LABSHEET 5: DATA PROCESSING INSTRUCTIONS IN ARM

Name: Vinayak V Thayil **Roll Number:AM.EN.U4CSE21161**

The data processing instructions manipulate data within registers. They are classified into:

(i) Move instructions
(ii) Arithmetic instructions
(iii) Logical instructions
(iv) Comparison instructions
(v) Multiply instructions.

Move instructions

Copies N into destination register R_d where N is a register or immediate value.

MOV	Move a 32-bit value into a register	$Rd = N$
MVN	move the NOT of the 32-bit value into a register	$Rd = \sim N$

Arithmetic instructions

The arithmetic instructions implement addition and subtraction of 32-bit signed and unsigned values.

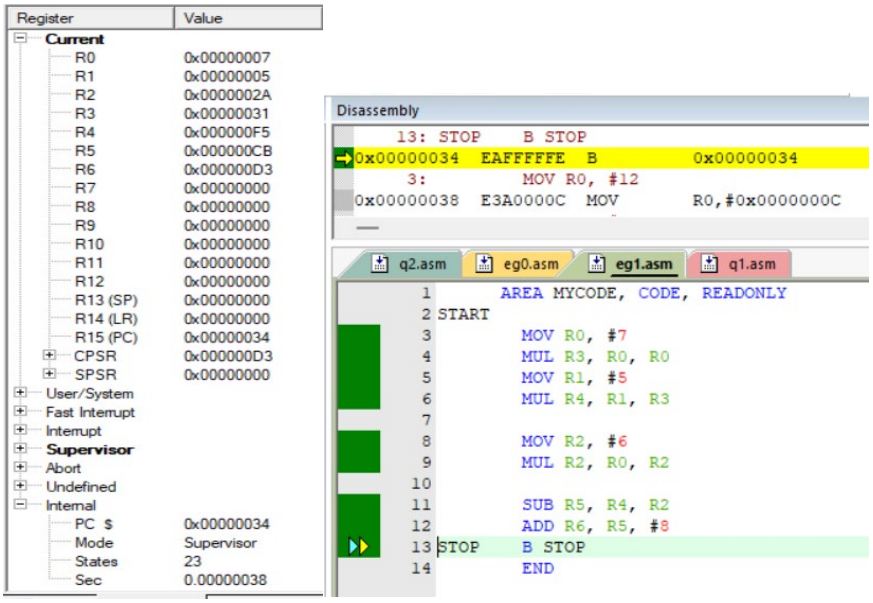
```
Syntax: <instruction>{<cond>}{S} Rd, Rn, N
```

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

Example 1: Perform **$5x^2$ - 6x + 8** using data processing instructions in ARM. Assume that the input value **x** is **7** and is loaded into register R0. Move the constants in the expression to appropriate registers and store the final result in register R6. Attach the screenshot of final status of PC and relevant registers.

LAB4_ARITH.asm

```
1        AREA MYCODE, CODE, READONLY
2  START
3          MOV R0, #7  ; x = 7
4          MUL R3, R0, R0  ; R3 = x^2
5          MOV R1, #5
6          MUL R4, R1, R3
7
8          MOV R2, #6
9          MUL R2, R0, R2  ; R2 = 6x
10
11         SUB R5, R4, R2  ; R5 = 5x^2 - 6x
12         ADD R6, R5, #8  ; R6 = 5x^2 - 6x + 8
13 STOP    B STOP
14         END
```



Logical instructions

Logical instructions perform bitwise logical operations on the two source registers.

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

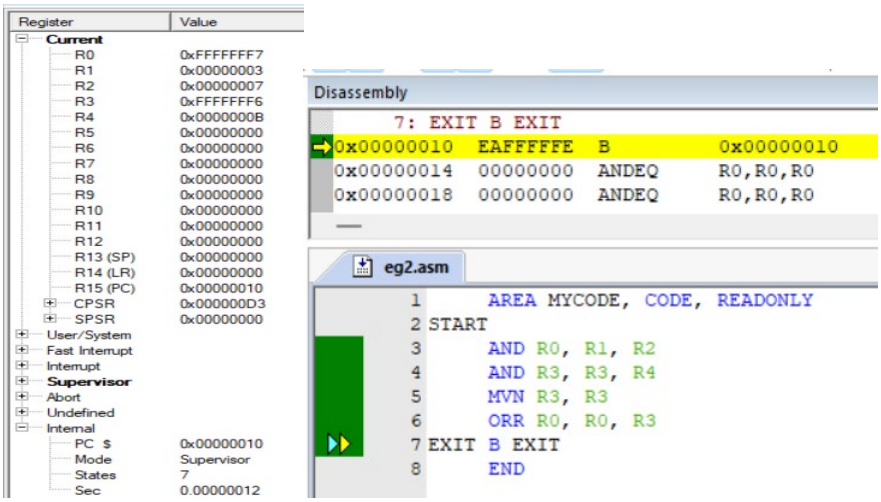
AND	logical bitwise AND of two 32-bit values	$Rd = Rn \& N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn \mid N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \wedge N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$

Example 2: Compute bitwise calculation for the following Boolean expression. Assume that A, B, C, D are in R1, R2, R3, R4, respectively. Load the values A = 3, B = 7, C = 9 and D = 11 in the respective registers. Attach the screenshot of final status of PC and relevant registers.

$$F = A \cdot B + \overline{C \cdot D}.$$

LAB4_LOGIC.asm

```
1       AREA MYCODE, CODE, READONLY
2  START
3       AND R0,R1,R2 ;R0 = A.B
4       AND R3,R3,R4 ;R3 = C.D
5       MVN R3,R3    ;R3 = (C.D)'
6       ORR R0,R0,R3 ;R0 = A.B + (C.D)'
7  EXIT B EXIT
8       END
```



Comparison instructions

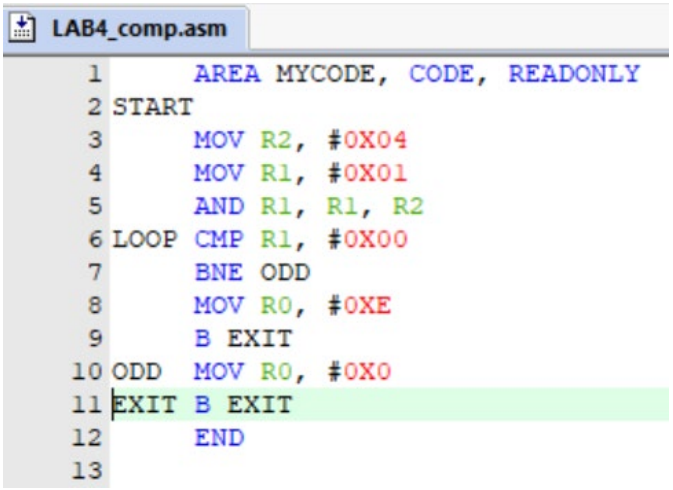
The comparison instructions are used to compare or test a register with a 32-bit value. They update the CPSR flag bits according to the result, but do not affect other registers. For these instructions no needs to apply the S suffix for update the flags.

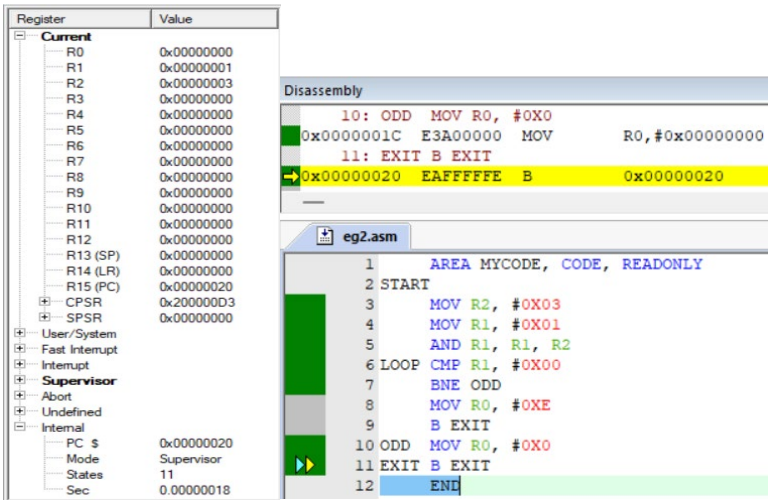
Syntax: <instruction>{<cond>} Rn, N

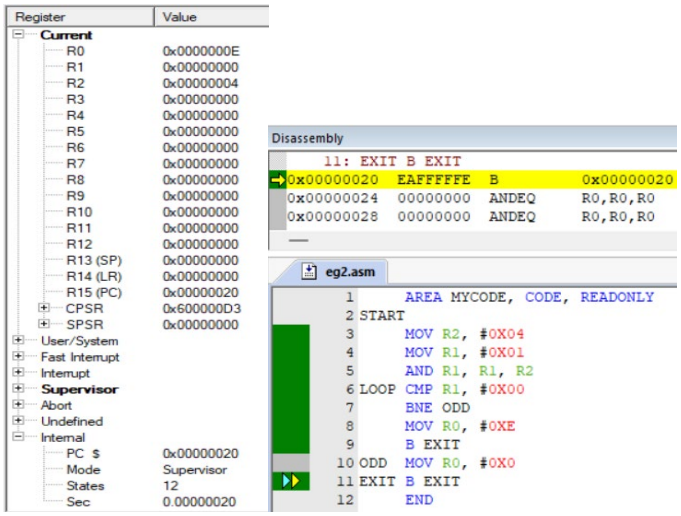
CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \& N$

Example 3: The following code checks whether the given input value (in register r2) is odd or even using compare instruction; if the value is even then it moves E to register r0 else moves

0. Attach the screenshot of final status of PC and relevant registers. Briefly specify how the condition codes get affected in CPSR.

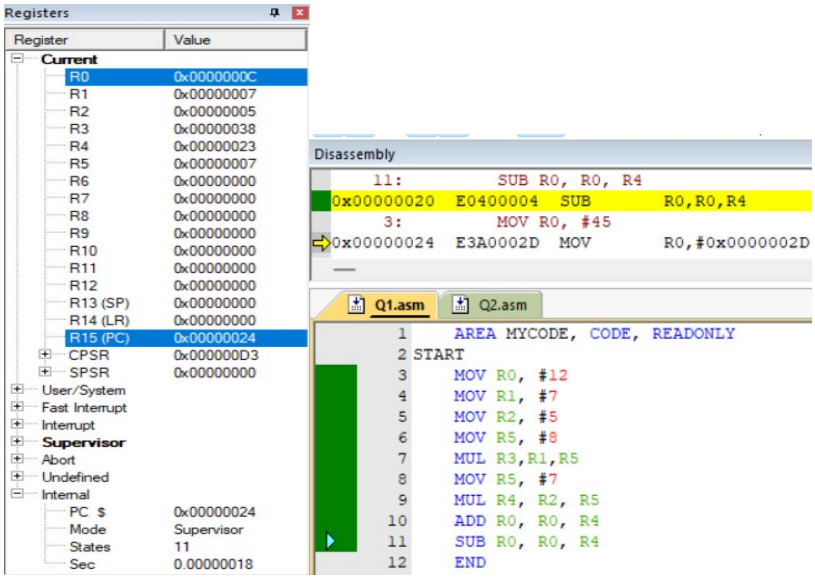




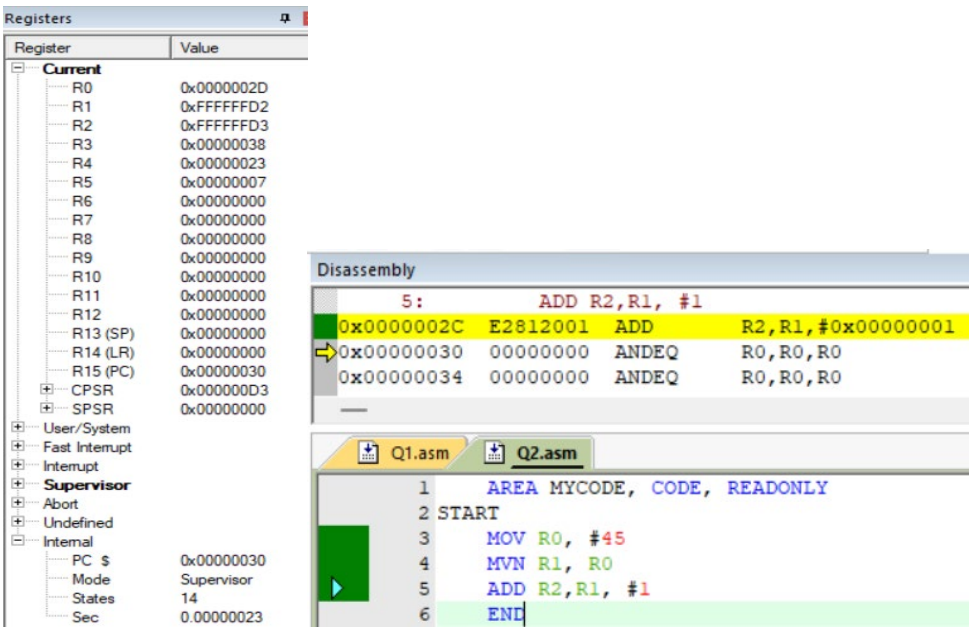


Exercise

1. Perform arithmetic operation for evaluating A + 8B – 7C where A = 12, B = 7 and C = 5 using appropriate arithmetic instructions. Attach the screenshot of the code and register window.



2. Compute the 2's complement of a number. Load the input value in register R0 and store the output in register R2. Attach the screenshot of the code and register window.



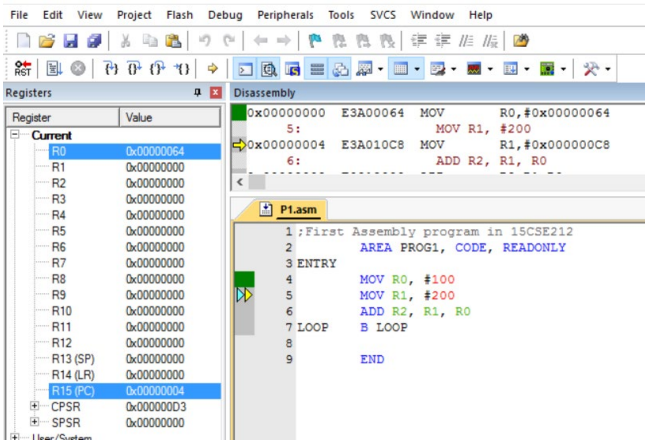
Decimal: 45

Binary: 0000 0000 0010 1101

Hex: 002D

1's Complement: 1111 1111 1101 0010

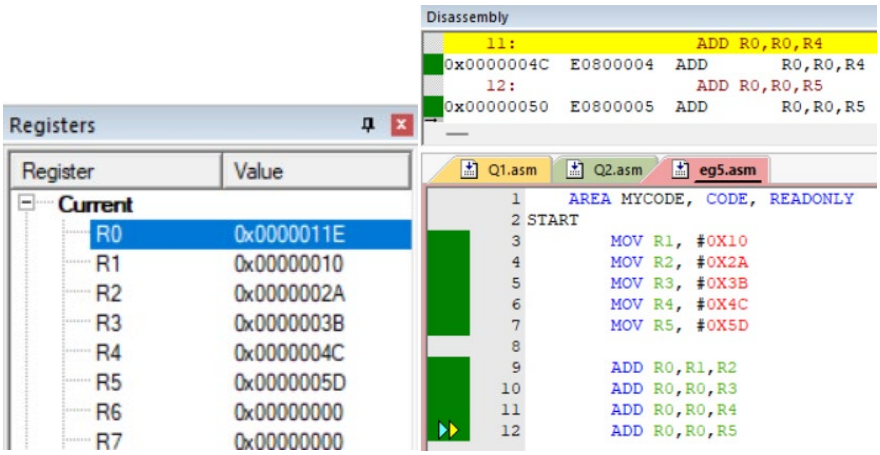
2's Complement: 1111 1111 1101 0011



1. Based on the screenshot above, What is the current instruction that it is executing/debugging?

MOV R0,#100 is the instruction as specified in the P1.asm file and it is located in the address **0X00000004** as shown in the listing in the disassembly window and the current instruction address is specified in the register **R0**.

1. After the execution of the entire program once, R0 contains the value **0X00000064**, which is actually the hexadecimal of the decimal **100**.R1 contains the value **0X000000C8**, which is actually the hexadecimal of the decimal **200**. R2 contains the value **0X0000012C**, which is actually the hexadecimal of the decimal **300**.
2. Write your assembly program to add the contents of registers R1, R2, R3, R4 and R5 and save the result in R0. You can initialize the contents of R1 to R5 as 0x10, 0x2A, 0x3B, 0x4C and 0x5D respectively. Ensure that you create this program file in the folder having the name as your roll number. Give the screenshot

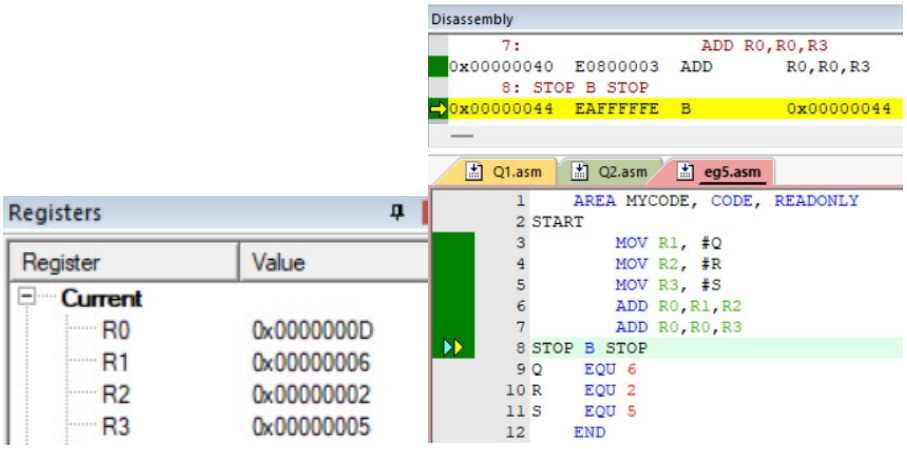


Problem P=Q+R+S with Q=6, R=2, S=5 and assume R1=Q, R2=R and R3=S. We can use symbolic names Q, R and S using EQU (equate) assembler directive.

The problem is solved using the assembly program shown below

3. Execute this program which is saved in a folder renamed as your roll number. Give the screenshot of successful execution and answer the following questions.
4. Which addressing mode is used in the first 3 assembly instruction in the program?

   Immediate Addressing Mode
5. Give proper proof with the screenshot captured during the execution in the debug mode.

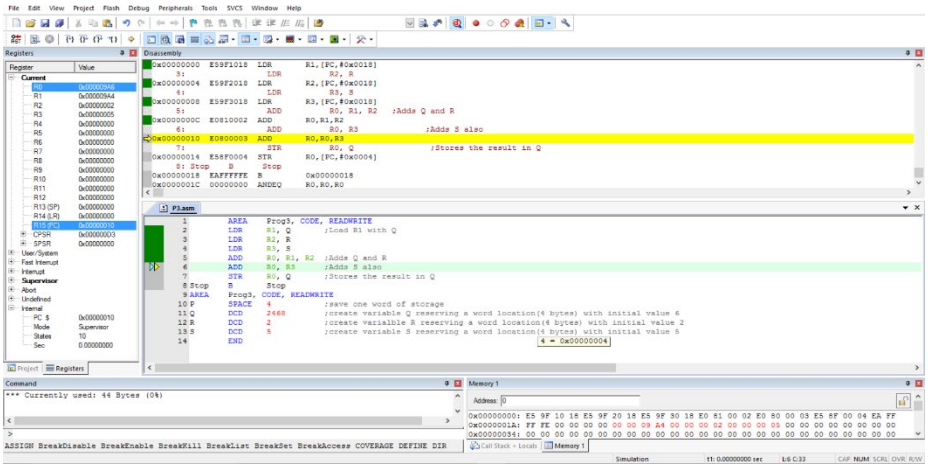


6. What is the final value of R0? **0x0000000D**

Same Problem P=Q+R+S with Q=6, R=2, S=5 and assume R1=Q, R2=R and R3=S. We will use load register LDR R1, Q instruction to load register R1 with the contents of memory location Q. This instruction does not exist and is not part of the ARM's instruction set. However, the ARM assembler automatically changes it into actual instruction.

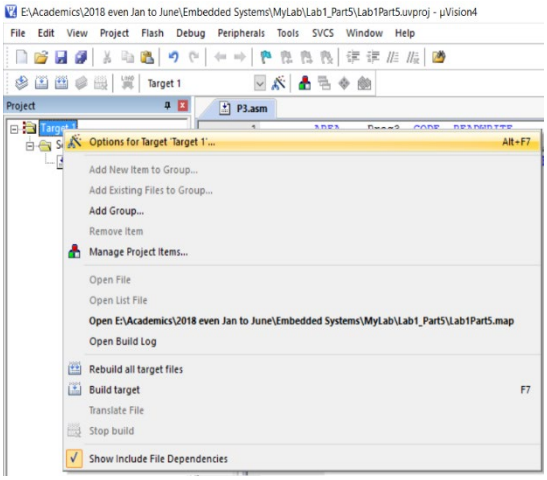
Answer the following questions:

1. The code generated by ARM in the disassembly window for the pseudo instruction LDR R1, Q is **MOV R1, #6**
2. The value stored in the address 0x0000001A is **FF EE 00 00**.
3. The value of PC when it is executing the first instruction is **0x00000000**
4. The addressing mode used in the LDR instruction is **Immediate Addressing Mode** , because **LDR R1, Q**
5. The address of R is **0X0000004**, with the value 2 in memory.
    a. The address of S is **0X00000008**, with the value 5 in memory



6. In Little Endian (Current Scenario), the first byte address of Q is **0X00000000**, and it stores the value **E5**. The second byte address of Q is **0X00000001**, and it stores the value **9F**. The third byte address of Q is **0X00000002**, and it stores the value **10**. The fourth byte address of Q is **0X00000003**, and it stores the value **18**.
7. The address [PC]+0x0018+8 = **0X0000000C**. If [PC] = 0x00000000 for the first instruction, then this address should refer to the starting address of Q. Note 8 bytes is added in ARM7, because of 3-stage pipeline concept.
8. You can change the Device ARM7 Little Endian to ARM7 Big Endian as shown in the below screenshot

Right click Target1 and select Options as shown below



Now select Device tab in the opening window and select ARM7 big Endian as shown below.

In Little Endian, the first byte address of Q is **0X00000000** and it stores the value **18**. The second byte address of Q is **0X00000001** and it stores the value **10**. The third byte address of Q is **0X00000002** and it stores the value **9F**. The fourth byte address of Q is **0X00000003** and it stores the value **E5**.

9. If X=0x12345678 and starting address of X is 0x00000010, How this value will be stored in Big Endian and Little Endian?

Little Endian:

0x00000013	1 2
0x00000012	3 4
0x00000011	5 6
0x00000011	7 8

Big Endian:

0x00000013	7 8
0x00000012	5 6
0x00000011	3 4
0x00000010	1 2