

## Lab sheet 05

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

#### Aims:

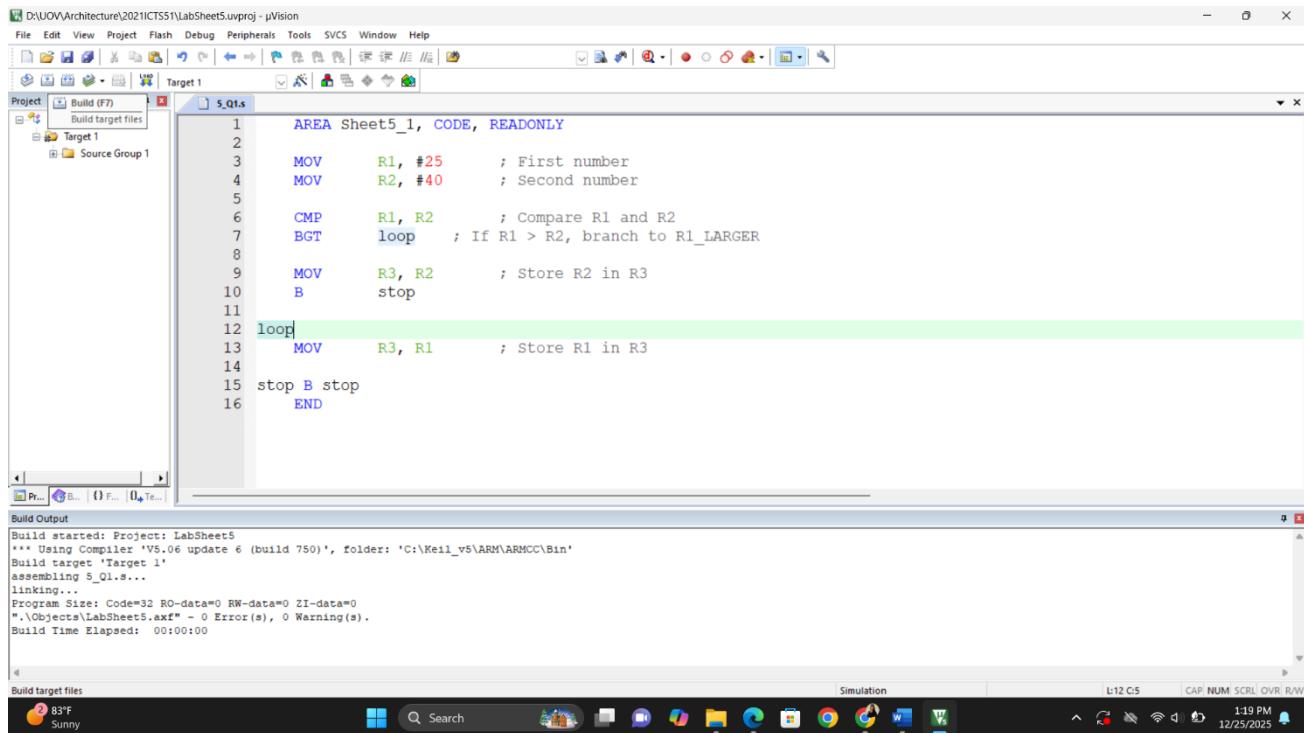
1. To understand how comparison operations are performed using ARM Assembly instructions.
2. To learn how loops are created using conditional and unconditional branch instructions.
3. To develop skills in writing, executing, and debugging loop-based and comparison- based ARM Assembly programs in Keil 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) find the larger number between two numbers and store the output in R3 register



The screenshot shows the Keil µVision IDE interface. The assembly code for the project is displayed in the main window:

```

1 AREA Sheet5_1, CODE, READONLY
2
3 MOV    R1, #25      ; First number
4 MOV    R2, #40      ; Second number
5
6 CMP    R1, R2      ; Compare R1 and R2
7 BGT   loop        ; If R1 > R2, branch to R1_LARGER
8
9 MOV    R3, R2      ; Store R2 in R3
10 B    stop         ; Stop
11
12 loop:
13    MOV   R3, R1      ; Store R1 in R3
14
15 stop B stop
16 END

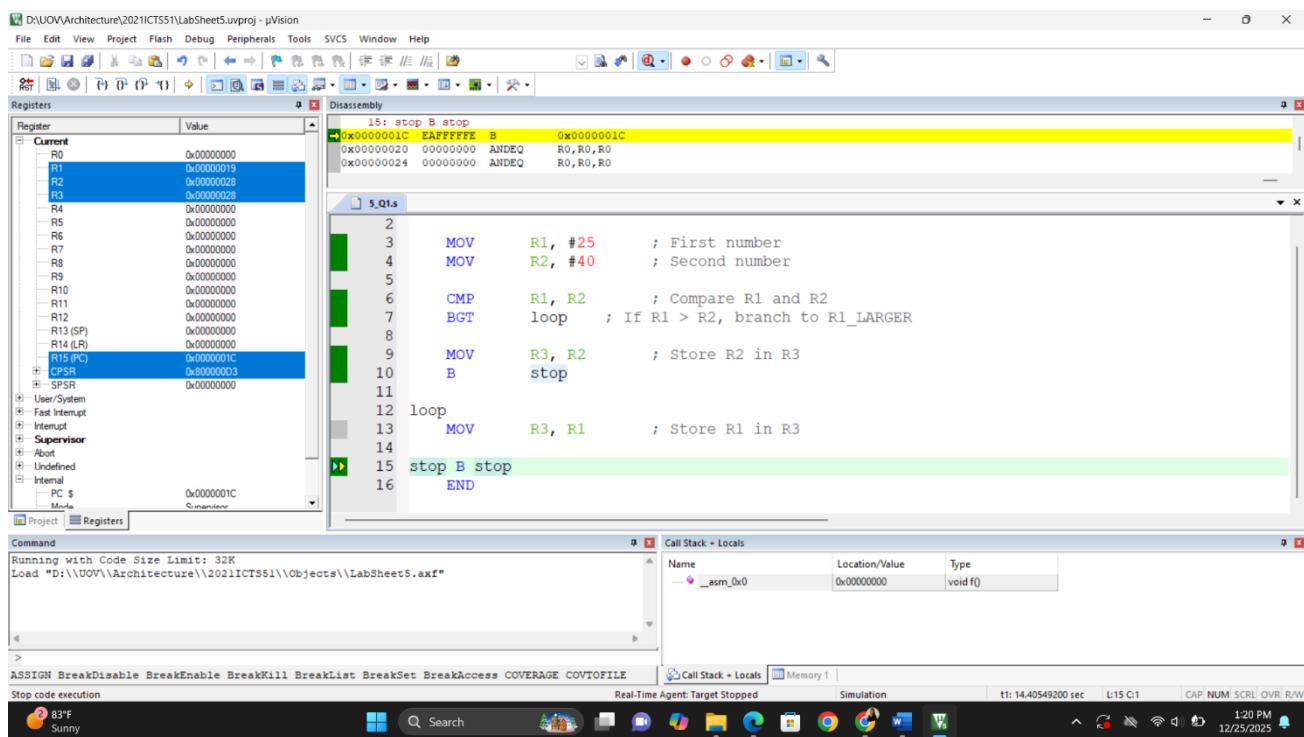
```

The build output window shows the compilation process:

```

Build started: Project: LabSheet5
*** Using Compiler 'VS.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling 5_Q1.s...
linking...
Program Size: Code=32 RO-data=0 RW-data=0 ZI-data=0
".\Objects\LabSheet5.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00

```



The screenshot shows the Keil µVision IDE interface with multiple windows open. The Registers window displays the current register values:

Register	Value
R0	0x00000000
R1	0x00000019
R2	0x00000028
R3	0x00000028
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)	0x0000001C
CPSR	0x800000D3
SPSR	0x00000000

The Disassembly window shows the assembly code again, with the instruction at address 0x0000001C highlighted.

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 Keil µVision IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar below has various icons for file operations like Open, Save, and Build. The Project window on the left shows a project named 'LabSheet5' with a target 'Target 1' containing a source group 'Source Group 1' with a file '5\_Q2\_1.s'. The main window displays the assembly code:

```
1 AREA Sheet5_2, CODE, READONLY
2
3 MOV R0, #5      ; Number to find factorial of (N)
4 MOV R1, #1      ; Counter (starts at 1)
5 MOV R2, #1      ; Result accumulator (starts at 1)
6
7 loop
8     CMP R1, R0    ; Compare counter with N
9     BGT stop      ; If counter > N, stop
10    MUL R3, R2, R1 ; Temp = Result * Counter
11    MOV R2, R3      ; Result = Temp
12    ADD R1, R1, #1   ; Increment Counter
13    B loop
14
15 stop
16    MOV R0, R2      ; Store final result in R0
17    B stop
18 END
```

Below the code, the Build Output window shows the build process:

```
Build started: Project: LabSheet5
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling 5_Q2_1.s...
linking...
Program Size: Code=44 RO-data=0 RW-data=0 ZI-data=0
".\Objects\LabSheet5.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

The status bar at the bottom right shows the date and time: 12/25/2025, 12:28 PM. The taskbar at the bottom of the screen also displays the date and time.

This screenshot shows the Keil µVision IDE with multiple windows open. The top menu bar and toolbar are identical to the previous screenshot. The Registers window on the left shows the current register values:

Register	Value
R0	0x00000078
R1	0x00000006
R2	0x00000078
R3	0x00000078
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)	0x00000028
CPSR	0x20000003
SPSR	0x00000000

The Disassembly window shows the assembly code for the factorial program. The Call Stack + Locals window at the bottom shows a single entry: `_asm_0x0` with value `0x00000000` and type `void f()`.

## ▪ Decrement

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

The screenshot shows the µVision IDE interface with the assembly code for calculating the factorial of 5. The code uses registers R0, R1, and R2. It initializes R0 to 5, R1 to 5, and R2 to 1. It then enters a loop where it compares R1 with 1. If R1 is less than or equal to 1, it stops. Otherwise, it multiplies R3 (which is R2) by R1, moves the result to R2, decrements R1, and loops back. Finally, it moves the result to R0 and stops.

```
AREA Sheet5_2_2, CODE, READONLY
MOV R0, #5      ; Number to find factorial of (N)
MOV R1, R0      ; Counter (starts at N)
MOV R2, #1      ; Result 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          ; loop

stop:
    MOV R0, R2
    B stop          ; stop

END
```

Build Output:

```
Build started: Project: LabSheet5
*** Using Compiler 'VS.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling 5_Q2_2.s...
linking...
Program Size: Code=44 RO-data=0 RW-data=0 ZI-data=0
".\Objects\LabSheet5.axf" - 0 Errors(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

This screenshot shows the µVision IDE with the assembly code for factorial calculation and the current register values. The PC register is highlighted, showing its value as 0x00000024. The registers pane shows various processor status flags like CPSR and SPSR. The assembly code is identical to the one in the previous screenshot, showing the steps to calculate  $5!$ .

Registers:

Register	Value
R0	0x00000078
R1	0x00000000
R2	0x00000078
R3	0x00000078
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)	0x00000024
CPSR	0x00000003
SPSR	0x00000000

Disassembly:

```
16: MOV R0, R2
0x00000024 E1A00002 MOV R0,R2
17: B stop
0x00000028 EAFFFFF4 B 0x00000024

1 AREA Sheet5_2_2, CODE, READONLY
2
3 MOV R0, #5      ; Number to find factorial of (N)
4 MOV R1, R0      ; Counter (starts at N)
5 MOV R2, #1      ; Result accumulator
6
7 loop:
8     CMP R1, #1      ; Compare counter with 1
9     BLT stop        ; If counter < 1, stop
10    MUL R3, R2, R1  ; Temp = Result * Counter
11    MOV R2, R3      ; Result = Temp
12    SUB R1, R1, #1  ; Decrement Counter
13    B loop          ; loop
14
15 stop:
16    MOV R0, R2
17    B stop          ; stop
18 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 µVision IDE interface with the assembly code for the program. The code is as follows:

```
1 AREA Sheet5_3, CODE, READONLY
2
3 MOV R0, #25      ; Number to check
4 MOV R2, R0      ; Copy R0 to R2 for processing
5 MOV R3, #5      ; Divisor
6
7 loop
8 CMP R2, R3      ; Compare Current Value with 5
9 BLT check       ; If Current Value < 5, go to check
10 SUB R2, R3, R3 ; Subtract 5
11 B loop         ; Repeat
12
13 check
14 CMP R2, #0      ; Check if remainder is 0
15 BEQ is_divisible ; If remainder == 0, it is divisible
16
17 MOV R1, #0      ; Else, set R1 = 0
18 B stop          ; Stop
19
20 is_divisible
21 MOV R1, #1      ; Set R1 = 1
22
23 stop B stop
24 END
```

The assembly code is highlighted in green. Below the code, the build output window shows the compilation process:

```
Build started: Project: LabSheet5
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling 5_Q3.s...
linking...
Program Size: Code=52 RO-data=0 RW-data=0 ZI-data=0
".\Objects\LabSheet5.axf" - 0 Errors), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

The system tray at the bottom indicates it's 83°F and sunny.

This screenshot shows the µVision IDE with the Registers and Disassembly windows open. The Registers window shows the current register values:

Register	Value
R0	0x00000019
R1	0x00000001
R2	0x00000000
R3	0x00000005
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)	0x00000030
+ CPSR	0x600000D3
+ SPSR	0x00000000

The Disassembly window shows the assembly code again, with the instruction at address 0x00000030 highlighted in yellow:

```
23: stop B stop
0x00000030 EAFFFFFF B 0x00000030
0x00000034 00000000 ANDEQ R0,R0,R0
0x00000038 00000000 ANDEQ R0,R0,R0
```

The assembly code is also shown in the code editor window below:

```
1 AREA Sheet5_3, CODE, READONLY
2
3 MOV R0, #25      ; Number to check
4 MOV R2, R0      ; Copy R0 to R2 for processing
5 MOV R3, #5      ; Divisor
6
7 loop
8 CMP R2, R3      ; Compare Current Value with 5
9 BLT check       ; If Current Value < 5, go to check
10 SUB R2, R3, R3 ; Subtract 5
11 B loop         ; Repeat
12
13 check
14 CMP R2, #0      ; Check if remainder is 0
15 BEQ is_divisible ; If remainder == 0, it is divisible
16
17 MOV R1, #0      ; Else, set R1 = 0
18 B stop          ; Stop
19
20 is_divisible
21 MOV R1, #1      ; Set R1 = 1
22
23 stop B stop
24 END
```

The system tray at the bottom indicates it's 83°F and sunny.

4) Write a program to find the Greatest Common Divisor (GCD) of 816 and 1816.

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

The screenshot shows the Keil uVision IDE interface. The assembly code for finding the GCD of 816 and 1816 is displayed in the main editor window. The code uses R1 and R2 registers to store the numbers, and a loop to repeatedly subtract the smaller number from the larger one until they are equal, at which point R1 contains the GCD. The code is annotated with comments explaining each step.

```
AREA Sheet5_4, CODE, READONLY
1
2
3
4 LDR R1, =816 ; First number
5 LDR R2, =1816 ; Second number
6
7 loop
8 CMP R1, R2 ; Compare the two numbers
9 BEQ stop ; If R1 == R2, GCD found
10 SUBGT R1, R1, R2 ; If R1 > R2, R1 = R1 - R2
11 SUBLT R2, R2, R1 ; If R2 > R1, R2 = R2 - R1
12 B loop ; Repeat until they are equal
13
14 stop
15 MOV R0, R1 ; Store GCD in R0
16 B stop
17
18 END
```

Build Output

```
Build started: Project: LabSheet5
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling 5_Q4.s...
linking...
Program Size: Code=40 RO-data=0 RW-data=0 ZI-data=0
".\Objects\LabSheet5.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

System tray icons include weather (85°F Partly sunny), search, taskbar, and system status.

This screenshot shows the Keil uVision IDE during the execution of the GCD program. The Registers window on the left shows the current values of the R0, R1, and R2 registers, all initialized to 0x00000000. The Disassembly window shows the instruction flow, with the current instruction being a MOV R0, R1. The assembly code window shows the same GCD algorithm as the previous screenshot. The Call Stack + Locals window at the bottom indicates that the program has stopped at a breakpoint.

Registers

Register	Value
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)	0x0000001C
+ CPSR	0x60000003
+ SPSR	0x00000000

Disassembly

```
15: MOV R0, R1 ; Store GCD in R0
+0x0000001C E1A00001 MOV R0,R1
16: B stop
+0x00000020 EAFFFFF D B 0x0000001C
```

Assembly

```
1
2
3
4 LDR R1, =816 ; First number
5 LDR R2, =1816 ; Second number
6
7 loop
8 CMP R1, R2 ; Compare the two numbers
9 BEQ stop ; If R1 == R2, GCD found
10 SUBGT R1, R1, R2 ; If R1 > R2, R1 = R1 - R2
11 SUBLT R2, R2, R1 ; If R2 > R1, R2 = R2 - R1
12 B loop ; Repeat until they are equal
13
14 stop
15 MOV R0, R1 ; Store GCD in R0
16 B stop
17
18 END
```

Call Stack + Locals

Name	Location/Value	Type
_asm_0x0	0x00000000	void f()

Command

```
Load "D:\\UOV\\Architecture\\2021ICTS51\\Objects\\LabSheet5.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVFILE
Stop code execution
```

System tray icons include weather (85°F Partly sunny), search, taskbar, and system status.

Discussion:

- 1) \*\*\*\*\*
  - Logic: Uses the CMP instruction to subtract two values internally and update processor flags.
  - Conditional Branching: The BGT (Branch if Greater Than) instruction directs the program flow based on the comparison result.
  - Storage: If the first number is larger, it jumps to a specific label to move that value to R3; otherwise, it proceeds to move the second number.
  
- 2) \*\*\*\*\*
  - Accumulation: Both methods use a register initialized to 1 to store the running product.
  - Multiplication: The MUL instruction performs the core calculation in every loop iteration.
  - Direction:
    - Increment: Starts the counter at 1 and climbs up to N.
    - Decrement: Starts the counter at \$N\$ and counts down to 1.
  - Boundary: Both loops use a condition (BGT or BLT) to stop the process once the counter exceeds the target range.
  
- 3) \*\*\*\*\*
  - Modulo via Subtraction: Since direct remainder instructions aren't always available, the program repeatedly subtracts 5 from the original number.
  - Remainder Identification: Once the value becomes less than 5, the remaining value is the "remainder."
  - Result Logic: A final comparison (CMP) checks if that remainder is exactly 0. If it is, the number is divisible, and the indicator R1 is set to 1.
  
- 4) \*\*\*\*\*
  - Euclidean Algorithm: This program uses the subtraction-based Euclidean method to find the GCD.
  - Iterative Reduction: The larger of the two numbers is replaced by the difference between the two numbers ( $R1 = R1 - R2$  or  $R2 = R2 - R1$ ).
  - Convergence: The process repeats until both registers hold the same value. That final equal value is the Greatest Common Divisor.
  - Large Constants: Because 816 and 1816 are large, the LDR pseudo-instruction is required instead of MOV to avoid "Immediate value out of range" errors.

Reference: Keil Software