

## Lab sheet 05

**Title:** Exercises for Comparison and Loop in ARM Assembly

**Aims:**

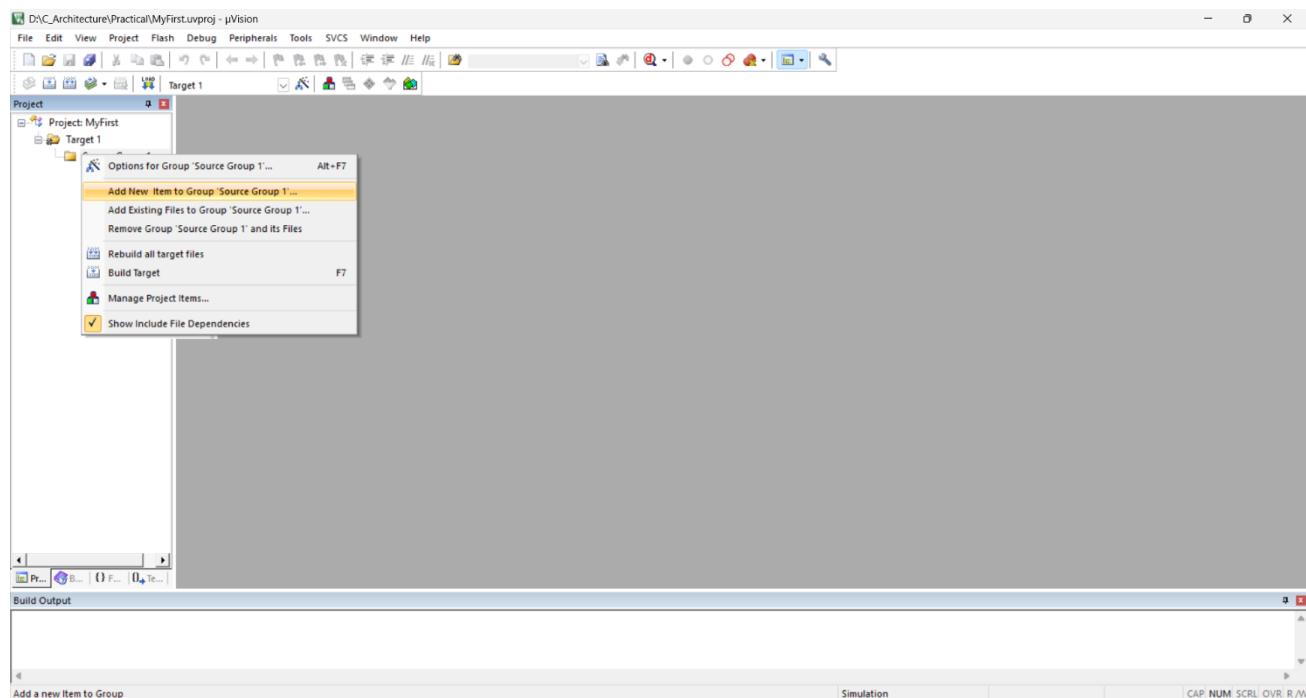
1. To understand how comparison operations are performed using ARM Assembly instructions.
2. To learn how loops are created using conditional and unconditional branch instructions.
3. To develop skills in writing, executing, and debugging loop-based and comparison- based ARM Assembly programs in Keil µVision.

**Tasks:**

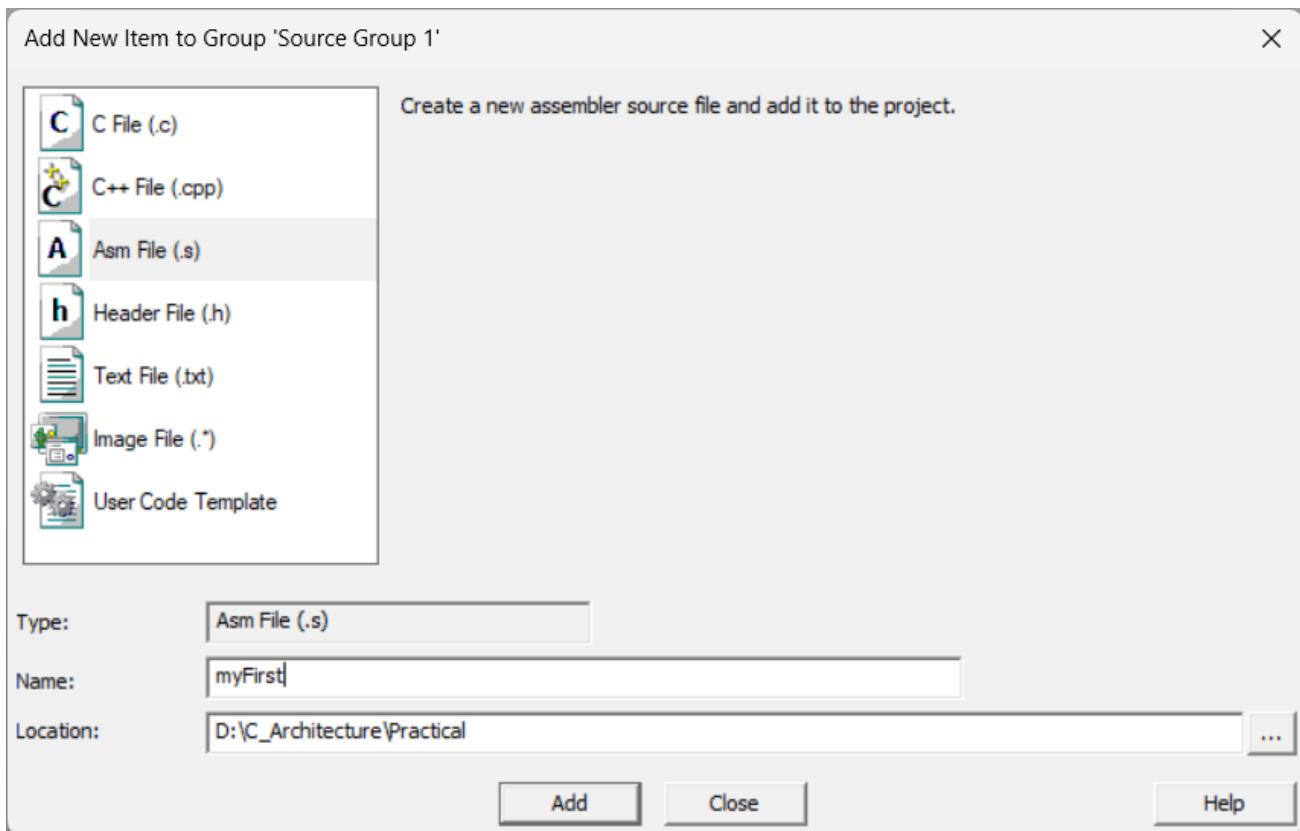
1. Identify and explain comparison and branch instructions such as CMP, BEQ, BNE, BGT, BLT, and B used in ARM Assembly for decision-making and looping.
2. Analyze the program flow by tracing each step to understand:
  - a. How Comparisons Affect Branching
  - b. How loops repeat based on conditions
  - c. How ARM uses condition flags to control program execution

**Activities**

1. Click on the + next to Target 1 to expand the tree. Right click on Source Group 1 and choose Add New Item to Group.

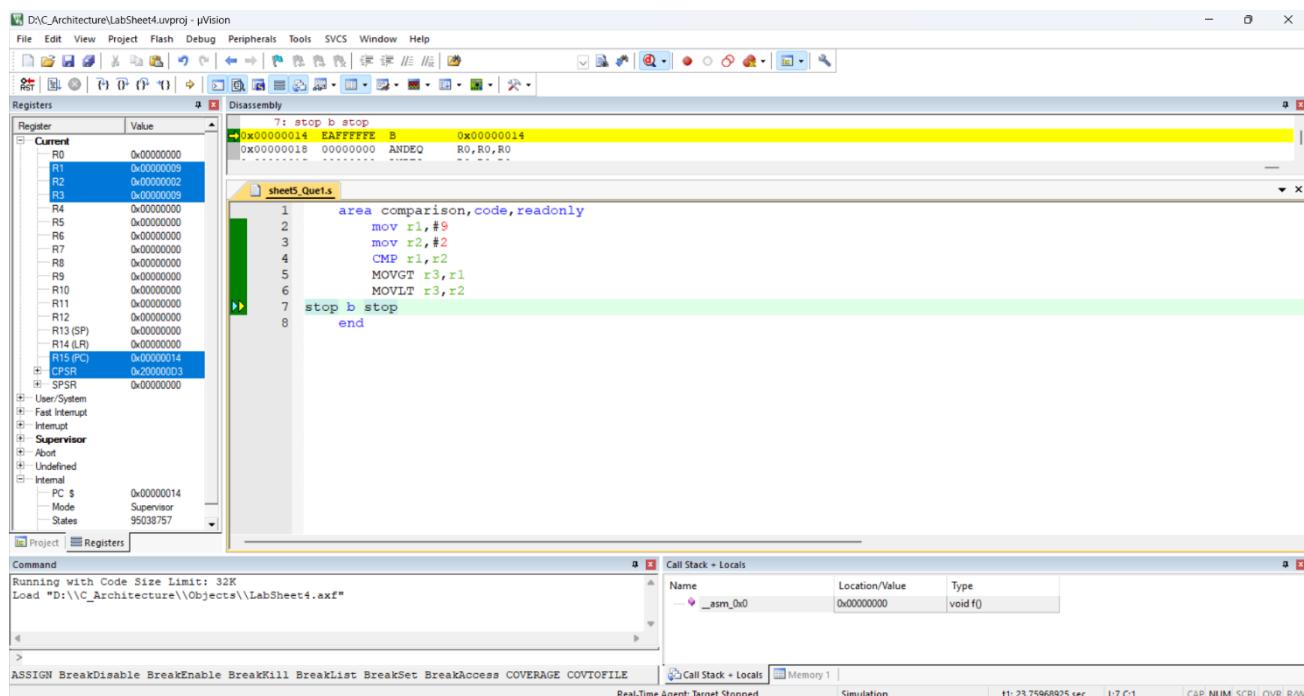


2. Select Asm File (. s) , then type program name First and Add to source.



## Exercises:

- Find the larger number between two numbers and store the output in R3 register



2. Write a program to find the factorial of a number in each way.

- Increment Hint:  $5! = 1*2*3*4*5$

The screenshot shows the µVision IDE interface with the following details:

- Registers:** Shows the current state of registers R0-R15, CPSR, and SPSR.
- Disassembly:** Displays the assembly code for the increment factorial program. The code initializes R0 to 5, R1 to 1, and R2 to 1. It then enters a loop where it multiplies R2 by R1, adds R1 to R2, and increments R1. The loop exits when R1 is greater than 5.
- Command:** Shows the command line with "Running with Code Size Limit: 32K" and "Load D:\C\_Architecture\Objects\LabSheet4.axf".
- Call Stack + Locals:** Shows a single entry in the call stack: \_asm\_0x0 at address 0x00000000.

```

area factorial_Increment,code,readonly
    mov r0,#5 ;Number to find factorial of (N)
    mov r1,#1 ;Counter (starts at 1)
    mov r2,#1 ;Results accumulator (starts at 1)

loop
    cmp r1,r0 ;compare counter with N
    bgt stop ;if counter > N, stop
    mul r3,r2,r1 ;Temp = Result * Counter
    mov r2,r3 ;Result = Temp
    add r1,r1,#1 ;Increment Counter
    b loop

stop
    mov r0,r2 ;Store final result in r0
    b stop
end

```

- Decrement Hint:  $5! = 5*4*3*2*1$

The screenshot shows the µVision IDE interface with the following details:

- Registers:** Shows the current state of registers R0-R15, CPSR, and SPSR.
- Disassembly:** Displays the assembly code for the decrement factorial program. The code initializes R0 to 5, R1 to 5, and R2 to 1. It then enters a loop where it multiplies R2 by R1, subtracts 1 from R1, and decrements R1. The loop exits when R1 is less than 1.
- Command:** Shows the command line with "Running with Code Size Limit: 32K" and "Load D:\C\_Architecture\Objects\LabSheet4.axf".
- Call Stack + Locals:** Shows a single entry in the call stack: \_asm\_0x0 at address 0x00000000.

```

area factorial_Decrement,code,readonly
    mov r0,#5 ;Number to find factorial of (N)
    mov r1,r0 ;Counter (starts at N)
    mov r2,#1 ;Results accumulator

loop
    cmp r1,#1 ;compare counter with 1
    blt stop ;if counter < 1, stop
    mul r3,r2,r1 ;Temp = Result * Counter
    mov r2,r3 ;Result = Temp
    sub r1,r1,#1 ;Decrement Counter
    b loop

stop
    mov r0,r2 ;Store final result in r0
    b stop
end

```

3. Write a program to check whether a number in R0 is divisible by 5.  
If divisible, set R1 = 1; otherwise, R1 = 0

The screenshot shows the pVision software interface for a Cortex-M processor. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The Registers pane on the left displays the current values of various registers, including R0 through R15, CPSR, and SPSR. The Disassembly pane in the center shows assembly code for a function named 'sheet5.Que3.s'. The code implements a loop that divides a value in R3 by R2, with a branch to the start of the loop if the result is not zero. The Call Stack & Locals pane at the bottom right shows a single entry for '\_asm\_0x0' with a value of 0x00000000.

4. Write a program to find the Greatest Common Divisor (GCD) of  $8_{16}$  and  $18_{16}$ .

Hint: GCD is the biggest number that can divide both numbers completely.

The screenshot shows the µVision IDE interface with the following windows:

- Registers**: Shows the current register values. The R0 register is highlighted.
- Disassembly**: Shows the assembly code for the current program. The instruction at address 0x00000024 is highlighted in yellow: `18: stop b stop`. The assembly code includes:

```
18: stop b stop
0x00000024 EAXFFFE B      0x00000024
0x00000028 00000000 ANDEQ R0,R0,R0

sheet5_Queue
1     area GCD,code,readonly
2       MOV r0,#0x08      ;first number = 0x8
3       MOV r1,#0x18      ;second number = 0x18
4   GCD_Loop
5     CMP r0,r1
6     BEQ DONE          ;if equal, GCD found
7
8     BGT R0_Greater   ;if r0 > r1
9     SUB r1,r1,r0      ;r1 = r1-r0
10    B GCD_Loop
11
12 R0_Greater
13    SUB r0,r0,r1      ;r0 = r0-r1
14    B GCD_Loop
15
16 DONE      ;GCD stored in r0 (and r1)
17
18 stop b stop
19 end
```
- Call Stack + Locals**: Shows the call stack and local variables. The variable `_asm_0x0` is listed with a value of `0x00000000`.

## **Discussion**

This lab sheet mainly discusses how comparison operations and loop control are implemented in ARM Assembly language. Through a series of guided exercises, the lab focuses on using instructions such as CMP along with conditional and unconditional branch instructions (BEQ, BNE, BGT, BLT, and B) to make decisions and repeat instructions based on conditions. By solving problems like finding the larger number, calculating factorials using increment and decrement loops, checking divisibility by 5, and finding the GCD of two numbers, the lab helps understand how condition flags control program flow, how loops are formed using branches, and how logical decision-making works at the assembly level. Overall, the lab strengthens practical skills in writing, tracing, executing, and debugging ARM assembly programs using Keil µVision, while clearly demonstrating how comparisons and loops work together in low-level programming.

## **Reference**

- Lecturer's Notes.