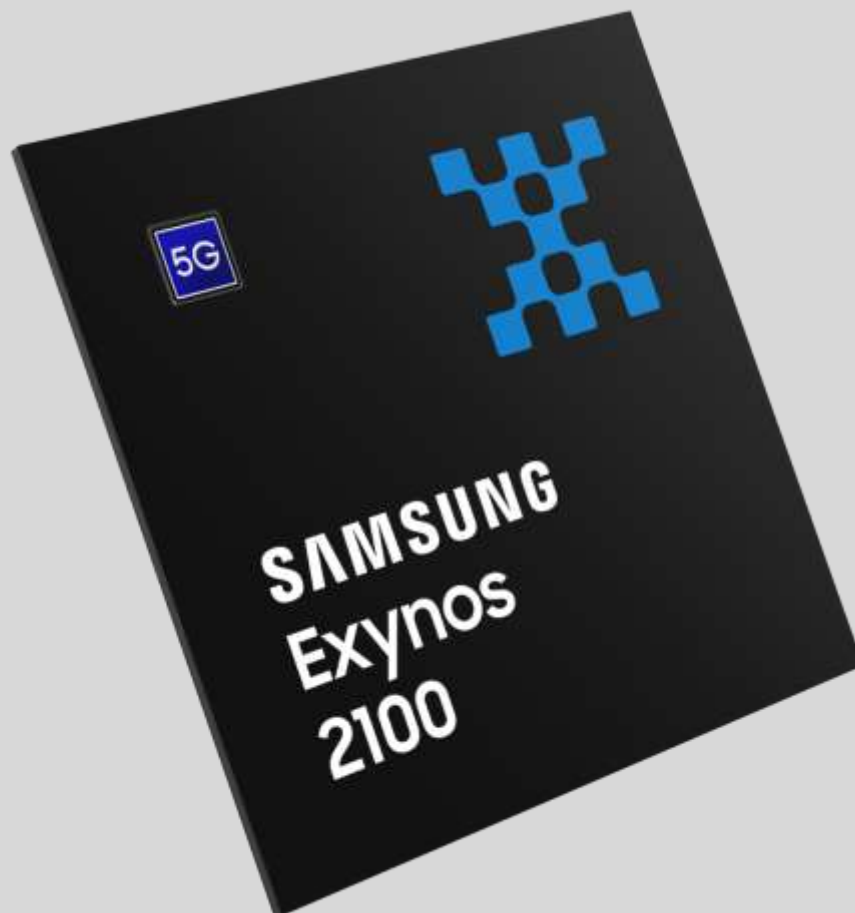


EC-210

MICROPROCESSORS LAB

LAB-1



UTKARSH MAHAJAN 201EC164

ARNAV RAJ 201EC109

Objective: The aim of this lab is to introduce to ARM assembly level programming and use of KEIL μ vision tools.

Exercise:

1.2] Observe the use of MOV and MVN instructions in loading a 32 bit immediate data into the registers.

Syntax for MOV:

```
MOV{cond}{S} Rd, Operand2
```

for MVN:

```
MVN{cond}{S} Rd, Operand2
```

S: is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation

Cond: is an optional condition code

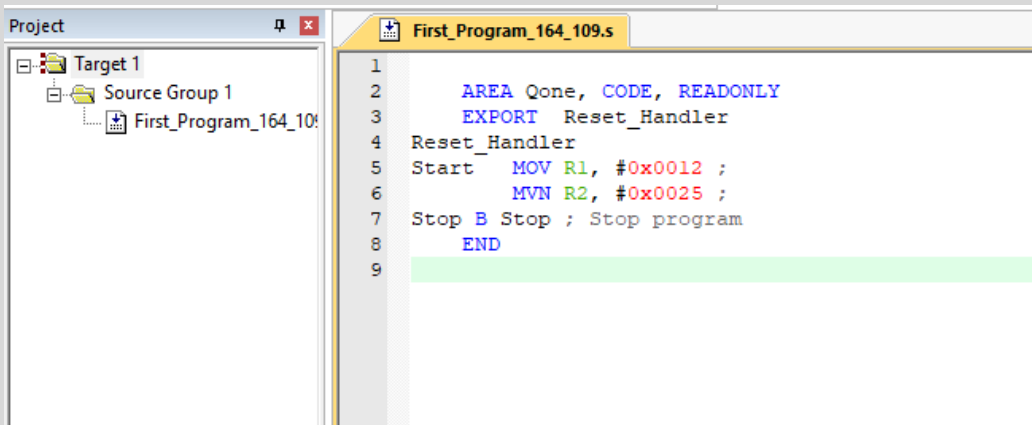
Rd: is the ARM register for the result.

Operand2: is a flexible second operand.

Source Code for the exercise:

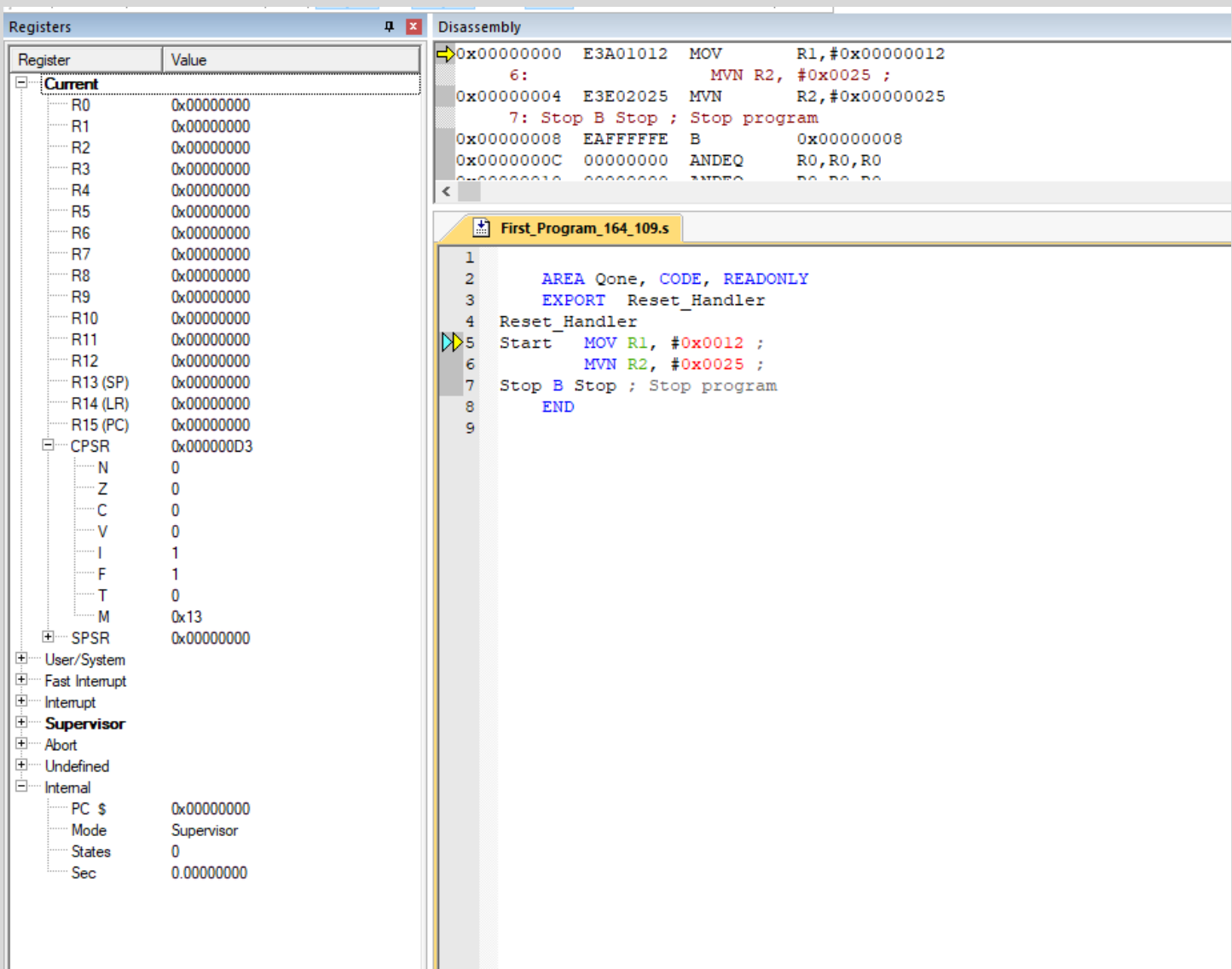
```
AREA Qone, CODE, READONLY
EXPORT Reset_Handler
Reset_Handler
Start    MOV R1, #0x0012 ;
        MVN R2, #0x0025 ;
Stop B Stop ; Stop program
END
```

Keil µvision setup:



Debugging

Initially:



Step 1:

The screenshot displays a debugger interface with two main panels: **Registers** and **Disassembly**.

Registers Panel:

Register	Value
R0	0x00000000
R1	0x00000012
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)	0x00000004
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000004
Mode	Supervisor
States	1
Sec	0.00000002

Disassembly Panel:

```
5: Start  MOV R1, #0x0012 ;
0x00000000 E3A01012 MOV R1,#0x00000012
6:        MVN R2, #0x0025 ;
0x00000004 E3E02025 MVN R2,#0x00000025
7: Stop B Stop ; Stop program
0x00000008 EAffffff B 0x00000008
0x0000000C 00000000 ANDEQ R0,R0,R0
0x00000010 00000000 ANDEQ R0,R0,R0
```

The **First_Program_164_109.s** file is open, showing the following assembly code:

```
1
2 AREA Qone, CODE, READONLY
3 EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0x0012 ;
6      MVN R2, #0x0025 ;
7 Stop B Stop ; Stop program
8      END
9
```

We can see that the immediate value 0x0012(18 in decimal) gets loaded in R1.

Step 2:

We can see that the value loaded in R2 is 0xFFFFFDA which is a bit-wise negation of our operand2 immediate value 0x0025 in 32bits.

The screenshot displays a debugger interface with two main panels: **Registers** and **Disassembly**.

Registers Panel:

Register	Value
R0	0x00000000
R1	0x00000012
R2	0xFFFFFDA
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)	0x00000008
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000008
Mode	Supervisor
States	2
Sec	0.00000003

Disassembly Panel:

First_Program_164_109.s

```

1
2   AREA Qone, CODE, READONLY
3   EXPORT Reset_Handler
4   Reset_Handler
5   Start  MOV R1, #0x0012 ;
6         MVN R2, #0x0025 ;
7   Stop B Stop ; Stop program
8   END
9

```

Step 3:

here the program stops and everything after the END is ignored.

Register	Value
Current	
R0	0x00000000
R1	0x00000012
R2	0xFFFFFDA
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)	0x00000008
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	


```

5: Start  MOV R1, #0x0012 ;
0x00000000 E3A01012 MOV      R1,#0x00000012
6:        MVN R2, #0x0025 ;
0x00000004 E3E02025 MVN      R2,#0x00000025
7: Stop B Stop ; Stop program
0x00000008 EAffffff B        0x00000008
0x0000000C 00000000 ANDEQ    R0,R0,R0
0x00000010 00000000 ANDEQ    R0,R0,R0

```



```

1
2      AREA Qone, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0x0012 ;
6        MVN R2, #0x0025 ;
7 Stop B Stop ; Stop program
8      END
9

```

Observation:

From the above we can see that,

MOV instruction stores the value of operand2 into Rd (destination register).

MVN instruction stores the result of bitwise NOT operation of 32-bit extended operand2 value into Rd (destination Register).

1.3] Demonstrate the use of LSL, LSR, ASR, ROR, RRX with MOV, MVN, MOVS and MNS for different data. Observe the results and conditional code flags in CPSR.

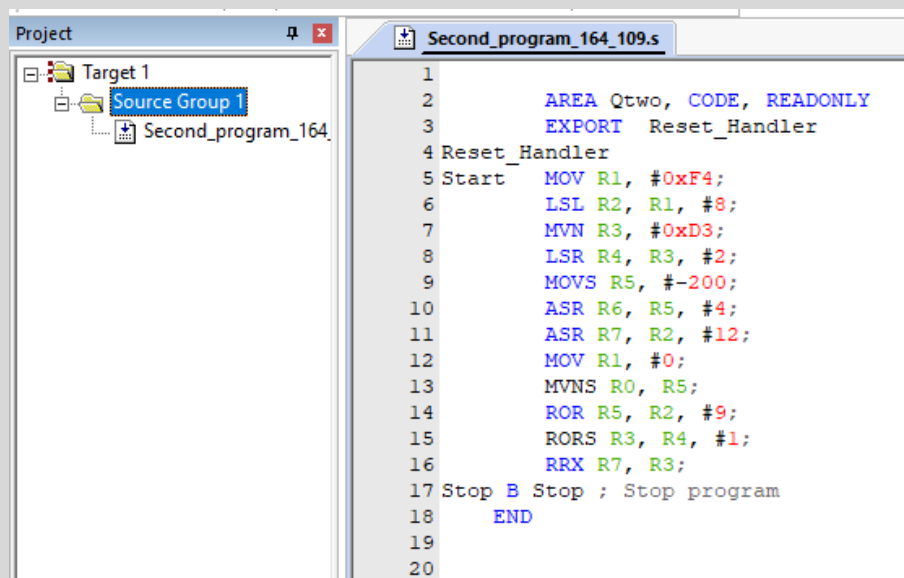
-> To demonstrate the use of , LSR, ASR, ROR, RRX with MOV, MVN, MOVS and MNS, we will write a source code using these instruction and try debugging it in Keil µVision 4 to display its working

Source Code for the exercise:

```
        AREA Qtwo, CODE, READONLY
        EXPORT Reset_Handler
Reset_Handler
Start    MOV R1, #0xF4;
        LSL R2, R1, #8;
        MVN R3, #0xD3;
        LSR R4, R3, #2;
        MOVS R5, #-200;
        ASR R6, R5, #4;
        ASR R7, R2, #12;
        MOV R1, #0;
        MVNS R0, R5;
        ROR R5, R2, #9;
        RORS R3, R4, #1;
        RRX R7, R3;

Stop B Stop ; Stop program
END
```

Keil µvision setup:



Debugging and Observations:

Initially:

The screenshot displays a debugger interface with two main panels: 'Registers' on the left and 'Disassembly' on the right.

Registers Panel:

Register	Value
Current	
R0	0x00000000
R1	0x00000000
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)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

Disassembly Panel:

5: Start MOV R1, #0xF4;
→ 0x00000000 E3A010F4 MOV R1, #0x000000F4
6: LSL R2, R1, #8;
0x00000004 E1A02401 MOV R2, R1, LSL #8
7: MVN R3, #0xD3;
0x00000008 E3E030D3 MVN R3, #0x000000D3
8: LSR R4, R3, #2;
0x0000000C E1A04123 MOV R4, R3, LSR #2
9: MOVS R5, #-200;

Second_program_164_109.s

```
1
2     AREA Qtwo, CODE, READONLY
3     EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6     LSL R2, R1, #8;
7     MVN R3, #0xD3;
8     LSR R4, R3, #2;
9     MOVS R5, #-200;
10    ASR R6, R5, #4;
11    ASR R7, R2, #12;
12    MOV R1, #0;
13    MVNS R0, R5;
14    ROR R5, R2, #9;
15    RORS R3, R4, #1;
16    RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20
```

Step1:

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
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)	0x00000004
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000004
Mode	Supervisor
States	1
Sec	0.00000002

Disassembly

```

5: Start  MOV R1, #0xF4;
0x00000000 E3A010F4 MOV R1,#0x000000F4
6:      LSL R2, R1, #8;
0x00000004 E1A02401 MOV R2,R1,LSL #8
7:      MVN R3, #0xD3;
0x00000008 E3E030D3 MVN R3,#0x000000D3
8:      LSR R4, R3, #2;
0x0000000C E1A04123 MOV R4,R3,LSR #2
9:      MOVS R5, #-200;

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

MOV instruction stores the immediate value 0xF4(hexadecimal) in the 32-bit register R1.

Step2:

LSL instruction logically left shifts the value stored in register R1 by 8 bits and is stores it in R2.

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
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)	0x00000008
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000008
Mode	Supervisor
States	2
Sec	0.00000003

Disassembly

```

5: Start  MOV R1, #0xF4;
0x00000000 E3A010F4 MOV      R1,#0x000000F4
6:                LSL R2, R1, #8;
0x00000004 E1A02401 MOV      R2,R1,LSL #8
7:                MVN R3, #0xD3;
0x00000008 E3E030D3 MVN      R3,#0x000000D3
8:                LSR R4, R3, #2;
0x0000000C E1A04123 MOV      R4,R3,LSR #2
9:                MOVS R5, #-200;

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0xF4;
6        LSL R2, R1, #8;
7        MVN R3, #0xD3;
8        LSR R4, R3, #2;
9        MOVS R5, #-200;
10       ASR R6, R5, #4;
11       ASR R7, R2, #12;
12       MOV R1, #0;
13       MVNS R0, R5;
14       ROR R5, R2, #9;
15       RORS R3, R4, #1;
16       RRX R7, R3;
17 Stop B Stop ; Stop program
18      END
19
20

```

Step 3:

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
R3	0xFFFFF2C
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	3
Sec	0.00000005

Disassembly

```

5: Start  MOV R1, #0xF4;
0x00000000 E3A010F4 MOV      R1,#0x000000F4
6:                LSL R2, R1, #8;
0x00000004 E1A02401 MOV      R2,R1,LSL #8
7:                MVN R3, #0xD3;
0x00000008 E3E030D3 MVN      R3,#0x000000D3
8:                LSR R4, R3, #2;
0x0000000C E1A04123 MOV      R4,R3,LSR #2
9:                MOVS R5, #-200;

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0xF4;
6        LSL R2, R1, #8;
7        MVN R3, #0xD3;
8        LSR R4, R3, #2;
9        MOVS R5, #-200;
10       ASR R6, R5, #4;
11       ASR R7, R2, #12;
12       MOV R1, #0;
13       MVNS R0, R5;
14       ROR R5, R2, #9;
15       RORS R3, R4, #1;
16       RRX R7, R3;
17 Stop B Stop ; Stop program
18      END
19

```

We can see that MVN instruction stores the value 0xFFFFF2C which is a bit-wise negation of our operand2 immediate value 0xDB in 32bits.

Step 4:

The screenshot displays the ARM Register window and the Disassembly window. The Register window shows the current state of the registers, with R4 containing 0x3FFFFFFCB and R15 (PC) containing 0x00000010. The Disassembly window shows the instructions being executed, including MVN, LSR, MOV, and ASR instructions.

Register	Value
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
R3	0xFFFFF2C
R4	0x3FFFFFFCB
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000010
Mode	Supervisor
States	4
Sec	0.00000007

```
7:          MVN R3, #0xD3;
0x00000008  E3E030D3  MVN      R3, #0x000000D3
8:          LSR R4, R3, #2;
0x0000000C  E1A04123  MOV      R4, R3, LSR #2
9:          MOVS R5, #-200;
0x00000010  E3F050C7  MVNS     R5, #0x000000C7
10:         ASR R6, R5, #4;
0x00000014  E1A06245  MOV      R6, R5, ASR #4
11:         ASR R7, R2, #12;
```

Second_program_164_109.s

```
1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0xF4;
6        LSL R2, R1, #8;
7        MVN R3, #0xD3;
8        LSR R4, R3, #2;
9        MOVS R5, #-200;
10       ASR R6, R5, #4;
11       ASR R7, R2, #12;
12       MOV R1, #0;
13       MVNS R0, R5;
14       ROR R5, R2, #9;
15       RORS R3, R4, #1;
16       RRX R7, R3;
17 Stop B Stop ; Stop program
18      END
19
```

We can see that LSR instruction stores the 2-bits logically right shifted value of R3 into R4 with 0 in vacated bits location.

Step 5:

MOVS instruction here stores the value of the operand2 (-200 here) into register R5 and updates the register CPSR's N and Z flags. Since -200 is negative and not zero. N will be set to 1 and Z to 0.

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
R3	0xFFFFFFFF2C
R4	0x3FFFFFFCB
R5	0xFFFFFFFF38
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000014
CPSR	0x800000D3
<div> <div>N</div> <div>1</div> </div>	
Z 0	
C 0	
V 0	
I 1	
F 1	
T 0	
M 0x13	
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000014
Mode	Supervisor
States	5
Sec	0.00000008

Disassembly

9:

MOVNS R5, #-200;

0x00000010

E3F050C7

MOVNS R5, #0x000000C7

10:

ASR R6, R5, #4;

0x00000014

E1A06245

MOV R6, R5, ASR #4

11:

ASR R7, R2, #12;

0x00000018

E1A07642

MOV R7, R2, ASR #12

12:

MOV R1, #0;

0x0000001C

E3A01000

MOV R1, #0x00000000

13:

MOVNS R0, R5;

Second_program_164_109.s

```

1
2     AREA Qtwo, CODE, READONLY
3     EXPORT Reset_Handler
4 Reset_Handler
5 Start  MOV R1, #0xF4;
6        LSL R2, R1, #8;
7        MVN R3, #0xD3;
8        LSR R4, R3, #2;
9        MOVS R5, #-200;
10       ASR R6, R5, #4;
11       ASR R7, R2, #12;
12       MOV R1, #0;
13       MOVNS R0, R5;
14       ROR R5, R2, #9;
15       RORS R3, R4, #1;
16       RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 6:

Here, ASR instruction right shifts the value stored in register R5(0xFFFFFFFF38) by 4bits and stores it in Register R6(0xFFFFFFFF3) and copies the sign bit into vacated bit positions on the left.

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
R3	0xFFFFFFFF2C
R4	0x3FFFFFFCB
R5	0xFFFFFFFF38
R6	0xFFFFFFFFF3
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000018
Mode	Supervisor
States	6
Sec	0.00000010

Disassembly

9:

MOVNS R5, #-200;

0x00000010

E3F050C7

MOVNS R5, #0x000000C7

10:

ASR R6, R5, #4;

0x00000014

E1A06245

MOV R6, R5, ASR #4

11:

ASR R7, R2, #12;

0x00000018

E1A07642

MOV R7, R2, ASR #12

12:

MOV R1, #0;

0x0000001C

E3A01000

MOV R1, #0x00000000

13:

MOVNS R0, R5;

Second_program_164_109.s

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

AREA Qtwo, CODE, READONLY

EXPORT Reset_Handler

Reset_Handler

Start

MOV R1, #0xF4;

LSL R2, R1, #8;

MVN R3, #0xD3;

LSR R4, R3, #2;

MOVNS R5, #-200;

ASR R6, R5, #4;

ASR R7, R2, #12;

MOV R1, #0;

MOVNS R0, R5;

ROR R5, R2, #9;

RORS R3, R4, #1;

RRX R7, R3;

Stop B Stop ; Stop program

END

Step 7:

Similar to the previous step of execution, we can see that the value stored in register R2(0x0000F400) right shifted by 12bits with left vacant bits copying the sign bit is stored in register R7(0x0000000F).

Registers

Register	Value
Current	
R0	0x00000000
R1	0x000000F4
R2	0x0000F400
R3	0xFFFFFFFF2C
R4	0x3FFFFFFCB
R5	0xFFFFFFFF38
R6	0xFFFFFFFFF3
R7	0x0000000F
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001C
CPSR	0x800000D3
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x0000001C
Mode	Supervisor
States	7
Sec	0.00000012

Disassembly

```

10:                                ASR R6, R5, #4;
0x00000014 E1A06245 MOV        R6,R5,ASR #4
11:                                ASR R7, R2, #12;
0x00000018 E1A07642 MOV        R7,R2,ASR #12
12:                                MOV R1, #0;
0x0000001C E3A01000 MOV        R1,#0x00000000
13:                                MVNS R0, R5;
0x00000020 E1F00005 MVNS        R0,R5
14:                                ROR R5, R2, #9;

```

Second_program_164_109.s

```

1
2     AREA Qtwo, CODE, READONLY
3     EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6     LSL R2, R1, #8;
7     MVN R3, #0xD3;
8     LSR R4, R3, #2;
9     MOVS R5, #-200;
10    ASR R6, R5, #4;
11    ASR R7, R2, #12;
12    MOV R1, #0;
13    MVNS R0, R5;
14    ROR R5, R2, #9;
15    RORS R3, R4, #1;
16    RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 8:

MOV instruction stores the immediate value 0 in the 32-bit register R1. We can see that it does not affect(update) the Zflag of the CPSR due to the absence of S suffix.

Registers

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x0000F400
R3	0xFFFFFFFF2C
R4	0x3FFFFFFCB
R5	0xFFFFFFFF38
R6	0xFFFFFFFFF3
R7	0x0000000F
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000020
CPSR 0x800000D3	
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
SPSR 0x00000000	
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000020
Mode	Supervisor
States	8
Sec	0.00000013

Disassembly

```

11:                                ASR R7, R2, #12;
0x00000018 E1A07642 MOV          R7,R2,ASR #12
12:                                MOV R1, #0;
0x0000001C E3A01000 MOV          R1,#0x00000000
13:                                MVNS R0, R5;
0x00000020 E1F00005 MVNS          R0,R5
14:                                ROR R5, R2, #9;
0x00000024 E1A054E2 MOV          R5,R2,ROR #9
15:                                RORS R3, R4, #1;

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 9:

We can see that MVNS instruction stores the value 0x000000C7 which is a bit-wise negation of the value stored in register R5 and also updates the N and Z flags. Here the stored value is positive hence N and Z are updated to 0.

Registers

Register	Value
Current	
R0	0x000000C7
R1	0x00000000
R2	0x0000F400
R3	0xFFFFF2C
R4	0x3FFFFFFCB
R5	0xFFFFF38
R6	0xFFFFF3
R7	0x0000000F
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000024
CPSR	0x000000D3
<div> <div>N</div> <div>0</div> </div> <div> <div>Z</div> <div>0</div> </div> <div> <div>C</div> <div>0</div> </div> <div> <div>V</div> <div>0</div> </div> <div> <div>I</div> <div>1</div> </div> <div> <div>F</div> <div>1</div> </div> <div> <div>T</div> <div>0</div> </div> <div> <div>M</div> <div>0x13</div> </div>	
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000024
Mode	Supervisor
States	9
Sec	0.00000015

Disassembly

```

11:          ASR R7, R2, #12;
0x00000018 E1A07642 MOV     R7,R2,ASR #12
12:          MOV R1, #0;
0x0000001C E3A01000 MOV     R1,#0x00000000
13:          MVNS R0, R5;
0x00000020 E1F00005 MVNS    R0,R5
14:          ROR R5, R2, #9;
0x00000024 E1A054E2 MOV     R5,R2,ROR #9
15:          RORS R3, R4, #1;

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 10:

We can see that the ROR instruction rotates the value stored in register R2 towards the right by 9 bits as specified and gets it gets stored in Register R5.

Registers

Register	Value
Current	
R0	0x000000C7
R1	0x00000000
R2	0x0000F400
R3	0xFFFFFFFF2C
R4	0x3FFFFFFCB
R5	0x0000007A
R6	0xFFFFFFFFF3
R7	0x0000000F
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000028
CPSR	0x000000D3
<div> <div>N</div> <div>0</div> </div> <div> <div>Z</div> <div>0</div> </div> <div> <div>C</div> <div>0</div> </div> <div> <div>V</div> <div>0</div> </div> <div> <div>I</div> <div>1</div> </div> <div> <div>F</div> <div>1</div> </div> <div> <div>T</div> <div>0</div> </div> <div> <div>M</div> <div>0x13</div> </div>	
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000028
Mode	Supervisor
States	10
Sec	0.00000017

Disassembly

12:

MOV R1, #0;

0x0000001C

E3A01000

MOV R1, #0x00000000

13:

MVNS R0, R5;

0x00000020

E1F00005

MVNS R0, R5

14:

ROR R5, R2, #9;

0x00000024

E1A054E2

MOV R5, R2, ROR #9

15:

RORS R3, R4, #1;

0x00000028

E1B030E4

MOVS R3, R4, ROR #1

16:

RRX R7, R3;

Second_program_164_109.s

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

AREA Qtwo, CODE, READONLY

EXPORT Reset_Handler

Reset_Handler

Start

MOV R1, #0xF4;

LSL R2, R1, #8;

MVN R3, #0xD3;

LSR R4, R3, #2;

MOVS R5, #-200;

ASR R6, R5, #4;

ASR R7, R2, #12;

MOV R1, #0;

MVNS R0, R5;

ROR R5, R2, #9;

RORS R3, R4, #1;

RRX R7, R3;

Stop B Stop ; Stop program

END

Step 11:

Here, RORS instruction rotates the value of register R4 by a bit towards the right and that value is stored in register R3. Since the S suffix is used. It will also update the CPSR flags. The carry flag C gets updated with the value stored in the right most bit of the before the last rotation (here it rotates once hence before the rotation). The N and Z flags are also updated according to the final value stored in register R3. N for negative and Z for zero value.

Registers

Register	Value
Current	
R0	0x000000C7
R1	0x00000000
R2	0x0000F400
R3	0x9FFFFFFE5
R4	0x3FFFFFFCB
R5	0x0000007A
R6	0xFFFFFFFF3
R7	0x0000000F
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000002C
CPSR	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x0000002C
Mode	Supervisor
States	11
Sec	0.00000018

Disassembly

0x0000001C	E3A01000	MOV	R1,#0x00000000
13:		MVNS	R0, R5;
0x00000020	E1F00005	MVNS	R0,R5
14:		ROR	R5, R2, #9;
0x00000024	E1A054E2	MOV	R5,R2,ROR #9
15:		RORS	R3, R4, #1;
0x00000028	E1B030E4	MOVS	R3,R4,ROR #1
16:		RRX	R7, R3;
0x0000002C	E1A07063	MOV	R7,R3,RRX

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 12: Here, The RRX instruction rotates the value stored in register R3 towards right by 1 bit and replaces the MSB (most significant bit) with the C flag stored in CPSR and later stores this value in register R7. Since in the previous instruction C flag was set to 1. The MSB here is set to 1 in register R7.

Registers

Register	Value
Current	
R0	0x000000C7
R1	0x00000000
R2	0x0000F400
R3	0x9FFFFFFE5
R4	0x3FFFFFFCB
R5	0x0000007A
R6	0xFFFFFFFF3
R7	0xCFFFFFF2
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000030
CPSR 0xA00000D3	
N 1	
Z 0	
C 1	
V 0	
I 1	
F 1	
T 0	
M 0x13	
SPSR 0x00000000	
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$ 0x00000030	
Mode Supervisor	
States 12	
Sec 0.00000020	

Disassembly

```

16:                                RRX R7, R3;
0x0000002C E1A07063 MOV          R7,R3,RRX
17: Stop B Stop ; Stop program
0x00000030 EAffffff B          0x00000030
0x00000034 00000000 ANDEQ       R0,R0,R0
0x00000038 00000000 ANDEQ       R0,R0,R0
0x0000003C 00000000 ANDEQ       R0,R0,R0
0x00000040 00000000 ANDEQ       R0,R0,R0
0x00000044 00000000 ANDEQ       R0,R0,R0

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Step 13: here the program stops and everything after the END is ignored.

Registers

Register	Value
Current	
R0	0x000000C7
R1	0x00000000
R2	0x0000F400
R3	0x9FFFFFFE5
R4	0x3FFFFFFCB
R5	0x0000007A
R6	0xFFFFFFFF3
R7	0xCFFFFFFF2
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000030
CPSR	0xA00000D3
<div> <div>N</div> <div>1</div> </div> <div> <div>Z</div> <div>0</div> </div> <div> <div>C</div> <div>1</div> </div> <div> <div>V</div> <div>0</div> </div> <div> <div>I</div> <div>1</div> </div> <div> <div>F</div> <div>1</div> </div> <div> <div>T</div> <div>0</div> </div> <div> <div>M</div> <div>0x13</div> </div>	
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000030
Mode	Supervisor
States	15
Sec	0.00000025

Disassembly

```

16:                                RRX R7, R3;
0x0000002C E1A07063 MOV        R7,R3,RRX
17: Stop B Stop ; Stop program
0x00000030 EAffffFE B          0x00000030
0x00000034 00000000 ANDEQ     R0,R0,R0
0x00000038 00000000 ANDEQ     R0,R0,R0
0x0000003C 00000000 ANDEQ     R0,R0,R0
0x00000040 00000000 ANDEQ     R0,R0,R0
0x00000044 00000000 ANDEQ     R0,R0,R0

```

Second_program_164_109.s

```

1
2      AREA Qtwo, CODE, READONLY
3      EXPORT Reset_Handler
4 Reset_Handler
5 Start MOV R1, #0xF4;
6      LSL R2, R1, #8;
7      MVN R3, #0xD3;
8      LSR R4, R3, #2;
9      MOVS R5, #-200;
10     ASR R6, R5, #4;
11     ASR R7, R2, #12;
12     MOV R1, #0;
13     MVNS R0, R5;
14     ROR R5, R2, #9;
15     RORS R3, R4, #1;
16     RRX R7, R3;
17 Stop B Stop ; Stop program
18     END
19
20

```

Results:

We can see that the instructions used in the program update the flag Values stored in CPSR only when the S suffix is used.

From above we can infer that the uses for the following instructions are:

LSL: provides the value of a register multiplied by a power of two, inserting zeros into the vacated bit positions.

- If S is specified, the LSL instruction updates the N and Z flags according to the result.
- The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.

LSR: It provides the unsigned value of a register divided by a variable power of two, inserting zeros into the vacated bit positions.

- If S is specified, the instruction updates the N and Z flags according to the result.
- The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.

ASR: It provides the signed value of the contents of a register divided by a power of two. It copies the sign bit into vacated bit positions on the left.

If S is specified, the ASR instruction updates the N and Z flags according to the result.

The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.

ROR: It provides the value of the contents of a register rotated by a value. The bits that are rotated off the right end are inserted into the vacated bit positions on the left.

- If S is specified, the instruction updates the N and Z flags according to the result.
- The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.

RRX: It provides the value of the contents of a register shifted right one bit. The old carry flag is shifted into bit[31]. If the S suffix is present, the old bit[0] is placed in the carry flag

- If S is specified, the instruction updates the N and Z flags according to the result.
- The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.