

Microprocessors LAB  
EC210  
Assignment 1

Submitted By:

Suhas S G

201EC263

Nikhil P Reddy

201EC241

### Program 1.1

```

AREA      MyFirst, CODE, READONLY
EXPORT    Reset_Handler

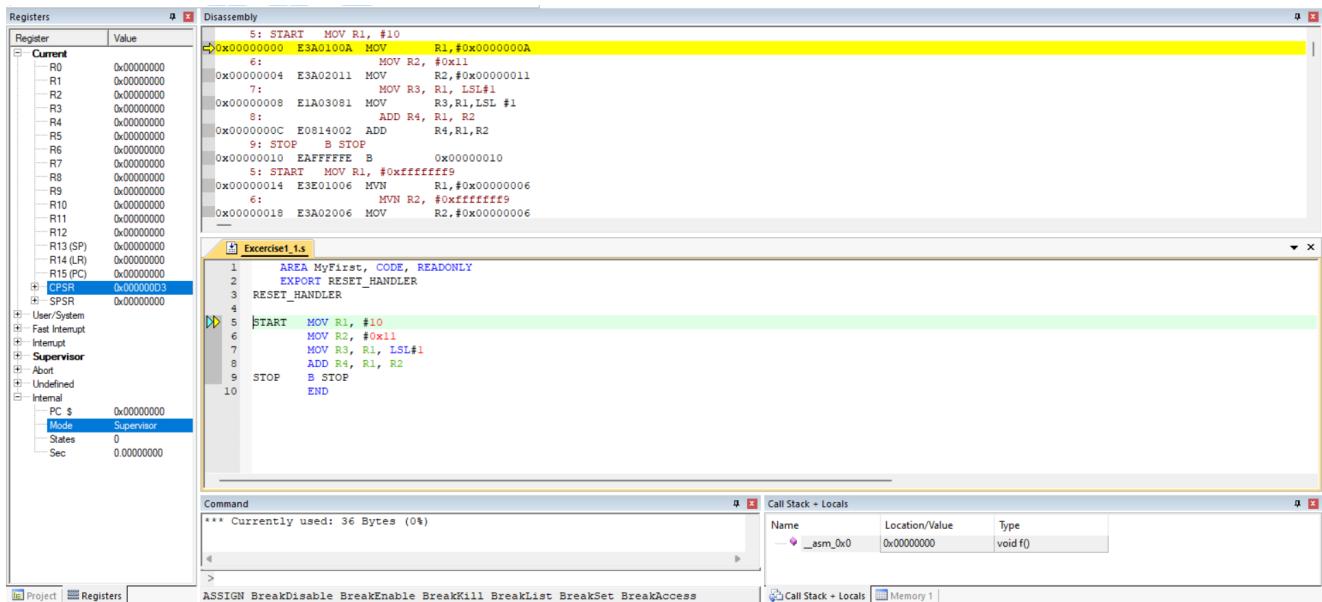
Reset_Handler

Start      MOV R1, #10           ; Load a decimal Value
          MOV R2, #0x11         ; Load an initial Hex Value
          MOV R3, R1, LSL#1     ; Shift left by 1bit
          ADD R4, R1, R2

Stop       B STOP             ; Stop program
          END

```

## Step-by-Step Execution



Program is compiled/translated and set to debug mode.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current state of various registers. Register R1 has a value of 0x0000000A.
- Disassembly**: Displays the assembly code for the program. The instruction at address 0x00000004 is highlighted: `MOV R2, #0x11`.
- Exercisel\_1.s**: Shows the source code for the assembly file. The instruction at line 7 is highlighted: `MOV R3, R1, LSL#1`.
- Command**: A text input field showing the memory usage: "Currently used: 36 Bytes (0%)".
- Call Stack + Locals**: A table showing the current stack frame. It contains one entry: `__asm_0x0` with a value of 0x00000000.

The decimal value 10 (0x0000000A) is loaded into register R1.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current state of various registers. Register R2 has a value of 0x00000011.
- Disassembly**: Displays the assembly code for the program. The instruction at address 0x00000008 is highlighted: `MOV R3, R1, LSL#1`.
- Exercisel\_1.s**: Shows the source code for the assembly file. The instruction at line 7 is highlighted: `MOV R3, R1, LSL#1`.
- Command**: A text input field showing the memory usage: "Currently used: 36 Bytes (0%)".
- Call Stack + Locals**: A table showing the current stack frame. It contains one entry: `__asm_0x0` with a value of 0x00000000.

The hexadecimal value 0x11 is loaded into register R2.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. R1 is highlighted with the value 0x0000000A.
- Disassembly**: Shows the assembly code for the program. Line 8 shows the instruction ADD R4, R1, R2.
- Exerciset\_1.s**: Shows the source code for the assembly file. It includes a RESET\_HANDLER section with the same ADD instruction.
- Command**: Shows the current memory usage and command history.
- Call Stack + Locals**: Shows a local variable `_asm_0x0` at address 0x00000000 with type void f().

The data/value of the register R1(0x0000000A) Logical Left Shifted by 1 unit =

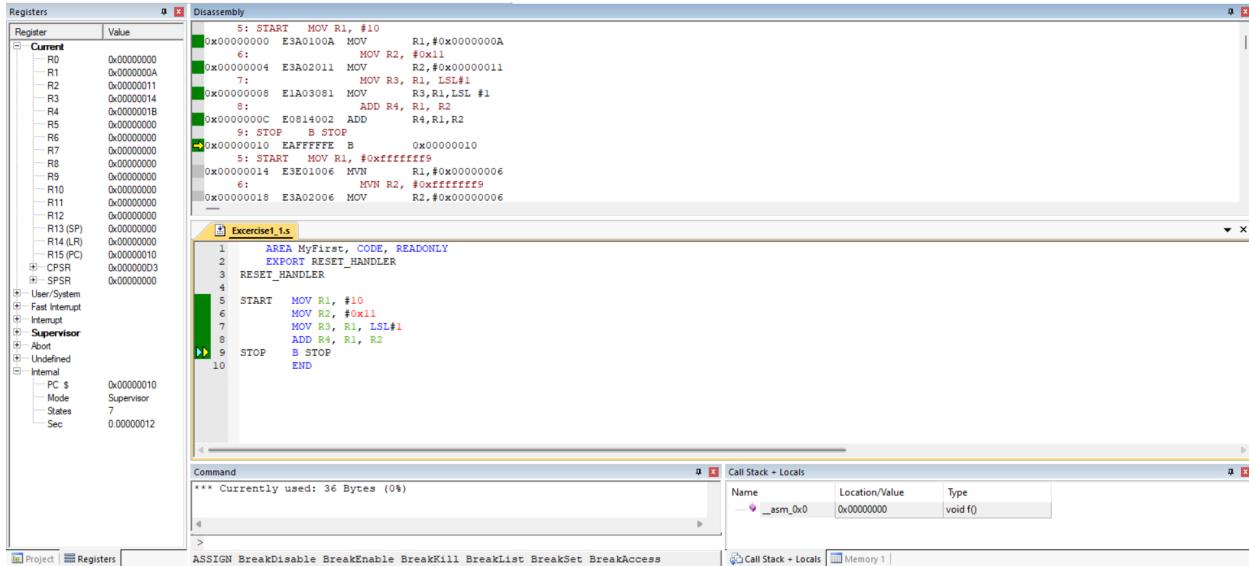
0x00000014 and then is copied into the register R3.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. R1 is highlighted with the value 0x0000000A.
- Disassembly**: Shows the assembly code for the program. Line 8 shows the instruction ADD R4, R1, R2.
- Exerciset\_1.s**: Shows the source code for the assembly file. It includes a RESET\_HANDLER section with the same ADD instruction.
- Command**: Shows the current memory usage and command history.
- Call Stack + Locals**: Shows a local variable `_asm_0x0` at address 0x00000000 with type void f().

The data/value of the register R1 (0x0000000A) is ADDED to the data present in

the register R2 = 0x0000001B , after which it is stored in the register R4.



The program stops.

### EXERCISE:

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

## Code

AREA MyFirst, CODE, READONLY

EXPORT RESET\_HANDLER

RESET\_HANDLER

START      MOV R1, #0xfffff63

              MVN R2, #0xfffff63

MOV R3, R1

STOP      B STOP

END

## Step-by-Step Execution

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current state of various registers. The CPSR register is highlighted.
- Disassembly**: Shows the assembly code being executed. The current instruction is highlighted.
- Exercise1\_1.s**: Shows the source code for the first exercise.
- Exercise1\_2.s**: Shows the source code for the second exercise, which is currently selected.
- Command**: A text input field where the user can enter commands. The text "ENI" is visible.
- Call Stack + Locals**: Shows the call stack and local variables.
- Memory**: Shows memory dump and search functions.
- Project**: Shows the project files.
- Breakpoints**: Shows the current breakpoints.
- Registers**: Shows the current state of the registers.
- Assignments**: Shows memory assignments.
- Real-Time Agent**: Shows target reset and simulation status.
- Simulation**: Shows time and capture information.
- Memory**: Shows memory dump and search functions.

Program is compiled/translated and set to debug mode.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. R1 is highlighted with the value 0xfffffff63.
- Disassembly**: Shows the assembly code for the program. Instruction at address 0x00000004 is highlighted: `MVN R2, #0xfffffff63`.
- Code Editor**: Displays the source code for Exercise1\_1.s and Exercise1\_2.s. The assembly code in both files includes the MVN instruction.
- Command Window**: Shows the command history and memory usage.
- Status Bar**: Provides real-time simulation information.

The hexadecimal value 0xfffffff63 is loaded into register R1 using MOV

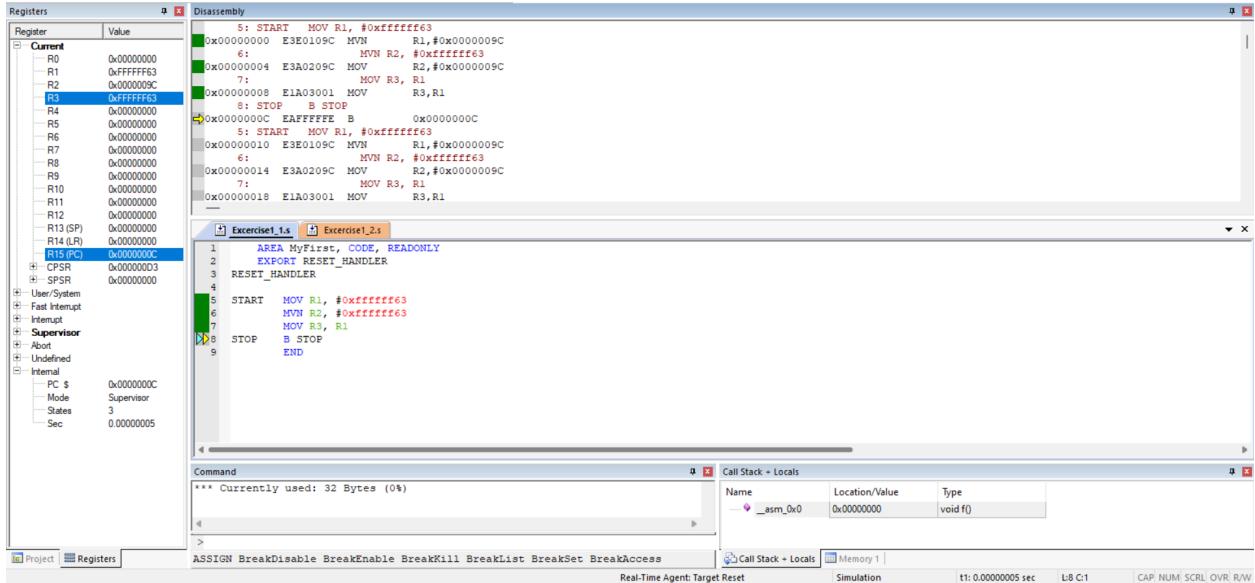
instruction.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. R2 is highlighted with the value 0xfffffff63.
- Disassembly**: Shows the assembly code for the program. Instruction at address 0x00000004 is highlighted: `MVN R2, #0xfffffff63`.
- Code Editor**: Displays the source code for Exercise1\_1.s and Exercise1\_2.s. The assembly code in both files includes the MVN instruction.
- Command Window**: Shows the command history and memory usage.
- Status Bar**: Provides real-time simulation information.

The hexadecimal value 0xfffffff63 is loaded into register R2 using MVN opcode.

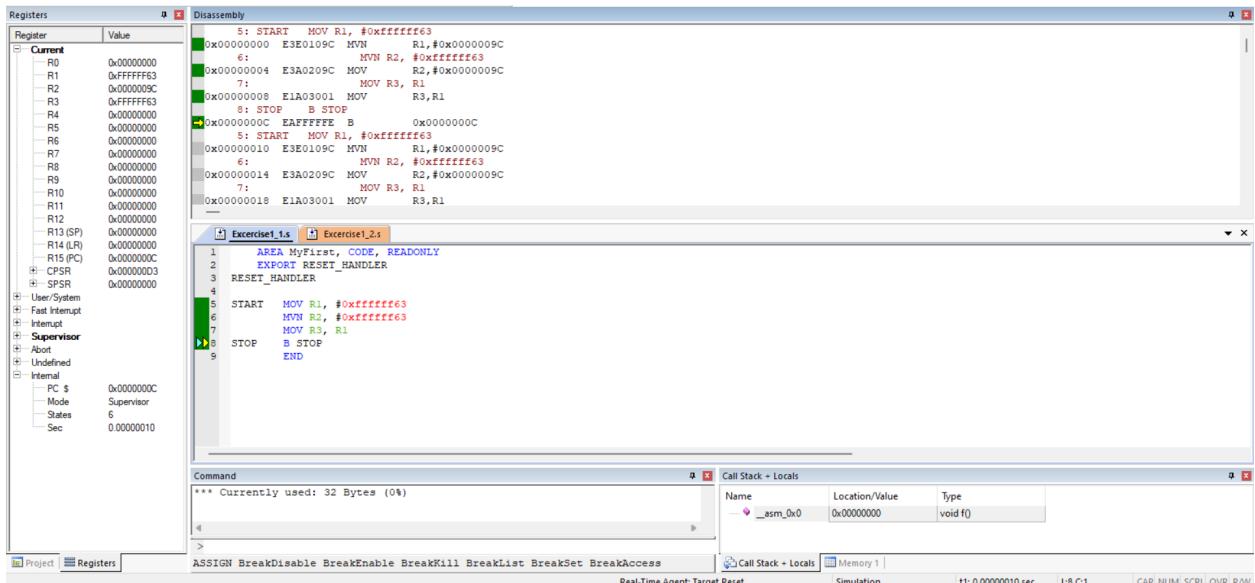
Notice that the bitwise negated value = 0xffffffff9c is stored in the register R1.



The MOV instruction can also be used to copy values from one register to another.

In the above example, the value of register R1(0xfffffff63) is copied into register R3.

R3.



The program stops.

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

## Code

```
____ AREA MyFirst, CODE, READONLY  
  
EXPORT RESET_HANDLER  
  
RESET_HANDLER  
  
START      MVN R1, #0xFFFFFC  
  
          MOV R2, #0x0000000A  
  
          MOV R3, R2, LSL #1  
  
          MOVS R4, R2, LSL #1  
  
          MOVS R3, R1, LSR #1  
  
          MOV R1, #10  
  
          MOV R2, #-10  
  
          MVNS R3, R1, ASR #1  
  
          MOVS R4, R2, ASR #1  
  
          MVNS R2, R2  
  
          MOVS R5, R2, ROR #1
```

MOVS R6, R2, RRX

STOP      B STOP

END

## Step-by-Step Execution

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. The PC register is highlighted with a blue border and has a value of 0x00000000.
- Disassembly**: Shows the assembly code for the program. The assembly code includes:
 

```

      5: START    MVN R1, #0xFFFFFFF
      6:          MOV R2, #0xF0000003
      7:          MOV R3, R2, LSL #1
      8:          MOV R4, R2, LSL #1
      9:          MOV R5, R1, LSR #1
     10:         MOV R1, #10
     11:         MOV R2, #-10
     12:         MVNS R3, R1, ASR #1
     13:         MOVNS R4, R2, ASR #1
     14:         MVNS R2,R2
     15:         MOVNS R5, R2, ROR #1
     16:         MOVNS R6, R2, RRX
     17: STOP     B STOP
     18: END
      
```
- Code Editor**: Shows the source code for "Exercise1.s".
 

```

1 AREA MyFirst, CODE, READONLY
2 EXPORT RESET_HANDLER
3 RESET_HANDLER
4
5 START    MVN R1, #0xFFFFFFF
6          MOV R2, #0xF000000A
7          MOV R3, R2, LSL #1
8          MOV R4, R2, LSL #1
9          MOV R5, R1, LSR #1
10         MOV R1, #10
11         MOV R2, #-10
12         MVNS R3, R1, ASR #1
13         MOVNS R4, R2, ASR #1
14         MVNS R2,R2
15         MOVNS R5, R2, ROR #1
16         MOVNS R6, R2, RRX
17 STOP     B STOP
18 END
      
```
- Command**: Shows the command line interface with the message: "\*\*\* Currently used: 84 Bytes (0%)".
- Call Stack + Locals**: Shows a table with one entry: `_asm_0x0 0x0000000 void f()`.
- Memory**: Shows memory dump information.

Program is compiled/translated and set to debug mode.

The screenshot shows the Keil MDK-ARM IDE interface. The Registers window on the left displays the current register values, with R1 highlighted and its value set to 0xF0000003. The Disassembly window in the center shows the assembly code for the program, with the MVN instruction at address 0x00000004. The Command window at the bottom shows the current memory usage and a call stack entry for '\_asm\_0x0'.

The hexadecimal value 0x0FFFFFFC is loaded into register R1 using MVN

instruction; therefore the value of R1 is set to 0xF0000003, which is the bitwise not of 0x0FFFFFFC.

The screenshot shows the Keil MDK-ARM IDE interface. The Registers window on the left displays the current register values, with R2 highlighted and its value set to 0x0000000A. The Disassembly window in the center shows the assembly code for the program, with the MOV instruction at address 0x00000004. The Command window at the bottom shows the current memory usage and a call stack entry for '\_asm\_0x0'.

The hexadecimal value 0x0000000A is loaded into register R2 using MOV.

The screenshot shows a debugger interface with several windows:

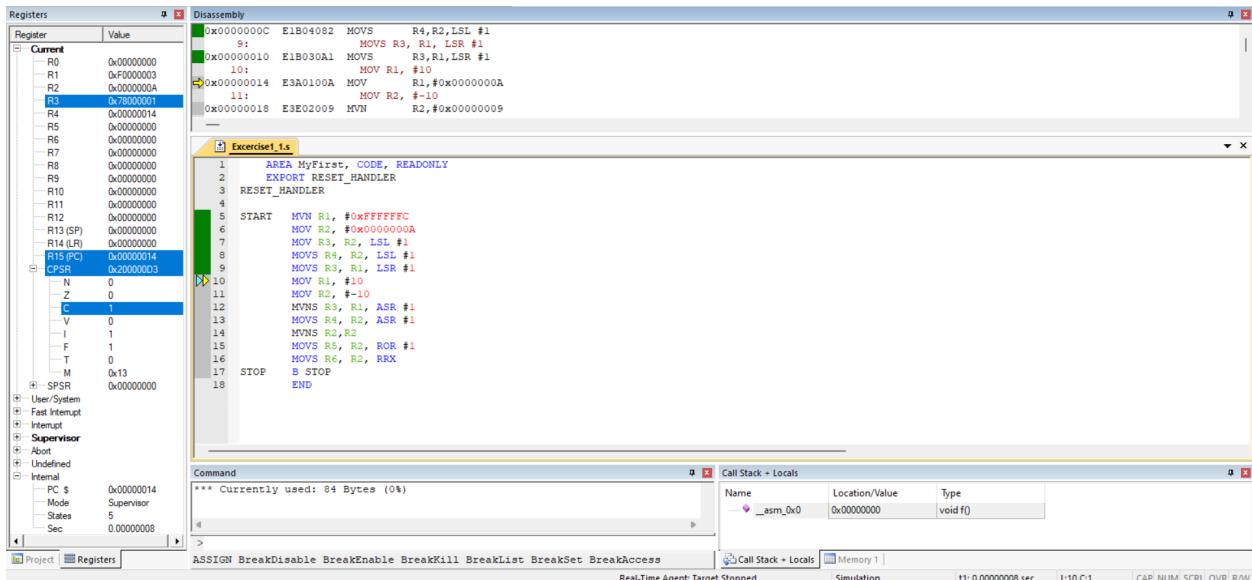
- Registers**: Shows the current register values. R3 is highlighted with a blue selection bar and has a value of 0x00000014.
- Disassembly**: Shows the assembly code starting with the instruction `MOV R1, #0x0000000A`.
- Memory**: Shows the memory dump with the address 0x00000004 containing the value 0x0000000A.
- Call Stack + Locals**: Shows a single entry: `_asm_0x0 | 0x00000000 void f()`.

The hexadecimal value 0x0000000A , which is present in the register R2 is logical left shifted by 1 unit (multiply by 2) = 0x00000014, and then stored in register R3.

The screenshot shows a debugger interface with several windows:

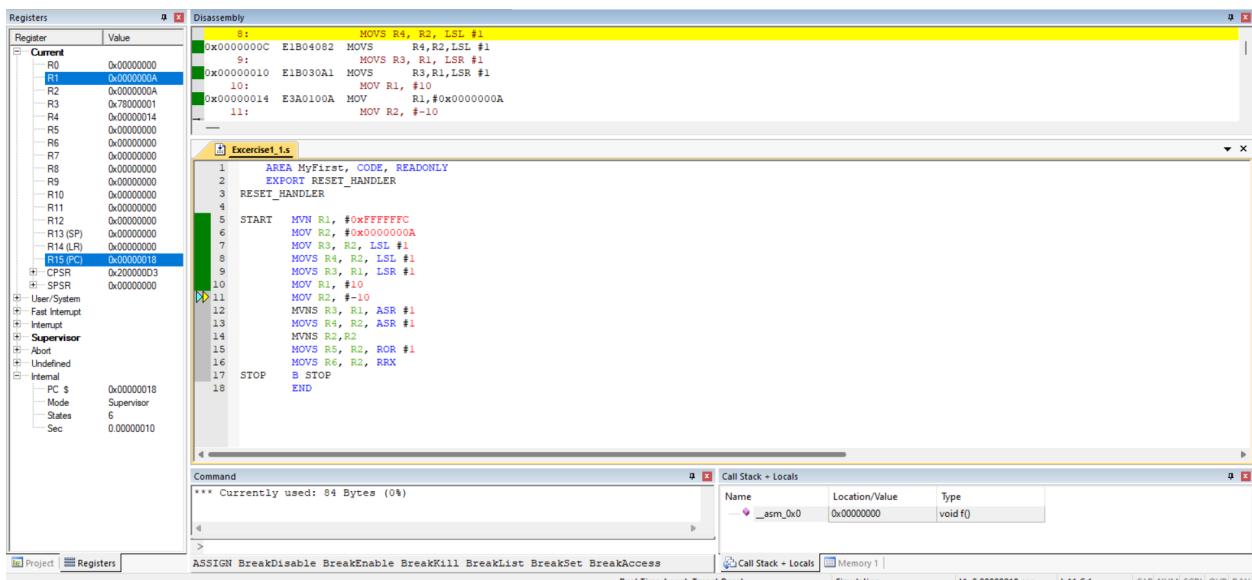
- Registers**: Shows the current register values. R4 is highlighted with a blue selection bar and has a value of 0x00000014.
- Disassembly**: Shows the assembly code starting with the instruction `MOV R1, #0x0000000A`.
- Memory**: Shows the memory dump with the address 0x00000004 containing the value 0x0000000A.
- Call Stack + Locals**: Shows a single entry: `_asm_0x0 | 0x00000000 void f()`.

We performed the same instruction however on register R4 instead of R3 and using the MOVS instead of MOV (not the CPSR does not change as there is no carry).



The hexadecimal value 0xF0000003, which is present in the register R1 is logical right shifted by 1 unit (divided by 2) = 0x78000001, and then stored in register R3.

Note the C flag (Carry) is incremented to 1 as the LSB of 0xF0000003 is one.



The decimal value 10 = 0xFFFFFFF0 is loaded into register R1 using MOV.

The screenshot shows a debugger interface with several windows:

- Registers**: Shows the current register values. R2 is highlighted with a blue selection bar and has a value of 0xFFFFFFF6.
- Disassembly**: Shows the assembly code for the current program. The instruction at address 0x00000018 is MOV R2, #-10.
- Exercise1\_1.s**: A text editor window containing the assembly source code for the program.
- Command**: A command line interface showing the memory usage: \*\*\* Currently used: 84 Bytes (0%).
- Call Stack + Locals**: A table showing a local variable: Name: \_\_asm\_0x0, Location/Value: 0x00000000, Type: void f()
- Project**: A list of project files.
- Registers**: Another view of the registers, showing R2 = 0xFFFFFFF6.
- ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess**: A toolbar with various debug options.
- Real-Time Agent**: Set to Target Reset.
- Simulation**: Set to Stop.
- Time**: t1: 0.00000012 sec.
- Log**: L12 C1.
- CAP NUM SCRL OVR R/W**: Control buttons for memory operations.

The decimal value -10 is loaded into register R2 using MOV. Hence R2 is set to 0xFFFFFFF6.

The screenshot shows a debugger interface similar to the previous one, but with different register values:

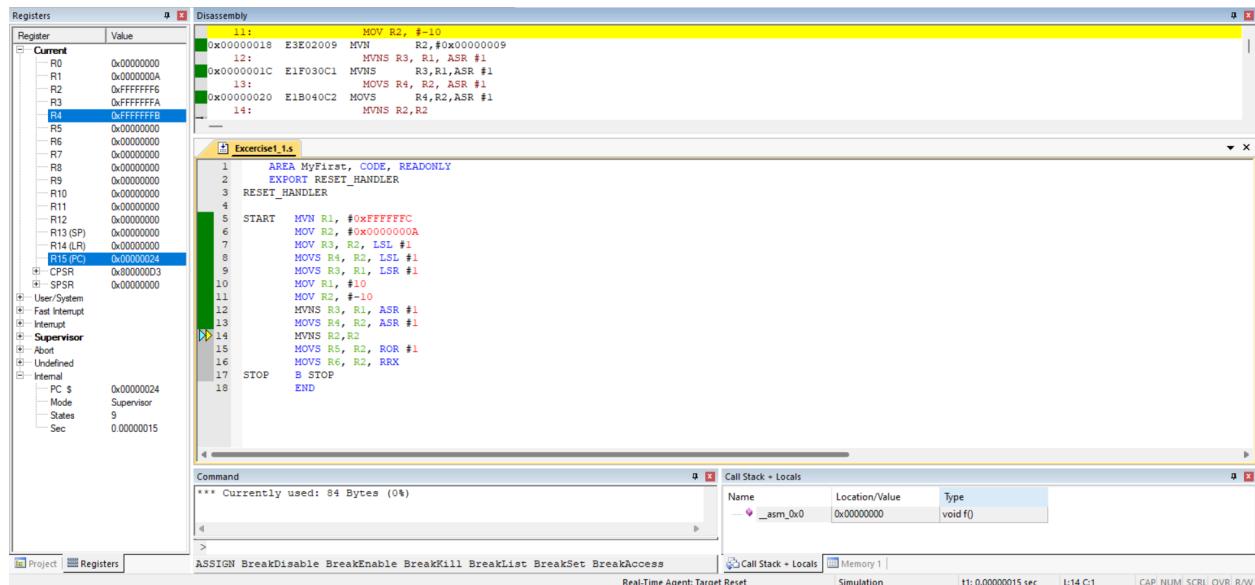
- Registers**: Shows the current register values. R2 is now 0x0000000A.
- Disassembly**: Shows the assembly code for the current program. The instruction at address 0x00000018 is MOV R2, #0x00000009.
- Exercise1\_1.s**: A text editor window containing the assembly source code for the program.
- Command**: A command line interface showing the memory usage: \*\*\* Currently used: 84 Bytes (0%).
- Call Stack + Locals**: A table showing a local variable: Name: \_\_asm\_0x0, Location/Value: 0x00000000, Type: void f()
- Project**: A list of project files.
- Registers**: Another view of the registers, showing R2 = 0x0000000A.
- ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess**: A toolbar with various debug options.
- Real-Time Agent**: Set to Target Stopped.
- Simulation**: Set to Stop.
- Time**: t1: 0.00000013 sec.
- Log**: L13 C1.
- CAP NUM SCRL OVR R/W**: Control buttons for memory operations.

The hexadecimal value 0x0000000A, which is present in the register R1 is

arithmetic right shifted by 1 unit (divided by 2) = 0x00000005, and then is bitwise

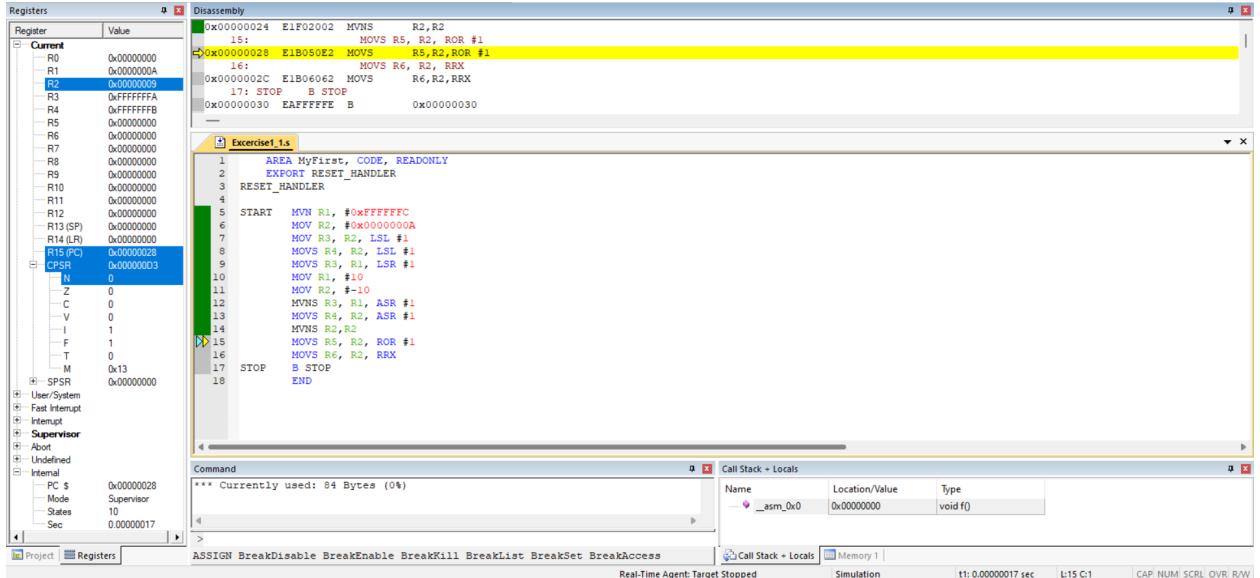
not is applied = 0xFFFFFFF6 as we are using MVNS after which we are storing it in the register R3. Note the C flag (Carry) is set to 0 as the LSB of 0x00000005 is 0, also since the output is negative the N flag (Negative) is set to 1 .

\*The ASR copies the MSB (the sign bit) into the empty slot which the LSR does not.

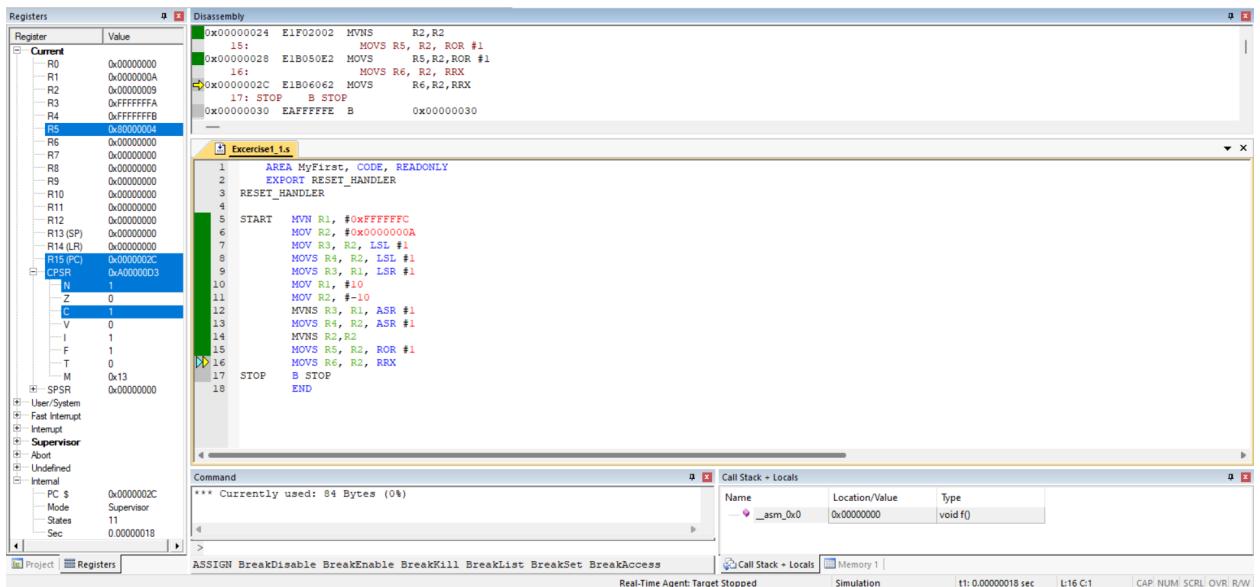


The hexadecimal value 0xFFFFFFF6, which is present in the register R2 is arithmetic right shifted by 1 unit (divided by 2) = 0xFFFFFFFFB, and then is stored in the register R4. CPSR remains the same as the carry is 0.

\*The ASR copies the MSB (the sign bit) into the empty slot which the LSR does not.



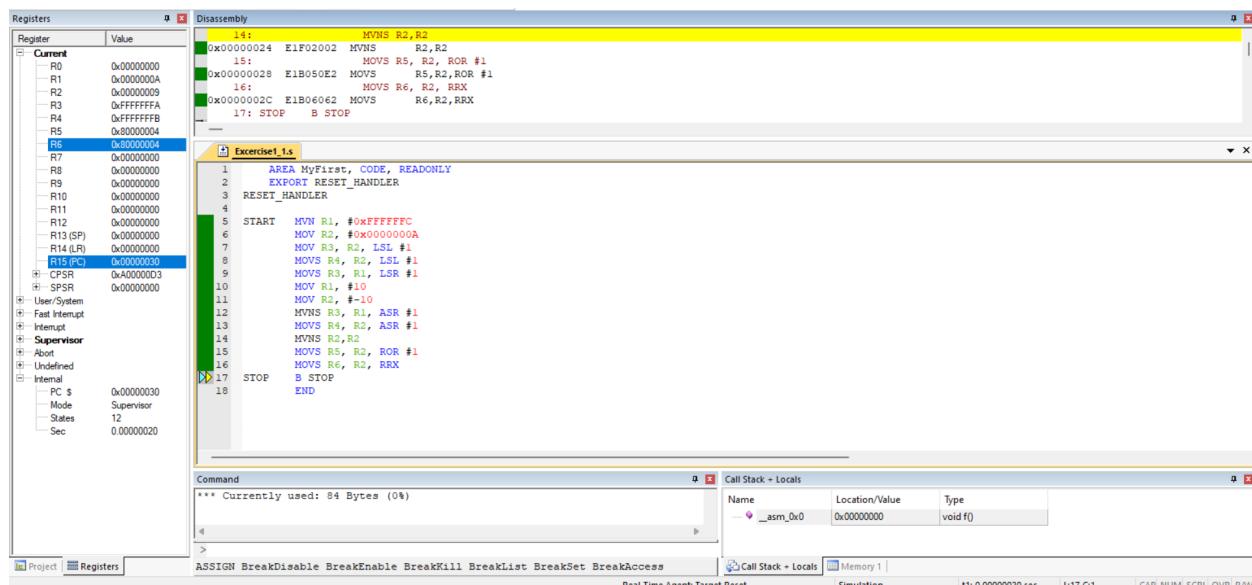
The hexadecimal value 0xFFFFFFF6, which is present in the register R2, is bitwise not as we are using MVNS instruction = 0x00000009 and stored in R2 itself. Note here the N flag (negative flag) is toggled to 0 as the result is positive.



The hexadecimal value 0x00000009, which is present in the register R2, is rotated

right by 1 unit using ROR = 0x80000004 and stored in R5. Note here the N flag (negative flag) is toggled to 1 as the result is negative and also the C flag (carry flag) is also set to 1 as LSB (of 0x00000009 )is 1 .

\*The ROR instruction gives the LSB as carry flag and n units are logically shifted by right and copied into the empty spaces.



The hexadecimal value 0x00000009, which is present in the register R2, is rotated right using RRX by 1 unit and stored in R5 = 0x80000004. Note here the N flag (negative flag) remains 1 as the result is negative and also the C flag (carry flag) remains as LSB (of 0x00000009 )is 1 .

\*The RXX is a 33 bit (carry is included as LSB of the value) rotation which is different from the ROR which is a 32 bit rotation.

The screenshot shows a debugger interface with several windows:

- Registers** window: Shows the current register values. Most registers (R0-R17) have a value of 0x00000000. The CPSR register has a value of 0x40000003, and the SPSR register has a value of 0x00000000.
- Disassembly** window: Displays assembly code from memory address 0x00000030 to 0x00000038. The code includes instructions like STOP, MOV, MVN, and MVNS, along with some comments and labels.
- Command** window: Shows the command line with the message "\*\*\* Currently used: 84 Bytes (0%)".
- Call Stack + Locals** window: Shows a single entry: a local variable named `_asm_0x0` located at 0x00000000 with a type of void f().

The program stops.