Computer Organization and Architecture (EET2211)

LAB IV: Evaluate Different Arithmetic Operations and Logical operations on two 32-bit data 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	Dinanath Dash

Marks:______/10

Remarks:

Teacher's Signature

I. OBJECTIVE:

1. Perform Addition and Subtraction of two 32-bit numbers using data processing addressing mode (with immediate data).
2. Perform Addition, Subtraction, and Multiplication of two 32-bit numbers using load/store addressing mode.
3. Perform the logical operations (AND, OR, XOR, and NOT) on two 32-bit numbers using load/store addressing mode.

II. PRE-LAB

Note: For each objective in prelab describe the following points:

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

For Obj. 1→

a) Pseudocode →

```
i) Load the value 10 into register r0
ii) Load the value 5 into register r1
iii) Add the values in registers r0 & r1, store the result in r2
iv) Subtract the value in register r1 from r0, store the result in r3.
```

b) Code→

```
.global _start      @ Start of the assembly code, global visibility
_start:             @ Label for the start of the program
mov r0, #10         @ Move the immediate value 10 into register r0
mov r1, #5          @ Move the immediate value 5 into register r1
add r2, r0, r1      @ Add the values in registers r0 & r1, store the result in r2
~~add r~~
sub r3, r0, r1      @ Subtract the value in register r1 from r0, store the result in r3
.end                @ End of the assembly code
```

For Obj. 2→

a) Pseudocode →

```
i) Load the value at memory address 0x10100000 into register r1
ii) Load the value at memory address 0x10100004 into register r2
iii) Add the values in registers r1 & r2, store the result ~~into~~ in r3
iv) Store the value in r3 at memory address 0x10100008
v) Subtract the value in r2 from r1, store result in r4.
vi) Store the value in r4 at memory address 0x1010000C
```

vii) Multiply the values in r1 and r2, store the result in r5
viii) Store the value in r5 at memory address 0x10100010

b) Code →

```
.global _start
_start:
    ldr r0, =0x10100000
    ldr r1, [r0], #4
    ldr r2, [r0], #4
    add r3, r1, r2
    str r3, [r0], #4
    sub r4, r1, r2
    str r4, [r0], #4
    mul r5, r1, r2
    str r5, [r0]
    my_exit: b my_exit
.end
```

For Obj. 3 →

a) Pseudocode →

```
i) Load the value at memory address 0x10100000 into register r0
ii) Load the value at the address in r0 into r1, then increment r0 by 4
iii) Load the value at the updated address in r0 into r2, then increment r0 by 4
iv) Perform bitwise AND operation between r1 & r2, store the result in r3
v) Store the value in r3 at the address stored in r0, then increment r0 by 4
vi) Perform bitwise OR operation between r1 & r2, store the result in r4.
vii) Store the value in r4 at the address stored in r0, then increment r0 by 4.
viii) Perform bitwise XOR operation between r1 & r2, store the result in r5.
ix) Store the value r5 at the address stored in r0, then increment r0 by 4.
x) Perform bitwise NOT operation on r1, store the result in r6.
xi) Store the value in r6 at the address stored in r0.
```

b) Code →

```
.global _start
_start:
    ldr r0,=0x101000000
    ldr r1, [r0], #4
    ldr r2, [r0], #4
    and r3, r1, r2
    str r3, [r0], #4
    orr r4, r1, r2
    str r4,[r0],#4
    eor r5,r1,r2
    str r5, [r0], #4
    mvn r6,r1
    str r6, [r0]
    my_exit : b my-exit
.end
```

III. LAB

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

- Screenshots of the Assembly language program (ALP)

For Obj. 1:

```
.global _start
_start:
    mov r0, #10
    mov r1, #5
    add r2, r0, r1
    sub r3, r0, r1
.end
```

For Obj. 2:

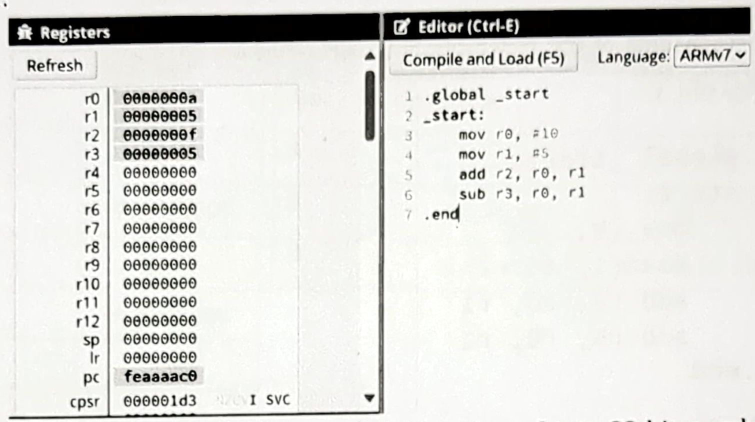
```
.global _start
_start:
    ldr r0, =0x10100000
    ldr r1, [r0], #4
    ldr r2, [r0], #4
    add r3, r1, r2
    str r3, [r0], #4
    sub r4, r1, r2
    str r4, [r0], #4
    mul r5, r1, r2
    str r5, [r0]
    my_exit: b my_exit
.end
```

For Obj. 3:

```
.global _start
_start:
    ldr r0, =0x10100000
    ldr r1, [r0], #4
    ldr r2, [r0], #4
    and r3, r1, r2
    str r3, [r0], #4
    orr r4, r1, r2
    str r4, [r0], #4
    eor r5, r1, r2
    str r5, [r0], #4
    mvn r6, r1
    str r6, [r0]
    my_exit: b my_exit
.end
```

- Observations (with screenshots)

For Obj. 1:



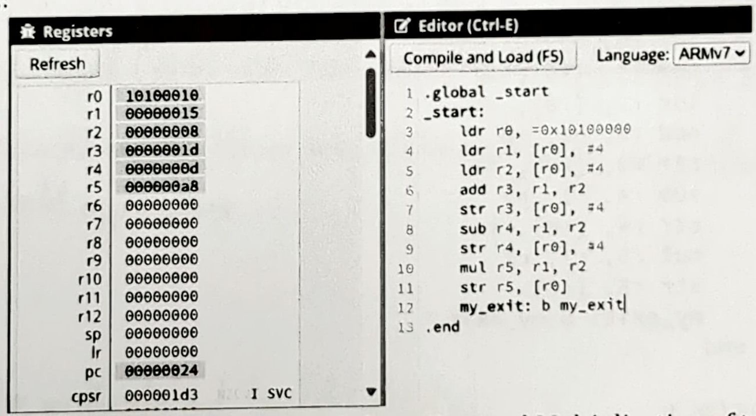
Language: ARMv7

```
r0    0000000a
r1    00000005
r2    0000000f
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    feaaaac0
cpsr  000001d3   I SVC
```

```
.global _start
_start:
        mov r0, #10
        mov r1, #5
        add r2, r0, r1
        sub r3, r0, r1
.end
```

Fig. 1. Execution results of Addition and Subtraction of two 32-bit numbers using data processingaddressing mode (with immediate data).

For Obj. 2:



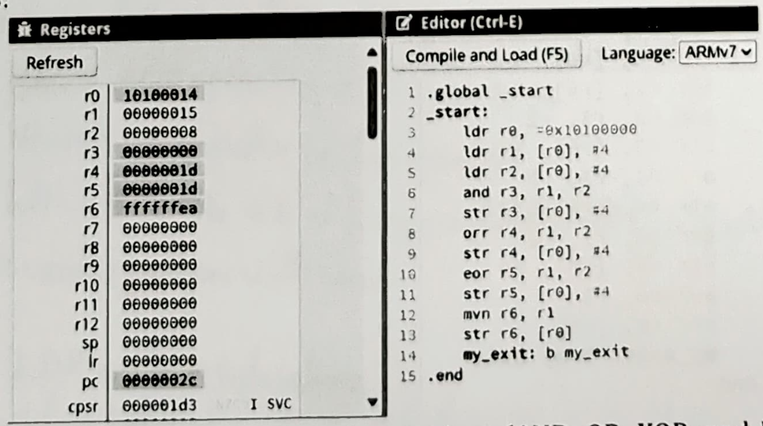
Language: ARMv7

```
r0    10100010
r1    00000015
r2    00000008
r3    0000001d
r4    0000000d
r5    000000a8
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000024
cpsr  000001d3   I SVC
```

```
.global _start
_start:
        ldr r0, =0x10100000
        ldr r1, [r0], #4
        ldr r2, [r0], #4
        add r3, r1, r2
        str r3, [r0], #4
        sub r4, r1, r2
        str r4, [r0], #4
        mul r5, r1, r2
        str r5, [r0]
        my_exit: b my_exit
.end
```

Fig. 2. Execution results of Addition, Subtraction, and Multiplication of two 32-bit numbers using load/store addressing mode.

For Obj. 3:



Language: ARMv7

```
r0    10100014
r1    00000015
r2    00000008
r3    00000000
r4    0000001d
r5    0000001d
r6    ffffffea
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    0000002c
cpsr  000001d3   I SVC
```

```
.global _start
_start:
        ldr r0, =0x10100000
        ldr r1, [r0], #4
        ldr r2, [r0], #4
        and r3, r1, r2
        str r3, [r0], #4
        orr r4, r1, r2
        str r4, [r0], #4
        eor r5, r1, r2
        str r5, [r0], #4
        mvn r6, r1
        str r6, [r0]
        my_exit: b my_exit
.end
```

Fig. 3. Execution results of the logical operations (AND, OR, XOR, and NOT) on two 32-bit numbers using load/store addressing mode.

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

Input:

Sl. No.	Memory Location	Operand (Data)
1	r0, r1	10, 05
2	r0	0x10100000
	r1, r2	15, 08
3	r0	0x10100000
	r1, r2	15, 08

Output:

Sl. No.	Memory Location	Operand(Data)
1	r2, r3	f, 05
2	r3, r4, r5	08, 1d, d
3	r3, r4, r5, r6	00, 1d, 1d, ffffffea

IV. CONCLUSION

The ~~arm~~ ARM processor effectively executed arithmetic (addition, subtraction, multiplication, division) and logical (AND, OR, XOR, logical shift) operations on two 32-bit data. It handled overflow conditions and bitwise manipulations efficiently, demonstrating its versatility and capability in real-world applications.

V) POST LAB →

Q1) Give any examples of five arithmetic and logical instructions.

Ans→ i) ADD → Adds two operands together.

ii) SUB → Subtracts one operand from another.

iii) AND → Performs a bitwise AND operation on two operands.

iv) ORR → Performs a bitwise OR operation on two operands.

v) LSL → Logical shift left → shifts the bits of an operand to the left by a specified number of positions, filling the vacated positions with zeros.

Q2) Differentiate between LDR and STR instruction.

Ans→ The LDR (Load Register) instruction loads data from memory into a register, while the STR (Store Register) instruction stores data from a register into memory. In other words, LDR fetches data from memory into a register, whereas STR writes data from a register into memory.

Q3) Which of the following instructions is not valid

a) MOV R7.R2

b) LDR R1, =LABEL

Ans→ a) MOV R7.R2 is not a valid ARM instruction. The correct syntax for moving data between registers in ARM assembly language is MOV R7, R2. ~~So, the co~~