIT
.END
```

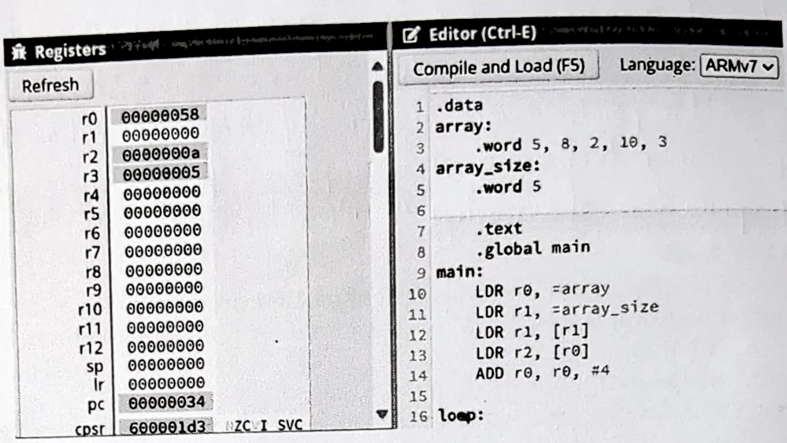
For Obj. 2:



```
.data
    count:
        .word 0x07
    array:
        .word 0x15, 0x35,0x32, 0x45, 0x10,0x4f,0x34,
    even:
        .word 0, 0, 0, 0, 0
    odd:
        .word 0, 0, 0, 0, 0

.global _start
    _start:
        ldr r0,=count
        ldr r1,[r0]
        ldr r3,=array
        ldr r4,=even
        ldr r5,=odd
    back: ldr r6, [r3],#4
        ands r7,r6,#1
        beq fwd
        str r6,[r5],#4
        b fwd1
    fwd:
    str r6,[r4],#4
    fwd1:
        subs r1,r1,#01
        bne back
    exit: b exit
.end
```

- Observations (with screen shots)

For Obj. 1:

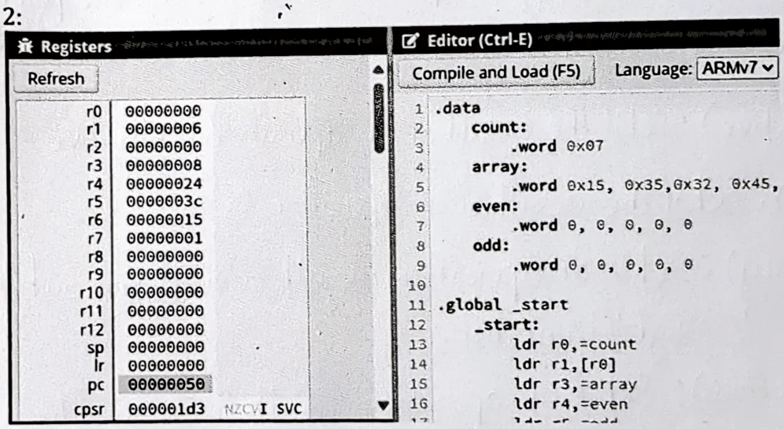


```
r0    00000058
r1    00000000
r2    0000000a
r3    00000005
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000034
cpsr  600001d3  NZCVI SVC
```

```
.data
array:
    .word 5, 8, 2, 10, 3
array_size:
    .word 5

    .text
    .global main
main:
    LDR r0, =array
    LDR r1, =array_size
    LDR r1, [r1]
    LDR r2, [r0]
    ADD r0, r0, #4

loop:
```

Fig. 1. Execution results using immediate addressing to find the largest/smallest number in an array of size N.

For Obj. 2:



```
r0    00000000
r1    00000006
r2    00000000
r3    00000008
r4    00000024
r5    0000003c
r6    00000015
r7    00000001
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000050
cpsr  000001d3  NZCVI SVC
```

```
.data
    count:
        .word 0x07
    array:
        .word 0x15, 0x35,0x32, 0x45,
    even:
        .word 0, 0, 0, 0, 0
    odd:
        .word 0, 0, 0, 0, 0

.global _start
    _start:
        ldr r0,=count
        ldr r1,[r0]
        ldr r3,=array
        ldr r4,=even
```

Fig. 2. Execution results using immediate addressing to separate Even numbers and odds numbers in an array of size N.

From this result I have observed......

Input:

Sl. No.	Memory Location	Operand (Data)
1	r0	Array = 5, 8, 2, 10, 3
	r1	Array size = 5
2	r0	Count = 0x07
	r3	Array = 0x15, 0x35, 0x32, 0x45
	r4	Even = 0, 0, 0, 0
	r5	Odd = 0, 0, 0, 0

Output:

Sl. No.	Memory Location	Operand (Data)
1	r2	a
2		

IV. CONCLUSION

The analysis of array operations on an ~~arm~~ ARM processor highlights its efficiency due to the RISC architecture, advanced instruction set, & pipeline capabilities. ARM processors excel in performance and energy efficiency, making the ideal for embedded systems. Optimization techniques and compiler support further enhance the execution of array operations, showcasing ARM's suitability for tasks involving extensive data manipulation

V. POST LAB

1. Explain briefly condition codes (flags) of ARM processor.
2. Which condition codes (flags) is considered for the following branch instructions?
   a. B Label
   b. BEQ label
   c. BLT label

Ans → 1) ARM processors use condition codes, also known as flags, to indicate the results of arithmetic & logical operations. These flags are part of the program status register (CPSR) & influence conditional execution of instructions.

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

ii) Z (zero): Set if the result of an operation is zero

iii) C (carry): Set if an operation results in a carry out from the most significant bit.

iv) V (Overflow): Set if an operation results in an overflow, meaning the result is too large to be represented in the available number of bits.

2) For the given branch instructions in an ARM processor, the relevant condition codes (flags) considered are:

a) B Label: This is an unconditional branch instruction. It does not consider codes (flags) and always branches to the specified label.

b) BEQ Label: This is a condititional branch instruction that branches if the Zero (Z) flag is set. It means the branch occurs if the result of the last operation was zero.

c) BLT Label: This is a conditional branch instruction that branches if the result is less than.