Course Code	**:**	18ECC313J	**Course Title**	**:**	EMBEDDED HARDWARE OPERATING SYSTEMS
Reg. No.	**:**	RA1911043010035	**Name**	**:**	SOUMITRA CHATTERJEE
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**	15.02.2022 / 16.02.2022	**Date of Submission**	**:**	10.03.2022
Name of the Lab Instructor:			Dr.K.Vadivukkarasi		

Title of the Experiment

5	**ARM DATA TRANSFER, ARITHMETIC, AND LOGICAL OPERATIONS USING KEIL UV4**

5.1. Aim(s) / Objective(s) / Purpose

The purpose of this experiment is to perform ARM data transfer, arithmetic, and logical instructions using assembly language programming in Keil UV4 software.

5.2 Introduction / Background

The purpose of this experiment is to learn data transfer, arithmetic, logical, one's and two's complement, and sum of series via the registers, instruction sets, by using MOV, ADD, SUB, SUBS, MUL, MLA, LDR, STR, ORR and etc. Perform all the instruction operations and store the result in appropriate destination.

5.3 Materials / Equipment

1. Andrew Sloss, Dominic Symes, and Chris Wright. ARM system developer's guide: designing and optimizing system software. Elsevier, 2004.
2. Mazidi, M., Naimi, S., Naimi, S. and Mazidi, J., ARM Assembly Language Programming & Architecture. y Pearson Education, Inc., 2013.

5.4 Software Requirement:

KEIL UV4 with supported device packages installed.

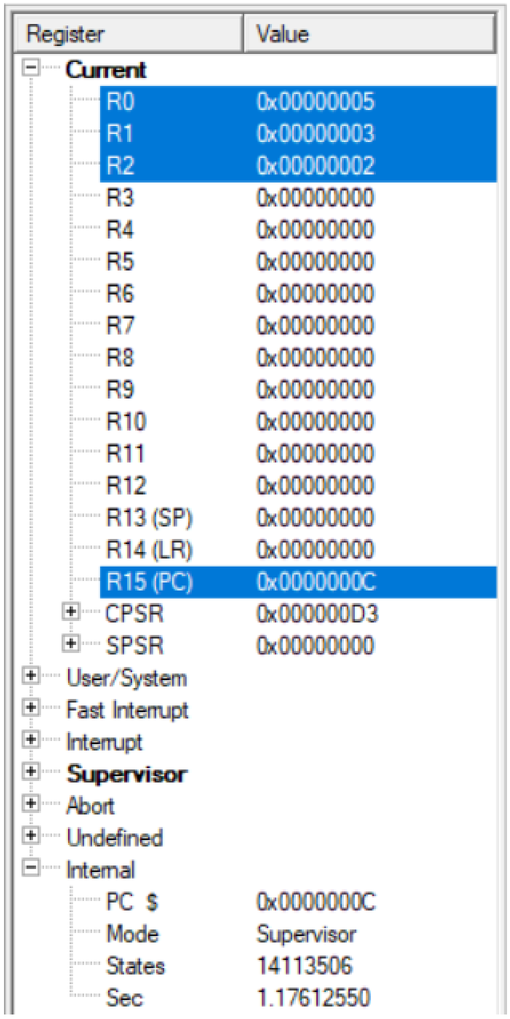
5.5 Procedure

i) Enter the PROGRAM
ii) Execute the program
iii) Check for the result in destination.

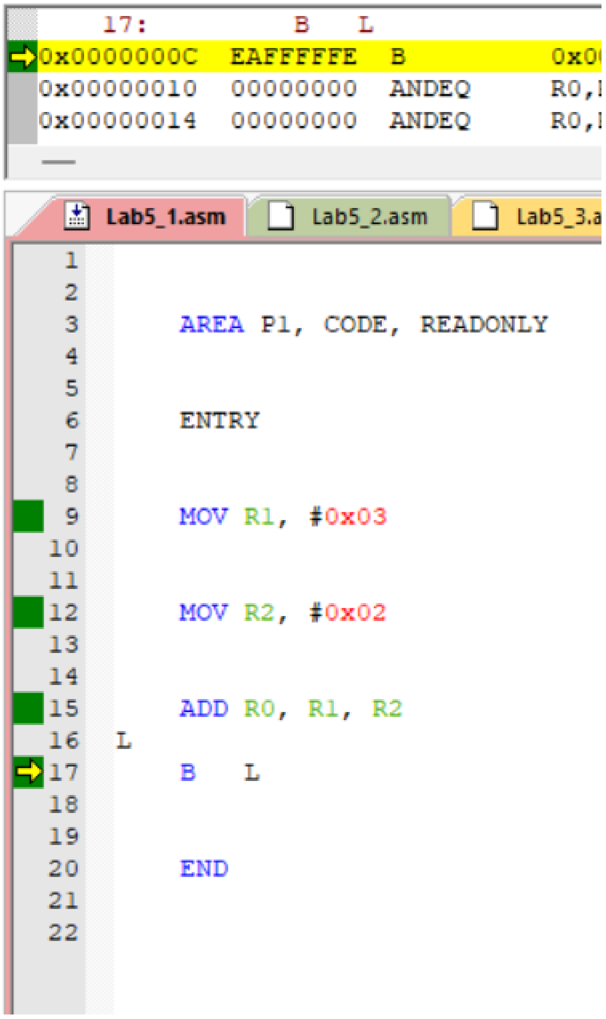
5.6 PROGRAMS:

Program 1: ADDITION OPERATION USING ARM ALP

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
		AREA P1, CODE, READONLY		
		ENTRY		
0x00000000		**MOV R1, #0x03**	E3A01002	Move 03 to R1
0X00000004		**MOV R2, #0x02**	E3A02003	Move 02 to R2
0X00000008		**ADD R0, R1, R2**	E0810002	Add R1&R2 then store result in R0
0X0000000C	**L**	**B L**	EAFFFFFE	Branch and link
		END		Halt the program



Register	Value
Current	
R0	0x00000005
R1	0x00000003
R2	0x00000002
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000000C
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x0000000C
Mode	Supervisor
States	14113506
Sec	1.17612550

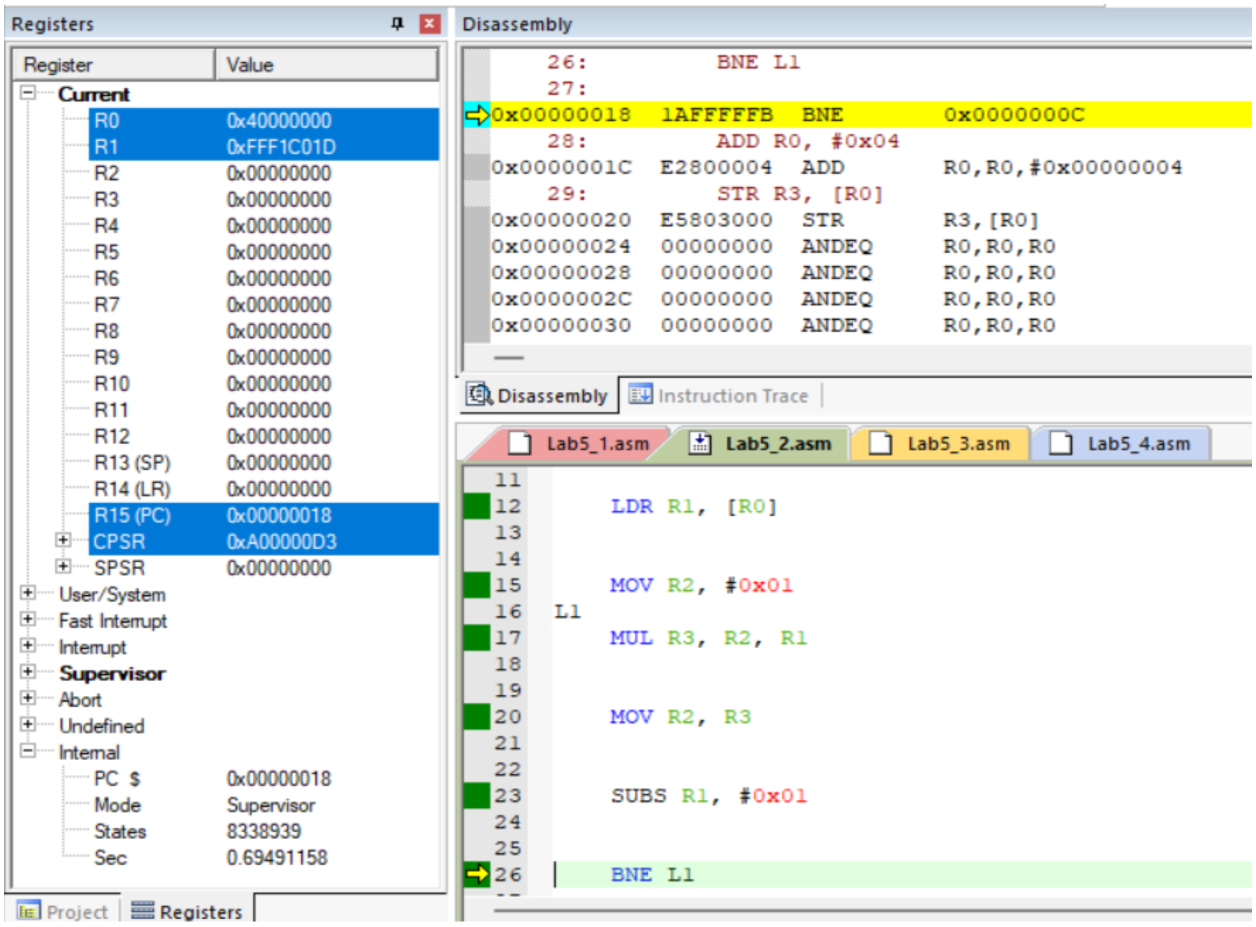


```
    17:            B    L
0x0000000C  EAFFFFFE  B         0x0
0x00000010  00000000  ANDEQ     R0,
0x00000014  00000000  ANDEQ     R0,
```

```

    AREA P1, CODE, READONLY

    ENTRY

    MOV R1, #0x03

    MOV R2, #0x02

    ADD R0, R1, R2
L
    B   L

    END
```

Program 2: ARITHMETIC OPERATIONS USING ARM ALP

ADDRESS	LABE L	MNEMONICS	OPCODE	COMMENTS
		AREA P3, CODE, READONLY		
		ENTRY		
0x00000000		**MOV R0, #0x40000000**	E3A00101	Load the value to R0
0x0000004		**LDR R1, [R0]**	E5901000	Load R0 to R1
0x0000008		**MOV R2, #0x01**	E3A02001	Move 0x01 toR2
0x000000C	**L1**	**MUL R3, R2, R1**	E0030192	Multiply R1&R2.send to R3
0x0000010		**MOV R2, R3**	E1A02003	Move value of R3 to R2
0x0000014		**SUBS R1, #0x01**	E2511001	subtract 01 then send to R1
0x0000018		**BNE L1**	1AFFFFFB	Branch if Not Equal rep L1
0x000001C		**ADD R0, #0x04**	E2800004	Add 0x04 then send to R0
0x0000020		**STR R3, [R0]**	E5803000	Store the value of R0 in R3
		END		Halt the program


Registers

Register	Value
Current	
R0	0x40000000
R1	0xFFF1C01D
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0xA00000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000018
Mode	Supervisor
States	8338939
Sec	0.69491158

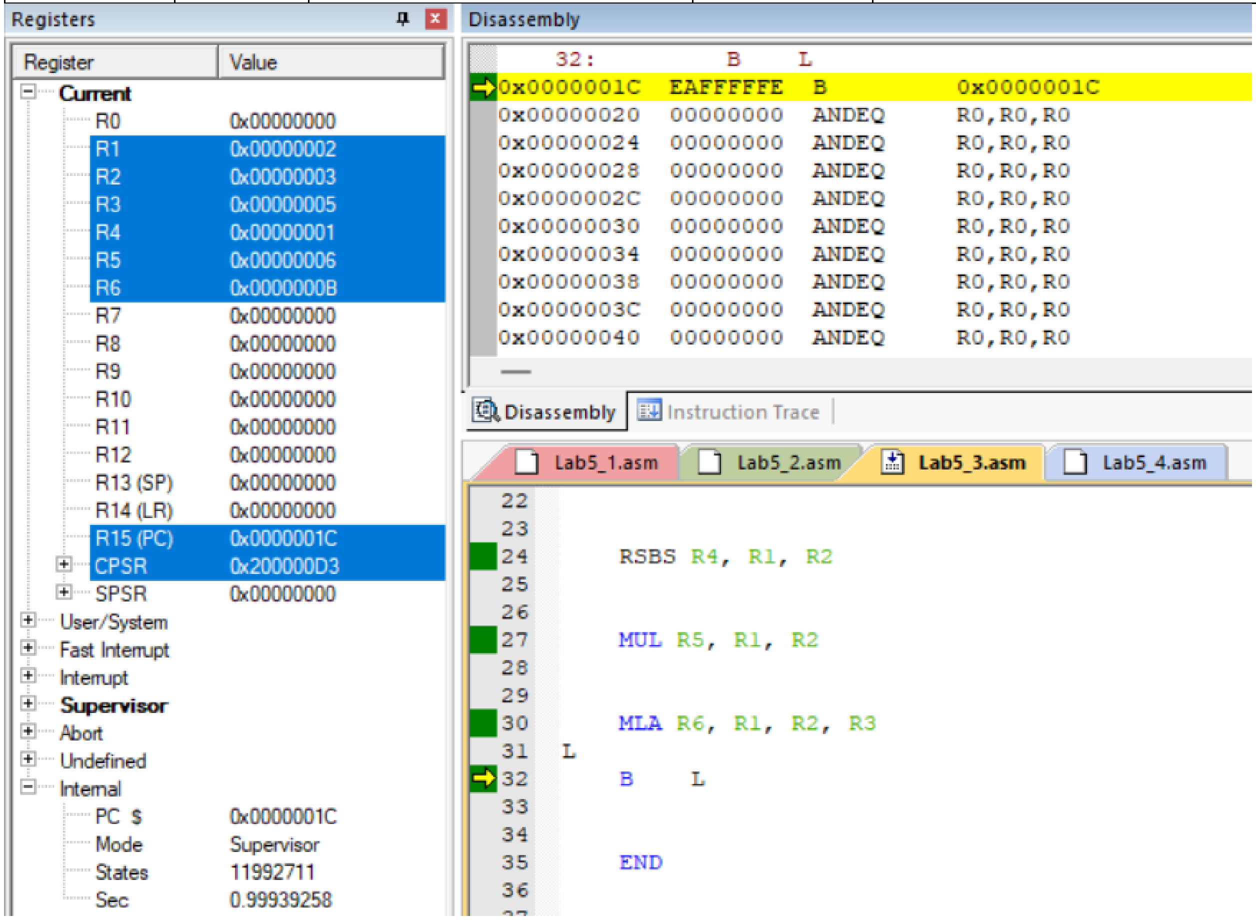
Disassembly

```
    26:          BNE  L1
    27:
0x00000018  1AFFFFFB  BNE      0x0000000C
    28:          ADD R0, #0x04
0x0000001C  E2800004  ADD      R0,R0,#0x00000004
    29:          STR R3, [R0]
0x00000020  E5803000  STR      R3,[R0]
0x00000024  00000000  ANDEQ    R0,R0,R0
0x00000028  00000000  ANDEQ    R0,R0,R0
0x0000002C  00000000  ANDEQ    R0,R0,R0
0x00000030  00000000  ANDEQ    R0,R0,R0
```

```

    LDR R1, [R0]


    MOV R2, #0x01
L1
    MUL R3, R2, R1


    MOV R2, R3


    SUBS R1, #0x01


    BNE L1
```

Program 3: ARITHMETIC OPERATIONS USING ARM ALP

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
		AREA P7, CODE, READONLY		
		ENTRY		
0x0000000		**MOV R1, #02**	E3A01002	Move 02 to R1
0x0000004		**MOV R2, #02**	E3A02002	Move 02 to R2
0x0000008		**ADD R3, R1, R2**	E0813002	Add R1&R2 store result in R3
0x000000C		**SUB R4, R1, R2**	E0414002	Subtract R1&R2 store result in R4
0x0000010		**RSBS R4, R1, R2**	E0714002	Reverse Subtract without carry
0x0000014		**MUL R5, R1, R2**	E0050291	Multiply R1&R2 store result in R5
0x0000018		**MLA R6, R1, R2, R3**	E0263291	Multiply-Accumulate store res in R6
0x000001C	**L**	**B L**	EAFFFFFE	Branch and link
		END		Halt the program



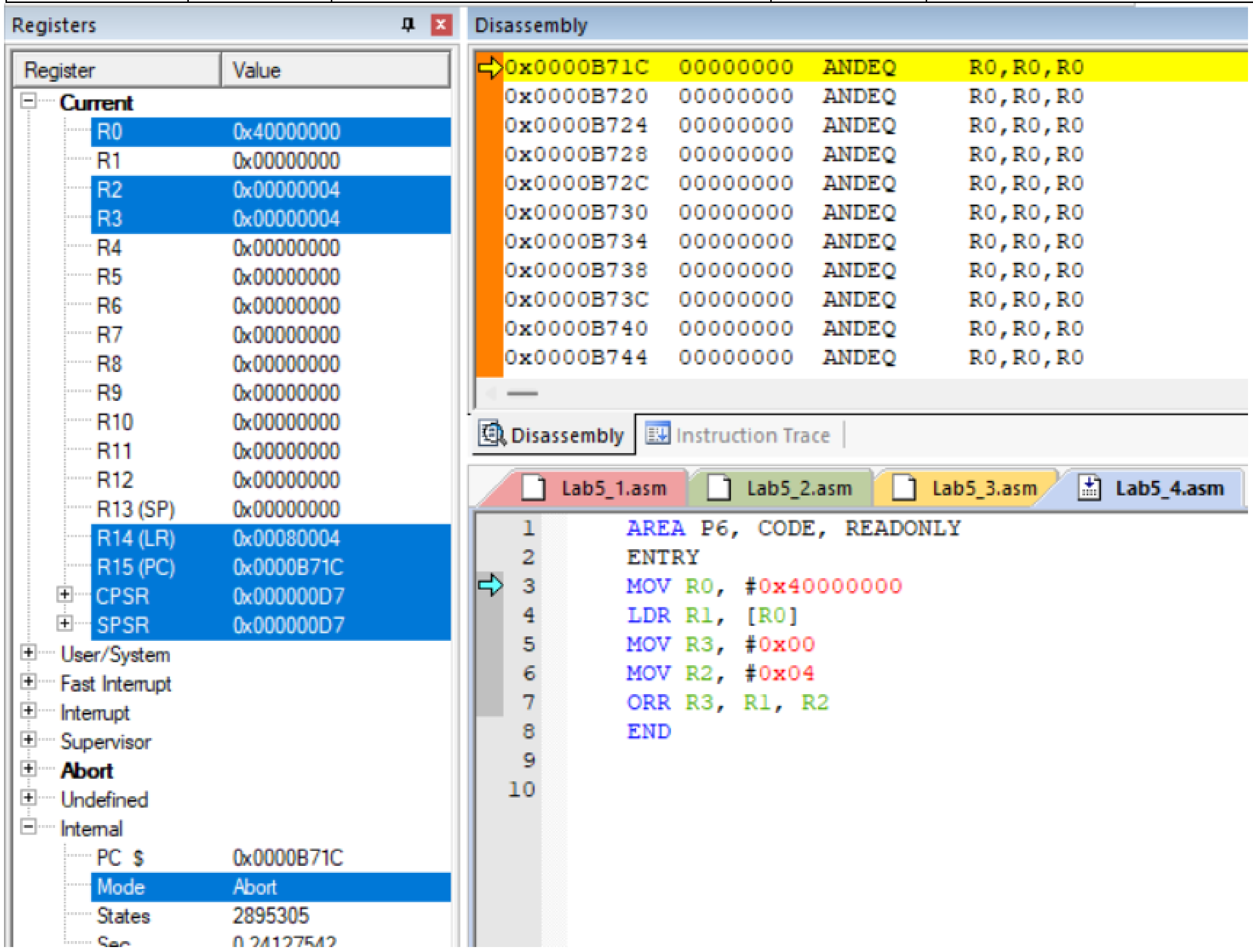
Register	Value
Current	
R0	0x00000000
R1	0x00000002
R2	0x00000003
R3	0x00000005
R4	0x00000001
R5	0x00000006
R6	0x0000000B
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001C
CPSR	0x200000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x0000001C
Mode	Supervisor
States	11992711
Sec	0.99939258

```
    32:         B     L
0x0000001C  EAFFFFFE  B         0x0000001C
0x00000020  00000000  ANDEQ     R0,R0,R0
0x00000024  00000000  ANDEQ     R0,R0,R0
0x00000028  00000000  ANDEQ     R0,R0,R0
0x0000002C  00000000  ANDEQ     R0,R0,R0
0x00000030  00000000  ANDEQ     R0,R0,R0
0x00000034  00000000  ANDEQ     R0,R0,R0
0x00000038  00000000  ANDEQ     R0,R0,R0
0x0000003C  00000000  ANDEQ     R0,R0,R0
0x00000040  00000000  ANDEQ     R0,R0,R0
```

```
        RSBS R4, R1, R2


        MUL R5, R1, R2


        MLA R6, R1, R2, R3
L
        B    L


        END
```

Program 4: LOGICAL OR OPERATION USING ARM ALP

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
		AREA P6, CODE, READONLY		
		ENTRY		
0x00000000		**MOV R0, #0x40000000**	E3A00101	Load the value to R0
0x00000004		**LDR R1, [R0]**	E5901000	Load R0 to R1
0x00000008		**MOV R3, #0x00**	E3A03000	Move 00 to R3
0x0000000C		**MOV R2, #0x04**	E3A02004	Move 04 to R2
0x00000010		**ORR R3, R1, R2**	E3A02004	R2 = R1 OR R2
		END		Halt the program



Registers

Register	Value
Current	
R0	0x40000000
R1	0x00000000
R2	0x00000004
R3	0x00000004
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00080004
R15 (PC)	0x0000B71C
CPSR	0x000000D7
SPSR	0x000000D7
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x0000B71C
Mode	Abort
States	2895305
Sec	0.24127542

Disassembly

```
0x0000B71C  00000000  ANDEQ    R0,R0,R0
0x0000B720  00000000  ANDEQ    R0,R0,R0
0x0000B724  00000000  ANDEQ    R0,R0,R0
0x0000B728  00000000  ANDEQ    R0,R0,R0
0x0000B72C  00000000  ANDEQ    R0,R0,R0
0x0000B730  00000000  ANDEQ    R0,R0,R0
0x0000B734  00000000  ANDEQ    R0,R0,R0
0x0000B738  00000000  ANDEQ    R0,R0,R0
0x0000B73C  00000000  ANDEQ    R0,R0,R0
0x0000B740  00000000  ANDEQ    R0,R0,R0
0x0000B744  00000000  ANDEQ    R0,R0,R0
```

```
    AREA P6, CODE, READONLY
    ENTRY
    MOV R0, #0x40000000
    LDR R1, [R0]
    MOV R3, #0x00
    MOV R2, #0x04
    ORR R3, R1, R2
    END
```

PRE-LAB QUESTIONS:

1. **Explain the arithmetic and data processing instructions of ARM processor.**

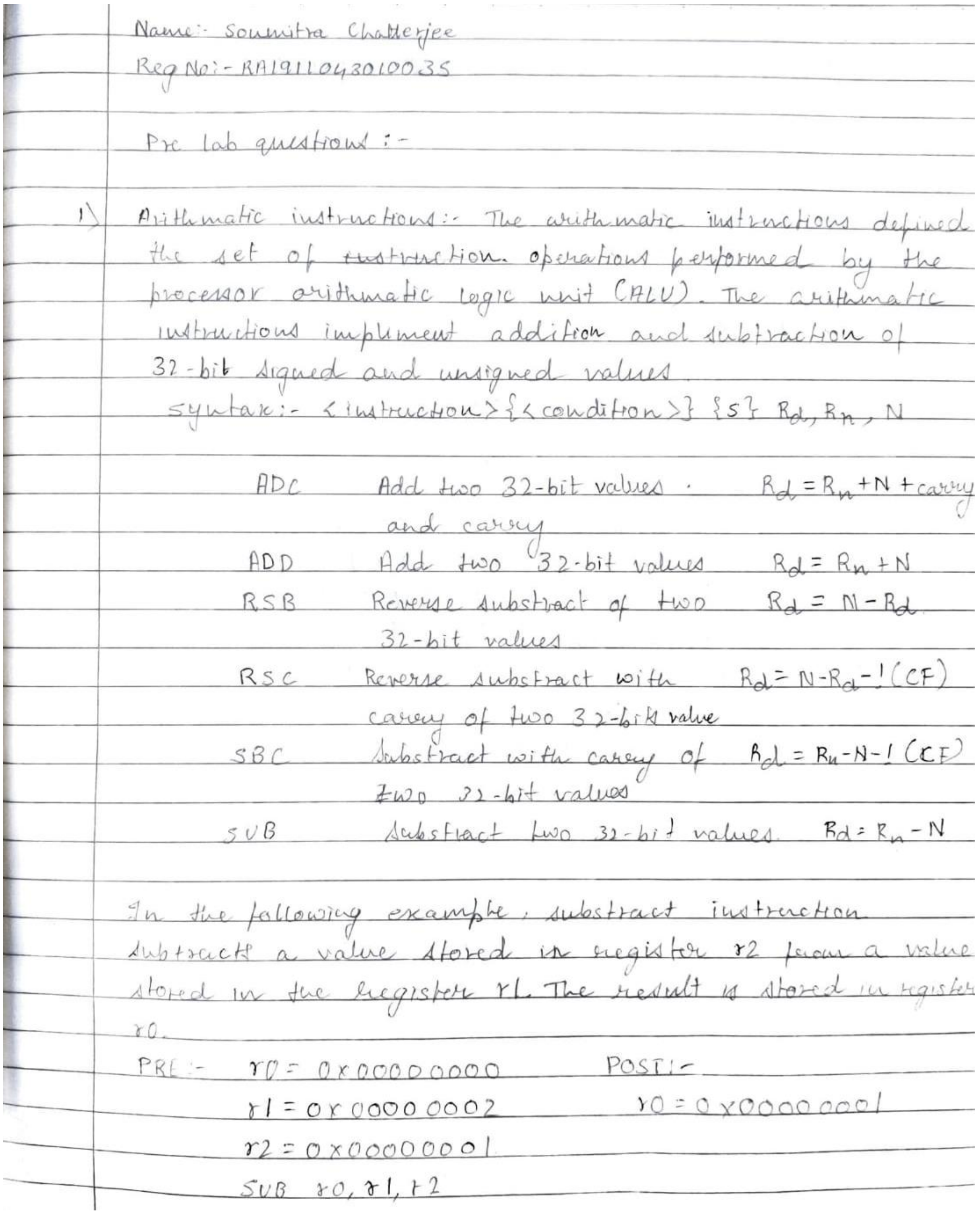
2.Explain the single register transfer and multiple register transfer instructions of ARM processor.

POST-LAB QUESTIONS:

1. **Explain the procedure to run and simulate the ARM code in KEIL IDE.**

2. Write short note on branch instructions of ARM processor.

PRE-LAB QUESTIONS:



Name:- Soumitra Chatterjee

Reg No:- RA1911043010035

Pre lab questions :-

1) Arithmatic instructions:- The arithmatic instructions defined the set of ~~instruction~~ operations performed by the processor arithmatic logic unit (ALU). The arithmatic instructions impliment addition and subtraction of 32-bit signed and unsigned values

syntax:- <instruction>{<condition>} {S} R_d, R_n, N

ADC	Add two 32-bit values and carry	$R_d = R_n + N + \text{carry}$
ADD	Add two 32-bit values	$R_d = R_n + N$
RSB	Reverse substract of two 32-bit values	$R_d = N - R_d$
RSC	Reverse substract with carry of two 32-bit value	$R_d = N - R_d - !(CF)$
SBC	Substract with carry of two 32-bit values	$R_d = R_n - N - !(CF)$
SUB	Substract two 32-bit values	$R_d = R_n - N$

In the following example, substract instruction subtracts a value stored in register r2 from a value stored in the register r1. The result is stored in register r0.

PRE:- r0 = 0x00000000 POST:-
r1 = 0x00000002 r0 = 0x00000001
r2 = 0x00000001
SUB r0, r1, r2

In the following example, the reverse subtract instruction (RSB) subtract r1 from the constant value #0, writing the result in r0.

PRE :-

r0 = 0x00000000

r1 = 0x00000077

RSB r0, r1, #0;

R_d = 0x 0-r1

POST

r0 = -r1 = 0x ffffff89

Data Processing:-

Each ARM instruction is encoded into a 32-bit word. Access to memory is provided only by load and store instructions. ARM data-processing instructions operate on data and produce new value. They are not like the branch instructions operate on data that control the operation of the processor and sequencing of instructions. The data processing instructions manipulate data within registers. They are move instructions, arithmatic instructions, logical instruction, comparison instruction and multiply instructions. Most data processing instructions can processes one of their operands using the barrel shifter. If S is suffixed on a data processing instruction, then it updates the flags in the cpsr. Data processing instructions are processed within the arithmatic and logic unit (ALU). A unique and powerful feature of the

ARM processor is the ability to shift the 32-bit binary pattern in the one of the source registers left or right by a specific number of positions before it enters the ALU. This shift increases the power and flexibility of many data processing operations

For examples we apply a logical shift left (LSL) to register Rm before moving it to the destination register.

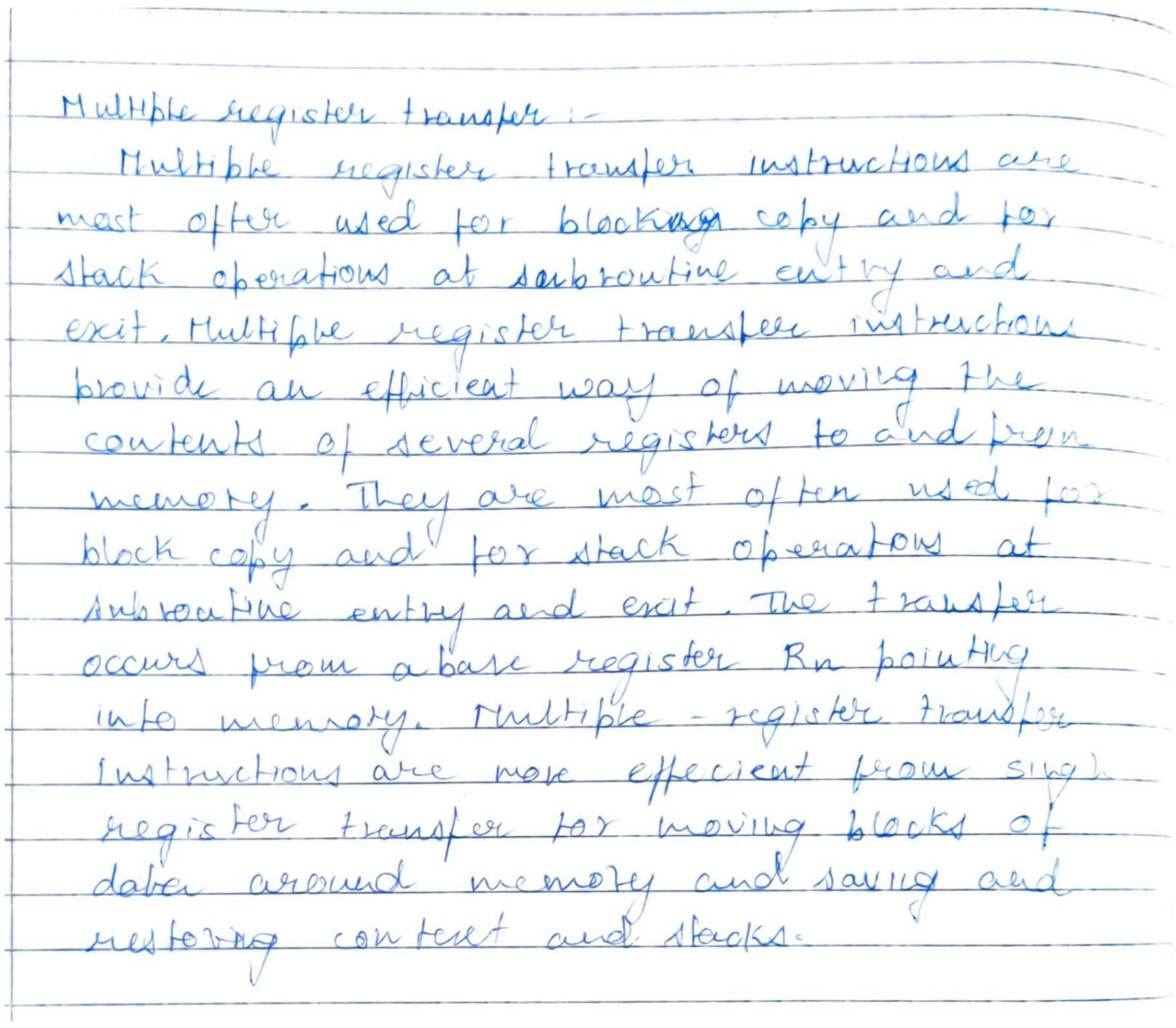
PRE	POST
r5=5	r5 = 5
r7=8	r7=5

MOV r7, r5

The above example shift logical left r5=5 (00000101 in binary) by two bits and then r7=20 (00010100 in binary)

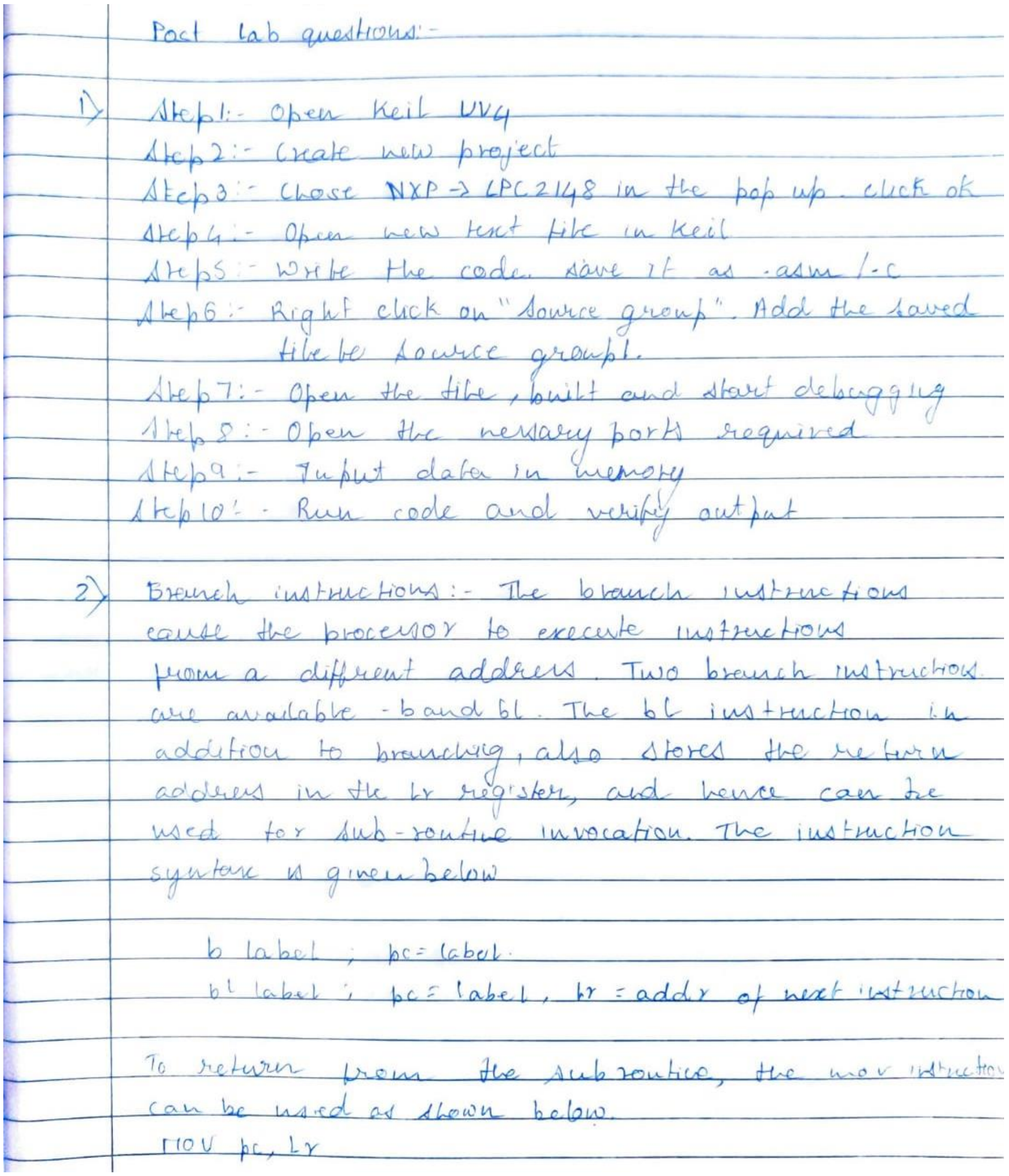
2) Load/store single register:-
The single register load/store instructions allow moving data between memory and an FP/NEON register. These instructions can load/store a byte, half-word, word (single precision), double-word (double precision), or a quad-word. The following instructions are used to load or store a single FP/NEON register.

- ldr: load FP/NEON register and
- str: store FP/NEON register


Multiple register transfer :-

Multiple register transfer instructions are most often used for ~~blocking~~ block copy and for stack operations at subroutine entry and exit. Multiple register transfer instructions provide an efficient way of moving the contents of several registers to and from memory. They are most often used for block copy and for stack operations at subroutine entry and exit. The transfer occurs from a base register Rn pointing into memory. Multiple - register transfer instructions are more effecient from single register transfer for moving blocks of data around memory and saving and restoring context and stacks.

POST-LAB QUESTIONS:



Post lab questions:-

1) Step1:- Open Keil UV4
Step2:- Create new project
Step3:- Chose NXP -> LPC2148 in the pop up. click ok
Step4:- Open new text file in Keil
Step5:- Write the code. save it as .asm /.c
Step6:- Right click on "Source group". Add the saved file to source group1.
Step7:- Open the file, built and start debugging
Step8:- Open the nessary ports required
Step9:- Input data in memory
Step10:- Run code and verify output

2) Branch instructions:- The branch instructions cause the processor to execute instructions from a diffrent address. Two branch instructions are available - b and bl. The bl instruction in addition to branching, also stores the return address in the lr register, and hence can be used for sub-routine invocation. The instruction syntax is given below

```
b label ; pc = label.
bl label ; pc = label, lr = addr of next instruction
```

To return from the subroutine, the mov instruction can be used as shown below.

```
MOV pc, lr
```

Result:

Thus, the ARM programming for data transfer, arithmetic and logical operations for a given input values number is coded and verified using Keil UV4 software.