

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.

1. Immediate Addressing: is used to load constant values into registers.

Ex: mov r4, #0x00

2. Literal Addressing: is used to load addresses of data labels into registers.

Ex: ldr r2, =array

3. Register Indirect Addressing: is used to access values from memory locations pointed by registers, often with part movement to step through arrays.

Ex: ldr r3, [r2]

4. Logical Immediate Addressing: is used for bitwise operation with immediate values.

Ex: ands r7, r6, #1

5. Memory Direct Addressing: is used to store values from registers into memory locations.

Ex: str r6, [r5], #4

6. Branch Instructions: 'bgt' & 'bne' branches to specified label if comparison is greater & result is not zero respectively.

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.

Objective 1:

=> @Largest number from array

• global - start

-start:

ldr r0, =count

ldr r1, [r0]

mov r4, #0x00

ldr r2, =array

back: ldr r3, [r2], #4

cmp r4, r3

bgt fwd // signed int comparison gives greater than

mov r4, r3

fwd: subs r1, r1, #01

bne back

str r4, [r2]

exit: b exit

• data

count: .word 0x05

array: .word 0x15, 0x35, 0x45, 0x10, 0x4f

=> @Smallest:

• global - start

-start: ldr r0, =count

ldr r1, [r0]

mov r4, #0x0ff

ldr r2, =array

back: ldr r3, [r2], #4

cmp r4, r3

blt fwd

mov r4, r3

fwd: subs r1, r1, #01

bne back

; str r4, [r2]

; exit: b exit

; • data

; count: .word 0x05

; array: .word 0x15, 0x35, 0x45, 0x10, 0x4f

Objective 2:

• global -start

-Start:

@ separate even and odd numbers from array

ldr r0, =count

ldr r1, [r0]

ldr r3, =array @ r3 = base address of array = array[0]

ldr r4, =even @ r4 = base address of even data locations. = even[0]

ldr r5, =odd @ r5 = base address of odd data locations = odd[0]

book: ldr r6, [r3], #4

adds r7, r6, #1

beg fwd

str r6, [r5], #4

b fwd1

fwd: str r6, [r4], #4

fwd1: subs r1, r1, #01

bne book

exit: b exit

.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

III. LAB

Objective 1 (Largest) :

The screenshot shows the Quartus II software interface with the following details:

- Registers:** Shows the state of registers R0 through R11.
- Editor (Ctrl-E):** Displays the assembly code for finding the largest number from an array. The code uses registers r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, and r11. It initializes r0 to 0x00000038, then iterates through the array, comparing each element with r0 and updating r0 if the current element is larger. Finally, it prints the value of r0 and exits.
- Messages:** Shows the compilation command and output: "Compile succeeded".

```

1. ORIGIN = 0x00000000
2. global _start
3. _start:
4.     ;largest number from a given array
5.     ldr r0, #0
6.     ldr r1, [r0]
7.     mov r4, #0x00
8.     ldr r2, [r1]
9.     back: ldr r3, [r2], #4
10.    cmp r4, r3
11.    bgt fwd //signed integer comparison gave greater than
12.    mov r4, r3
13.    fwd: subs r1, r1, #1
14.    bne back //comparison not equal or non zero result
15.    str r4, [r2]
16.    exit: b exit
17.
18. .data
19. count: .word 0x05
20. array: .word 0x15, 0x35, 0x45, 0x10, 0x4f
21.

```

The screenshot shows the Quartus II software interface with the following details:

- Registers:** Shows the state of registers R0 through R11.
- Memory (Ctrl-M):** Displays the memory dump for address 0x00000030. The memory contains the following data:

Address	Memory contents and ASCII
0x00000030	00000038
0x00000031	00000000
0x00000032	00000050
0x00000033	0000004f
0x00000034	00000040
0x00000035	00000090
0x00000036	00000034
0x00000037	00000000
0x00000038	00000000
0x00000039	00000000
0x0000003a	00000000
0x0000003b	00000000
0x0000003c	00000000
0x0000003d	00000000
0x0000003e	00000000
0x0000003f	00000000
0x00000040	00000000
0x00000041	00000000
0x00000042	00000000
0x00000043	00000000
0x00000044	00000000
0x00000045	00000000
0x00000046	00000000
0x00000047	00000000
0x00000048	00000000
0x00000049	00000000
0x0000004a	00000000
0x0000004b	00000000
0x0000004c	00000000
0x0000004d	00000000
0x0000004e	00000000
0x0000004f	00000000
0x00000050	00000000
- Messages:** Shows the compilation command and output: "Compile succeeded".

From this result I have observed.....

Input:

Sl. No.	Memory Location	Operand (Data)
1	0x000000038	0x05
2	0x00000003C	0x15
3	0x000000040	0x35
4	0x000000044	0x45
5	0x000000048	0x10
6	0x00000004C	0x4f

Output:

Sl. No.	Memory Location	Operand (Data)
1	0x00000050	0x4f

Objective 1 (Smallest) :

(Smallest) :

The screenshot shows the Quartus Prime software interface. The top menu bar includes options like 'File', 'Help', and various debugging tools. The left sidebar contains tabs for 'Registers', 'Call stack', 'Trace', 'Breakpoints', 'Watchpoints', 'Symbols', and 'Counters'. Below that is a 'Number Display Options' section with a 'Word' size selection. The main workspace displays assembly code for an ARMv7 processor. The code initializes registers r0 and r1, sets up a loop to find the smallest number from an array, and includes a 'back' label for comparison logic. It also defines sections for data and includes memory dump and search functions. The bottom status bar shows messages about code loading and compilation success.

```
0E.. Jagadeewar Patroj 2241016309
.global _start
_start:
    ;smallest number from a given array
    ldr r0, =count
    ldr r1, [r0]
    mov r4, #0x0f
    ldr r2, =array
back: ldr r3, [r2], #4
    cmp r4, r3
    blt fwd //signed integer comparision gave less than
    mov r4, r3
fwd: subs r1, r1, #01
    bne back //comparision not equal or non zero result
    str r4, [r2]
exit: b exit

.data
count: .word 0x05
array: .word 0x15, 0x35, 0x45, 0x10, 0x4f
```

Code and data loaded from ELF executable into memory. Total size is 60 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asm7d10qz.s.o
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/asm7d10qz.elf work/asm7d10qz.s.o
Compile succeeded.

The screenshot shows the QEMU ARM debugger interface. The top menu bar includes 'File' and 'Help'. Below the menu is a toolbar with buttons for 'Step Into' (F2), 'Step Over' (Ctrl-F2), 'Step Out' (Shift-F2), 'Continue' (F3), 'Stop' (F4), 'Restart' (Ctrl-R), and 'Reload' (Ctrl-Shift-L). The main window has tabs for 'Registers', 'Memory (Ctrl-M)', and 'Registers' again. The 'Registers' tab displays a list of general-purpose registers (r0 to pc) with their current values. The 'Memory' tab shows a memory dump with columns for Address, Memory contents, and ASCII representation. The bottom of the screen features a command bar with buttons for 'Call stack', 'Trace', 'Breakpoints', 'Watchpoints', 'Symbols', 'Counters', and 'Settings'. It also includes dropdown menus for 'Number Display Options' (Size: Word, Format: Hexadecimal), 'Memory words per row' (4), and tabs for 'Editor (Ctrl-E)', 'Disassembly (Ctrl-D)', and 'Memory (Ctrl-M)'.

From this result I have observed.....

Input:

Sl. No.	Memory Location	Operand (Data)
1	00000038	0x05
2	0000003C	0x15
3	00000040	0x35
4	00000044	0x45
5	00000048	0x10
6	0000004C	0x4f

Output:

Sl. No.	Memory Location	Operand (Data)
1	00000050	0x10

Objective 2 :

The screenshot shows the Atmel Studio 7 IDE interface. The top menu bar includes 'Stopped' (F5), 'Step Into' (F2), 'Step Over' (Ctrl+F2), 'Step Out' (Shift+F2), 'Continue' (F8), 'Stop' (F4), 'Restart' (Ctrl+R), 'Reload' (Ctrl+Shift+R), 'File' (dropdown), and 'Help' (dropdown). Below the menu is a toolbar with icons for Registers, Call stack, Trace, Breakpoints, Watchpoints, Symbols, Counters, and Settings.

The main window displays the assembly code for 'Lab5Obj2.s':

```
1 01. Jagadeewar_Patrol 2241016309
2 .global _start
3 _start:
4     ;separate even nums and odd nums in a given array of size 8
5     ldr r0, =count
6     ldr r1, [r0]
7     ldr r3, array @ r3 = base address of array = array[0]
8     ldr r4, =even @ r4 = base address of even data locations =even[0]
9     ldr r5, =odd @ r5 = base address of odd data location =odd[0]
10    back: ldr r6, [r3], #4
11        ands r7, r6, #1
12        beq fwd
13        str r6, [r5], #4
14        b fwd1
15    fwd: str r6, [r4], #4
16    fwd1: subs r1, r1, #1
17    bne back
18
19    exit: b exit
20
21 .data
22 count: .word 0x07
23 array: .word 0x15, 0x35, 0x32, 0x45, 0x10, 0x4f, 0x34,
24 even: .word 0, 0, 0, 0, 0
25 odd: .word 0, 0, 0, 0, 0
```

The 'Registers' panel on the left shows the following register values:

r0	00000038
r1	00000000
r2	00000000
r3	0000004f
r4	00000010
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	0000002c
cpsr	600001d3
spsr	00000000

The screenshot shows the QM-IDE interface with the following tabs open:

- Registers**: Shows memory dump for registers R0-R12, sp, lr, and pc.
- Call stack**
- Trace**
- Breakpoints**
- Watchpoints**
- Symbols**
- Settings**

Number Display Options (Settings):

- Size: Word
- Format: Hexadecimal
- Memory words per row: 4

Messages (Settings):

```
work/asmbCQbrU.s: Assembler messages:  
work/asmbCQbrU.s:22: Warning: zero assumed for missing expression  
Link: arm-unknown-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmbCQbrU.s.elf work/asmbCQbrU.s.o  
Compile succeeded.
```

From this result I have observed.....

Input:

Sl. No.	Memory Location	Operand (Data)
1	00000048	0x07
2	0000004C	0x15
3	00000050	0x35
4	00000054	0x32
5	00000058	0x45
6	0000005C	0x10
7	00000060	0x4f
8	00000064	0x34

Output:

Sl. No.	Memory Location	Operand (Data)
1	0000005C	0x10
2	00000060	0x4f
3	00000064	0x34
4	0000006c	0x32
5	00000080	0x15
6	00000084	0x35
7	00000088	0x45

IV. CONCLUSION

We have successfully achieved the two primary objectives (i.e., Identifying the largest and smallest numbers in an array of size N , and separating even and odd numbers from an array of size N) by implementing a low-level data manipulation and control flow. This provided practical experience in handling arrays and conditional logic within the constraints of ARM architecture.

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

1) The condition codes (flags) of ARM Processor are:-

- i) N (Negative flag): Set if the result of the operation is negative (i.e. MSB=1)
- ii) Z (zero flag): Set if the result of the operation is zero.
- iii) C (carry flag): Set if the operation result is a carry out from the most significant bit (MSB) for addition, or if a borrow in subtraction.
- iv) V (overflow flag): Set if the operation results in an overflow, which occurs when the result of a signed operation exceeds the range that can be represented in the number of bits available.

2) a) B label: No condition codes considered (unconditional)

b) BEQ label: zero flag (Z)

c) BLT label: Negative flag (N) and overflow flag (V) in combination ($N \neq V$)