

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: 4-4		
S. No.	Name	Registration No.	Signature
27	E. JAGADEESWAR PATRO	2241016309	E. Jagadeeswar Patro

Marks: _____ / 10

Remarks:

Teacher's Signature

I. OBJECTIVES:

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.

Objective 1:

→ Pseudocode :

1. Set the value 0x40 into register R₀
2. Set the value 0x50 into R₁
3. Add R₀ and R₁ and store in R₂
4. Subtract R₁ from R₀ and store in R₃
5. Terminate program

→ Assembly Code :

· global _start

- Start :

 mov r0, #0x40

 mov r1, #0x50

 adds r2, r0, r1

 subs r3, r0, r1

 mul r4, r0, r1

my_exit: b my_exit // infinite loop terminates the code

Objective 2:

→ Pseudocode:

1. Load R₀ with 0x10100000 memory location
2. Load value at address [10100000] into R₁
3. Load value at [10100004] into R₂
4. Add R₁ and R₂ and store in [10100008]
5. Subtract R₂ from R₁ and store at [1010000C]
6. Multiply R₁ and R₂ and store at [10100010]
7. Terminate Program.

→ Assembly Code:

.global _Start

_Start:

ldr r0, =0x10100000 // load register

ldr r1, [r0], #4 // address +4

ldr r2, [r0], #4

adds r3, r1, r2

str r3, [r0], #4

subs r4, r1, r2

str r4, [r0], #4

mul r5, r1, r2

str r5, [r0]

my_exit: b my_exit

Objective 3:

→ pseudocode :

1. load r_0 with [10100000] memory location
2. load r_1 from [10100000] and r_2 from [10100004] for input.
3. perform AND operation on r_1 and r_2 and store at [10100008]
4. perform OR on r_1 and r_2 and store at [1010000C]
5. Perform XOR on r_1 and r_2 and store at [10100010]
6. Copy complement of r_1 into r_6 and store at [10100014]
7. Copy complement of r_2 into r_7 and store at [10100018]
8. Terminate program.

→ Assembly Code :

.global _start

_start:

ldr r_0 , =0x10100000

ldr r_1 , [r_0], #4

ldr r_2 , [r_0], #4

and r_3 , r_1 , r_2 // AND operation

str r_3 , [r_0], #4

orr r_4 , r_1 , r_2 // OR operation

str r_4 , [r_0], #4

eor r_5 , r_1 , r_2 // XOR operation

str r_5 , [r_0], #4

mvn r_6 , r_1 // NOT of r_1

str r_6 , [r_0], #4

mvn r_7 , r_2 // NOT of r_2

str r_7 , [r_0]

my_exit: b my_exit

III. LAB

Objective 1:

stopped

Registers

r0	00000040
r1	00000050
r2	00000090
r3	fffffff0
r4	00001400
r5	00000000
r6	00000000
r7	00000000

Registers Call stack Breakpoints Watchpoints Symbols Counters Settings Number Display Options

size: Word

Messages

```
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmF3MMVb.s.elf work/asmF3MMVb.s.o
compile succeeded.
```

Editor (Ctrl-E) Compile and Load (F5) Language: ARMv7 untitled (3).s

```

1 .global _start
2 _start:
3     mov r0, #0x40
4     mov r1, #0x50
5     adds r2, r0, r1
6     subs r3, r0, r1
7     mul r4, r0, r1
8 my_exit: b my_exit
9 // E. Jagadeeswar Patro, 2241016309

```

Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)

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

Input:

Sl. No.	Memory Location	Operand (Data)
1	γ_0	0x40
2	γ_1	0x50

Output:

Sl. No.	Memory Location	Operand (Data)
1	γ_2	0x90
2	γ_3	0xfffffff0
3	γ_4	0x00001400

Objective 2:

Stopped Step Into Step Over Step Out Continue Stop Restart Reload File

Registers

r0	10100010
r1	00000040
r2	00000050
r3	00000090
r4	ffffffffff
r5	00001400
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000

Registers Call stack Trace
Breakpoints Watchpoints

Disassembly

```
Compile And Load (F5) Language: Assembly Working
1: global _start
2: _start:
3:     ldr r0, =0x10000040 //load register
4:     ldr r1, [r0], #0 //address of
5:     add r2, r1, r3
6:     str r3, [r0], #0 //store r3 at [r0] address
7:     sub r4, r1, r2
8:     str r4, [r0], #0
9:     mul r5, r1, r2
10:    str r5, [r0]
11:    my_exit: b my_exit
12: // E: Jagadeeswar Patro, 2345616399
13: // lab4 obj 2
14: //input value at memory
```

Stopped Step Into Step Over Step Out Continue Stop Restart Reload File Help

Registers

r0	10100010
r1	00000040
r2	00000050
r3	00000090
r4	ffffffffff
r5	00001400
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000

Registers Call stack Trace
Breakpoints Watchpoints
Symbols Counters

Memory (Ctrl-M)

Go to address, label, or register: 10100000

Address	Memory contents and ASCII
100ffff0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffff00	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffffa0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffffb0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffffc0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffffd0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100ffffe0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
100fffff0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100000	00000040 00000050 00000090 ffffff00 Good Pass.....
10100010	00001400 aaaaaaaaaaaaaaaaaaaaaaaa.....
10100020	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100030	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100040	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100050	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100060	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100070	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100080	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100090	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
101000a0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
101000b0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
101000c0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
101000d0	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....

Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)

Messages

Code and data loaded from ELF executable into memory. Total size is 48 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -afpu=neon-float -gdb=tui -o work/compilation.o
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/aangandar.aelf work/compilation.o
Compile succeeded.

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

Input:

Sl. No.	Memory Location	Operand (Data)
1	10100000	0x40
2	10100004	0x50

Output:

Sl. No.	Memory Location	Operand (Data)
1	10100008	c40
2	1010000c	0xffffffff0
3	10100010	0x00001000

Objective 3:

Registers

```

r0 1010001c
r1 00000040
r2 00000050
r3 00000040
r4 00000050
r5 00000010
r6 ffffffbf
r7 ffffffaf
r8 00000000
r9 00000000
r10 00000000
r11 00000000
r12 00000000
sp 00000000

```

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 **untitled.s [changed since save] [changed since compile]**

```

1 .global _start
2 _start:
3     ldr r0, =#0x10100000 //load register
4     ldr r1, [r0], #4      //address +4
5     ldr r2, [r0], #4
6     and r3, r1, r2
7     str r3, [r0], #4    // r3 = r1 and r2
8     orr r4, r1, r2
9     str r4, [r0], #4
10    eor r5, r1, r2
11    str r5, [r0], #4
12    mvn r6, r1
13    str r6, [r0], #4    //not
14    mvn r7, r2
15    str r7, [r0], #4
16 my_exit: b my_exit
17 //E. Jagadeeswar Patro, 2241016309
18 //lab4 obj 3
19 //input value at memory window

```

Registers

```

r0 1010001c
r1 00000040
r2 00000050
r3 00000040
r4 00000050
r5 00000010
r6 ffffffbf
r7 ffffffaf
r8 00000000
r9 00000000
r10 00000000
r11 00000000
r12 00000000
sp 00000000

```

Memory (Ctrl-M)

Go to address, label, or register: 10100000

Address	Memory contents and ASCII
10000000	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000001	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000002	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000003	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000004	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000005	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000006	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000007	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000008	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10000009	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1000000a	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1000000b	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1000000c	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1000000d	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100000	00000000 00000000 00000000 00000000 @.... P.... @.... P....
10100001	00000001 ffffffbf ffffffaf aaaaaaaaa.....
10100002	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100003	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100004	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100005	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100006	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100007	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100008	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
10100009	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1010000a	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1010000b	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1010000c	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....
1010000d	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....

Editor (Ctrl-E) </> Disassembly (Ctrl-D) Memory (Ctrl-M)

Code and data loaded from ELF executable into memory. Total size is 64 bytes.

Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/armPwXnZq.s.o work/armPwXnZq.s

Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/armPwXnZq.self work/armPwXnZq.s.o

Compile succeeded.

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

Input:

Sl. No.	Memory Location	Operand (Data)
1	10100000	0x40
2	10100004	0x50

Output:

Sl. No.	Memory Location	Operand (Data)
1	10100008	0x40
2	1010000C	0x50
3	10100010	0x10
4	10100014	0xffffffffbf
5	10100018	0xffffffffaf

CONCLUSION

On successful completion of the three objectives, marks a significant achievement in demonstrating proficiency in ARM assembly language programming. Performing arithmetic operations and logical operations is successfully done using both immediate data addressing mode and load/store addressing mode.

POST LAB

Give any examples of five arithmetic and logical instructions.

Differentiate between LDR and STR instruction

Which of the following instructions is not valid

- a) MOVR7.R2
- b) LDR R1, =LABEL

Answers:

Any five examples are:-

- a) ADD R1, R2, R3 : Adds values in R2 & R3 and stores in R1
- b) SUB R1, R2, R3 : Performs $R_2 - R_3$ and stores in R1
- c) MUL R1, R2, R3 : Performe $R_2 \times R_3$ and stores in R1
- d) AND R1, R2, R3 : AND between R2 and R3 is stored in R1
- e) ORR R1, R2, R3 : OR between R2 and R3 is stored in R1

LDR (Load)

Used to load data from memory into a register

Reads the data from the memory location specified by source address and loads into destination register.

LDR destination, [source]

STR (store)

→ Used to store data from a register into memory.

→ Writes the data from the source register into the memory location specified by the destination address.

Syntax: STR source, [destination]

3) a) MOV R7, R2

It is incorrect as the instruction should be written

as:

MOV R7, R2

in ARM7 assembly language.