LAB-4

Aim of the lab: Analyse & Evaluate Different Arithmetic operations and logical operations on two 32 bit data using ARM Processor.

OBJECTIVE - ①: Perform Addition and Substraction of a 32 bit numbers using data processing addressing mode (with immediate data)

① PSEUDOCODE:

```
⇒ Rx ← 0x40      // Load 1st Number
  R1 ← 0x50      // load 2nd Number
  R2 ← R0+0x50   // Add immediate 0x50 to R0, store in
                 //R2
  R3 ← R0 - 0x50 // Substract immediate 0x50 from R0
                 //store it in R3
  R4 ← R0 * R1   // Multiply R0 and R1, store in R4.
```

② ASSEMBLY CODE:

```
.global _start ; Declaring the starting point
_start:
mov R0, #0x40
   ; R0 ← 0x40
   ; Loading immediate value 0x40 in Register R0(1st
                                          Number)
```

```
MOV R1, #0x50
; R1 ← 0x50
; Load immediate value 0x50 in to Register R1 (second No.)

ADDS R2, R0, #0x50
 ; R2 ← R0 + 0x50
;Add immediate value to 0x50 to R0, store result in R2
 ; ADDS also sets condition flags (for carry, overflow zero, negative)

SUBS R3, R0, #0x50
 ; R3 ← R0 - 0x50
 ; Substract immediate 0x50 from R0, store Result in R3
 ; Subs sets condition flags as well

MUL R4, R0, R1
 ; R4 ← R0 x R1
 ; Multiply the values of R0 and R1, store the result in R4.

my_exit : B my_exit
 ; infinite loop to stop program.
```

③ INPUT / OUTPUT ANALYSIS:

(1) MOV R0, #0x40
loads R0 = 0x00000040

(2) MOV R1, #0x50
loads R1 = 0x00000050

} Inputs

(3) ADDS R2, R0, #0x50 ..... OUTPUTS
ADDS 0x40+0x50 and stores in R2 = 0x00000090

(4) SUBS R3, R0, #0x50 ..... OUTPUTS
SUBS 0x40 - 0x50 and store in R3 = 0xFFFFFFF0

(5) MUL R4, R0, R1 ..... OUTPUTS
multiplies 0x40 (*) 0x50 and stores in
R4 = 0x 00001400

④ OBSERVATION TABLE:

① Input:

S.No	memory location	Data
①	R0	0x00000040
②	R1	0x 00000050

① Output:

S.No	memory location	Data
①	R2	0x00000090
②	R3	0xFFFFFFF0
③	R4	0x00001400

OBJECTIVE-② : Perform Addition, Substraction, multiplication of two 32 bit numbers using load / store addressing mode.

① PSEUDOCODE:

R0 ← base address 0x100100000
R1 ← value at [R0], then R0 += 4
R2 ← value at [R0], then R0 += 4
R3 ← R1 + R2, then store at [R0], R0 += 4
R4 ← R1 - R2, then store at [R0], R0 += 4
R5 ← R1 × R2, then store at [R0]

② ASSEMBLY CODE:

```
.global start
_start:
LDR  R0, =0x10100000 ; R0 ← 0x10100000 (load base memory address)
LDR  R1,[R0],#4      ; R1 ← [R0], then R0 += 4 (1st operand)
LDR  R2,[R0],#4      ; R2 ← [R0], then R0 += 4 (2nd operand)

ADDS R3,R1,R2 ; R3 ← R1 + R2 (32 bit addition)
STR  R3,[R0],#4 ; Store R3 at [R0], then R0 += 4

SUBS R4,R1,R2 ; R4 ← R1 - R2 (32-bit substraction)
STR  R4,[R0],#4; Store R4 at [R0], then R0 += 4

MUL  R5,R1,R2; R5 ← R1 * R2 (32 bit multiplication)
STR  R5,[R0] ; Store R5 at current [R0] (offset - none)

my_ext: B my_ext ; infinite loop to end execution
```

(3) INPUT / OUTPUT ANALYSIS :

Input:

(1) mov R0, #0x60
Loads 0x60 in R0 : R0 = 0x00000060

(2) mov R1, #0x50
Loads 0x50 in R1 : R1 = 0x00000050

Outputs:

(3) ADDS R2, R0, #0x50
0x60 + 0x50 = 0xB0 ; R2 = 0x000000B0

(4) SUBS R3, R0, #0x50
0x60 - 0x50 = 0x10 ; R3 = 0x00000010

(5) MUL R4, R0, R1
0x60 * 0x50 = 0x1E00 at R4 : R4 = 0x00001E00

(4) OBSERVATION TABLE:

Input:

S.No	Memory location	Operand (Data)
(1)	R0	0x00000060
(2)	R1	0x00000050

Output

S.No	Memory location	Operand (Data)
(1)	R2	0x000000B0
(2)	R3	0x00000010
(3)	R4	0x00001E00

OBJECTIVE - ③: Perform the logical operations (AND, OR, XOR & NOT) on two 32 bit no. using load/store addressing mode.

① PSUEDOCODE:

⇒ Load Base address into R0
Load 1st 32 bit data from memory in R1
Load 2nd 32 bit data from memory in R2
Perform logical AND : R3 = R1 AND R2.
Store result (R3) back to memory.
Perform logical OR: R4 = R1 OR R2
Store Result : R4.
Perform logical XOR: R5 = R1 XOR R2
Store Result R5.
Perform bitwise NOT on R1 : R6 = Not R1
Store Result R6.

② ASSEMBLY CODE:

```
.global _start
_start:
LDR  R0, =0x10100000   // Load Base memory address : R0 = 0x10100000
LDR  R1, [R0], #4      // Load 1st operand from memory :
                          R1 ← mem[R0] , R0 += 4
LDR , R2, [R0], #4     // Load 2nd operand from memory:
                          R2 ← mem [R0] , R0 += 4
AND  R3, R1, R2        // Logical AND → R3 ← R1 & R2
STR  R3, [R0] #4       // Store result → [R0] ← R3
```

```
ORR  R4, R1, R2   // Logical OR → R4 ← R1 | R2
STR  R4, [R0], #4 // Store result of OR → [R0] ← R4, R0 += 4
EOR  R5, R1, R2   // Logical XOR → R5 ← R1 ^ R2
STR  R5, [R0], #4 // Store result of XOR → [R0] ← R5, R0 += 4
MVN  R6, R1       // Logical NOT → R6 ← ~R1 (bitwise NOT)
STR  R6, [R0]     // Store result of NOT → [R0] ← R6

my_exit : b my_exit // Stops execution.
```

③ INPUT / OUTPUT ANALYSIS:

Input:

(1) R1 = 0x00000080
(2) R2 = 0x00000070

Outputs:

(3) R3 ← R1 AND R2

So, 0x80 AND 0x70

and we get the results as 0x00000000

So, R3 = 0x00000000

[R0] ← R3 then R0 += 4

(4) R4 ← R1 OR R2

So, 0x80 OR 0x70

and we get the results as 0x000000f0

So, R4 = 0x000000f0

So, [R0] ← R4 then R0 += 4

(5) R5 ← R1 xoR R2

So, 0x080 xoR 0x70

gives R5 = 0x000000f0

So, [R0] ← R5 then R0 += 4

(6) R6 ← Not R1

So, Not (0x80) gives R6 = 0xFFFFFF7F

So, [R0] ← R6

(4) OBSERVATION TABLE :

Input :

S.No	Memory location	Operand Data
(1)	0x10100000	0x80
(2)	0x10100004	0x70

Output :

S.No	Memory location	operand (Data)
(1)	0x10100008	0x00000000
(2)	0x1010000C	0x000000f0
(3)	0x10100010	0x000000f0
(4)	0x10100014	0xFFFFFF7F

④ CONCLUSION :

⇒ The above 3 tasks demonstrates key ARM assembly techniques for arithmetic & logical operations using both immediate and load/store addressing modes. Immediate mode offers faster execution for fixed values, ideal for quick calculations whereas load/store mode enables operations on data stored in memory which is essential for handling dynamic real world data in embedded systems. These operations form the foundations for system level programming and efficient designs.

⑤ POST LAB :

(Q1) Give any 5 examples of arithmetic and logical instructions.

Soln) Arithmetic Instructions :

(i) ADD : Add
(ii) ADC : Add with carry
(iii) SUB : Subtract
(iv) SBC : Subtract with Carry
(v) RSB : Reverse Subtract

Logical Instructions :

(i) AND = Logical AND
(ii) EOR = Exclusive OR
(iii) ORR = logical OR
(iv) BIC = Bitwise clear

(Q2) Differentiate b/w LDR and STR instructions.

Soln) The LDR instructions are used to load data from a memory address, into a register; ie it reads from memory whereas STR instructions is used to store data from a register into a memory address ie it writes to memory.

(Q3) Which of the following instructions is not valid:

(a) mov R7. R2

(b) LDR R1, = LABEL

Soln) The instruction that is not valid is mov R7. R2.

Here's, there's a syntax error, the separator b/w registers should be comma not a dot.

Correct Syntax:

mov R7, R2 {move contents from R2 to R7}