ASSIGNMENT NO: 01

NAME: KAINAT ALI

SUBJECT: COAL

ENROLLMENT NO:468287

TOPIC: COAL ASSIGNMENT

Lab Manual: Computer Organization and Assembly Language (ARM)

Experiment 1: Understanding ARM Architecture

Objective:

To understand the basic components of the ARM architecture.

Task:

- Research and draw the ARM processor architecture.
- Label the following components:

- **Registers**: General-purpose registers (R0-R15), Program Counter (PC), Link Register (LR), Stack Pointer (SP), Current Program Status Register (CPSR).
- **ALU (Arithmetic Logic Unit)**: Handles arithmetic and logical operations.
- **Control Unit**: Manages the execution of instructions.
- **Pipeline Stages**: Fetch, Decode, Execute, Memory Access, and Writeback.
- **Coprocessor Interface**: Used for floating-point operations or specialized instructions.
- **Memory Management Unit (MMU)**: Handles memory accesses and translations.

Experiment 2: Basic Assembly Instructions

Objective:

To write, execute, and debug basic ARM assembly instructions.

Tasks:

Write a program to load and store data using LDR and STR:

The **LDR** (Load Register) instruction loads data from memory into a register.
The **STR** (Store Register) instruction stores data from a register into memory.

Sample Code for LDR and STR:

```
AREA Data Section, DATA
num1 DCD 5
num2 DCD 10
```

```
AREA Code Section, CODE
ENTRY
LDR R0, =num1; Load address of num1 into R0
LDR R1, [R0]; Load value from address in R0 into R1
LDR R0, =num2; Load address of num2 into R0
STR R1, [R0]; Store value of R1 into the address in
R0
```

Perform basic arithmetic operations (addition, subtraction) using ADD, SUB:

The **ADD** instruction performs addition.
The **SUB** instruction performs subtraction.

Sample Code for ADD and SUB:

```
AREA Code Section, CODE
ENTRY
MOV R0, #5; Load 5 into R0
MOV R1, #10; Load 10 into R1
ADD R2, R0, R1; R2 = R0 + R1 (15)
SUB R3, R1, R0; R3 = R1 - R0 (5)
```

Experiment 3: Conditional Execution

Objective:

To use conditional execution in ARM assembly.

Tasks:

Write a program to compare two numbers and output the larger number.

Use the **CMP** (Compare) and conditional branch instructions **BEQ** (Branch if Equal) and **BNE** (Branch if Not Equal).

Sample Code for Conditional Execution:

```
AREA Code Section, CODE
ENTRY
MOV R0, #15; Load 15 into R0
MOV R1, #10; Load 10 into R1
CMP R0, R1; Compare R0 and R1
BEQ Equal; If R0 == R1, branch to Equal
BGT Greater; If R0 > R1, branch to Greater
BLT Smaller; If R0 < R1, branch to Smaller

Equal:
MOV R2, R0; Numbers are equal, move R0 to R2
B End
Greater:
MOV R2, R0; R0 is greater, move R0 to R2
B End
Smaller:
MOV R2, R1; R1 is greater, move R1 to R2
End:
```

Implement a conditional block using CMP, BEQ, BNE:

Same as the task above with specific conditions being checked using CMP, BEQ, and BNE.

Experiment 4: Loops in Assembly

Objective:

To use loops for repetitive tasks.

Tasks:

Write a program to calculate the sum of the first N natural numbers:

Use a loop to sum the numbers from 1 to N.

Sample Code for Loop:

```
AREA Code Section, CODE
ENTRY
MOV R0, #0; Initialize sum to 0
MOV R1, #5; Set N to 5
MOV R2, #1; Initialize counter to 1
Loop:
ADD R0, R0, R2; sum = sum + counter
ADD R2, R2, #1; increment counter
CMP R2, R1; Compare counter to N
BLT Loop; Branch if counter < N
```

Implement a multiplication operation using iterative addition:

Use a loop to simulate multiplication by repeated addition.

Sample Code for Multiplication:

```
AREA Code Section, CODE
ENTRY
MOV R0, #0; Initialize result to 0
MOV R1, #4; Set first number
MOV R2, #3; Set second number (multiplier)
MOV R3, #0; Counter for second number
Loop:
```

```
ADD R0, R0, R1; result += first number
ADD R3, R3, #1; increment counter
CMP R3, R2; Compare counter to second number
BLT Loop; Loop until counter equals second number
```

Experiment 5: Arrays in Assembly

Objective:

To work with arrays in ARM assembly language.

Tasks:

Write a program to find the maximum value in an array:

Iterate through the array and track the maximum value.

Sample Code for Finding Maximum Value:

```
AREA Data Section, DATA
nums DCD 5, 12, 8, 2, 19
AREA Code Section, CODE
ENTRY
LDR R0, =nums  ; Load address of nums into R0
LDR R1, [R0]; Load first number into R1 (max)
MOV R2, #1; Set index to 1
MOV R3, #5; Set array size

Loop:
CMP R2, R3; Compare index to array size
BEQ End; If index == size, end loop
LDR R4, [R0, R2, LSL #2]; Load next element from
array
CMP R4, R1; Compare with current max
```

```
BGT DataMax; If greater, update max
B Next Iteration

Update:
MOV R1, R4; Update max with new value

Next Iteration:
ADD R2, R2, #1; Increment index
B Loop

End:
```

Sort an array using the bubble sort algorithm:

Implement the bubble sort algorithm to reorder elements.

Sample Code for Bubble Sort:

```
AREA Data Section, DATA
nums DCD 5, 12, 8, 2, 19
AREA Code Section, CODE
ENTRY
LDR R0, =nums  ; Load address of nums into R0
MOV R1, #5; Array size

Outer Loop:
MOV R2, #0; Reset inner loop counter
MOV R3, R1; Reset iteration counter

Inner Loop:
CMP R2, R3; Compare index to iteration count
BEQ Outer End; If end of array, exit inner loop
LDR R4, [R0, R2, LSL #2]; Load current element
LDR R5, [R0, R2, LSL #2 + 4]; Load next element
```

```
CMP R4, R5; Compare current and next elements
BLE Skip Swap; If current <= next, no swap
STR R5, [R0, R2, LSL #2]; Swap values
STR R4, [R0, R2, LSL #2 + 4]
Skip Swap:
ADD R2, R2, #1; Increment inner loop counter
B Inner Loop

Outer End:
SUB R1, R1, #1; Decrement outer loop counter
CMP R1, #1; Check if sorted
BGT Outer Loop; If not, continue
```

Assessment Questions

Describe the difference between RISC and CISC architectures:

RISC (Reduced Instruction Set Computing) focuses on a small set of simple instructions, while CISC (Complex Instruction Set Computing) uses a large set of complex instructions.

Explain the role of the program counter (PC) in ARM architecture:

The **PC** holds the address of the next instruction to be executed.

What is the significance of condition codes in ARM assembly?

Condition codes (N, Z, C, V) store the status of the result of the last operation (negative, zero, carry, overflow) and affect conditional execution.

How does the ARM pipeline improve performance?

The pipeline allows multiple instructions to be processed simultaneously at different stages, improving execution speed.

Compare direct and indirect addressing modes in ARM assembly:

Direct addressing directly uses a memory address, while indirect addressing uses a register that contains a memory address.

What is the difference between LDR and LDM instructions?

LDR loads a single value from memory into a register, while **LDM** (Load Multiple) loads multiple values into a set of registers.

Explain the use of the stack pointer (SP) in subroutine calls:

The **SP** points to the top of the stack, which stores return addresses, local variables, and saved registers during function calls.

How are interrupts handled in ARM architecture?

ARM uses a vector table to handle interrupts and jumps to a specific address based on the interrupt type.

What are the advantages of using thumb instructions in ARM?

Thumb instructions are more compact, reducing memory usage and improving performance on memory-constrained systems.

Write a code snippet for swapping two numbers without using a third variable:

```
MOV R0, #5; First number
MOV R1, #10; Second number
```

```
EOR R0, R1; XOR R0 and R1
EOR R1, R0; XOR R1 and R0 (now R1 = original R0)
EOR R0, R1; XOR R0 and R1 (now R0 = original R1)
```

Define the term "endianness" and its impact on memory storage in ARM:

- Endianness refers to the order in which bytes are stored. ARM can be configured as big-endian or little-endian, affecting how multi-byte data is stored and read.

How does the barrel shifter in ARM instructions work?

- The **barrel shifter** allows shifting or rotating of a value by a specified number of positions without needing multiple instructions.

Why is pipelining important in ARM processors?

- Pipelining increases the throughput of instructions by allowing multiple instructions to be processed simultaneously in different stages.

Explain how floating-point operations differ from integer operations:

- Floating-point operations handle real numbers (fractions), while integer operations deal with whole numbers.

What are the advantages of inline assembly in ARM-based C programming?

- Inline assembly allows developers to write highly optimized code for specific tasks, improving performance where C might be less efficient.