Understanding ARM Architecture

TASK: Search and draw the ARM processor architecture, labeling its components.

ARM processor architecture is a widely used design based on Reduced Instruction Set Computing (RISC). Below is an overview of the key components typically found in an ARM architecture :

Key Components:

1. Processor Core:

Executes instructions and performs arithmetic/logic operations.

Includes general-purpose registers (R0-R15), program counter (PC), stack pointer (SP), link register (LR), and Current Program Status Register (CPSR).

2. Memory Subsystem:

Contains caches (instruction and data), main memory, and a Memory Management Unit (MMU) for virtual-to-physical address translation.

Optimized for fast data access and low power consumption.

3. Interconnect (AMBA Bus):

Connects the processor to memory and peripherals.

Supports high-speed data transfer using protocols like AXI, AHB, and APB.

4. Pipeline Stages:

Typical stages include Fetch, Decode, and Execute.

Some models feature superscalar or multi-core designs for parallel processing.

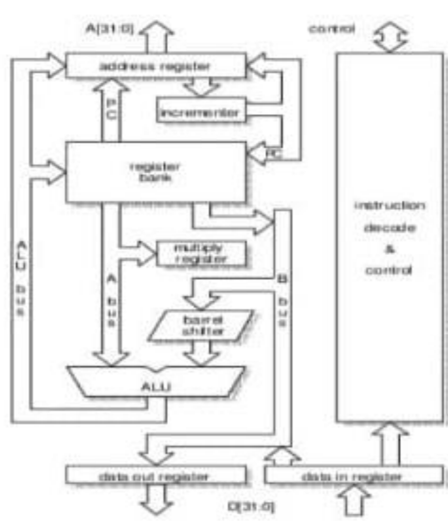
5. Power Management:

Includes dynamic voltage scaling and other techniques to optimize energy efficiency, especially critical for mobile devices.

6. Instruction Sets:

Supports ARM (32-bit), Thumb (16-bit), and ARMv8 (64-bit) sets for varied performance and code density needs.

The ARM Architecture



Basic Assembly Instructions

TASK 01: Write a program to load and store data using LDR and STR

```
    AREA MyProgram, CODE, READONLY
    ENTRY
start
    ; Initialize some values in memory
    LDR     R0, =num1       ; Load the address of num1 into R0
    LDR     R1, [R0]        ; Load the value of num1 into R1
    ADD     R1, R1, #10     ; Add 10 to the value in R1 (R1 = num1 + 10)
    LDR     R0, =num2       ; Load the address of num2 into R0
    STR     R1, [R0]        ; Store the result (num1 + 10) into num2
  ; End of program, loop indefinitely
loop
    B loop                  ; Infinite loop to end the program
    ; Data Section
num1    DCD     25          ; num1 = 25
num2    DCD     0           ; num2 will store the result (initialized to 0)
    END
```

TASK 02: Perform basic arithmetic operations (add, sub) using ADD and SUB

```
ADD R1, R2, R3
SUB R1, R2, R3

LOAD R2, 5      ; Load value 5 into R2
LOAD R3, 3      ; Load value 3 into R3

ADD R1, R2, R3  ; R1 = 5+ 3 = 8
SUB R4, R2, R3  ; 5 – 3 = 2
```

Conditional Execution

TASK 01: Write a program to compare two numbers and output the larger number.

```
    AREA CompareNumbers, CODE, READONLY
     ENTRY
start
     ; Load the first number into R0
     LDR    R0, =num1      ; Load the address of num1 into R0
     LDR    R0, [R0]       ; Load the value of num1 into R0

     ; Load the second number into R1
     LDR    R1, =num2      ; Load the address of num2 into R1
     LDR    R1, [R1]       ; Load the value of num2 into R1

     ; Compare the two numbers
     CMP    R0, R1         ; Compare R0 (num1) with R1 (num2)

     ; Branch if num1 > num2
     BGT    num1_is_larger

     ; If num2 is larger or equal, store num2 in the result
     MOV    R2, R1         ; R2 = num2
     B      done

num1_is_larger
     ; If num1 is larger, store num1 in the result
     MOV    R2, R0         ; R2 = num1
done
     ; Store the result in the result memory location
```

```
    LDR     R0, =result     ; Load the address of result
    STR     R2, [R0]        ; Store the larger number in result

    ; End of program, loop indefinitely
loop
    B loop                  ; Infinite loop to end the program

    ; Data Section
num1    DCD     25          ; num1 = 25
num2    DCD     30          ; num2 = 30
result  DCD     0           ; result will store the larger number
    END
```

TASK 02: Implement a conditional block using CMP, BEQ, BNE.

```
        AREA ConditionalBlock, CODE, READONLY
        ENTRY
    start
        ; Load the first number into R0
        LDR     R0, =num1       ; Load the address of num1 into R0
        LDR     R0, [R0]        ; Load the value of num1 into R0

        ; Load the second number into R1
        LDR     R1, =num2       ; Load the address of num2 into R1
        LDR     R1, [R1]        ; Load the value of num2 into R1

        ; Compare the two numbers
        CMP     R0, R1          ; Compare R0 (num1) with R1 (num2)

        ; If num1 == num2, branch to equal_block
        BEQ     equal_block
```

Loops in Assembly

TASK 01: Write a program to calculate the sum of the first N natural numbers.

```
        AREA SumNaturalNumbers, CODE, READONLY
        ENTRY

start
        ; Load the value of N into R0
        LDR     R0, =N          ; Load the address of N into R0
        LDR     R0, [R0]        ; Load the value of N into R0

        ; Initialize sum to 0
        MOV     R1, #0          ; R1 will hold the sum, initialized to 0

        ; Initialize counter to 1
        MOV     R2, #1          ; R2 will be the counter, starting from 1

loop
        CMP     R2, R0          ; Compare counter (R2) with N (R0)
        BGT     done            ; If counter > N, exit the loop

        ADD     R1, R1, R2      ; Add the value of counter (R2) to sum (R1)
        ADD     R2, R2, #1      ; Increment the counter (R2)

        B       loop            ; Repeat the loop

done
        ; Store the result in the result memory location
        LDR     R0, =result     ; Load the address of result
        STR     R1, [R0]        ; Store the sum (R1) in the result
```

```
        ; End of program, loop indefinitely
    loop_end
        B loop_end            ; Infinite loop to end the program

        ; Data Section
    N      DCD    10          ; N = 10 (change this value for different N)
    result  DCD    0          ; result will store the sum of the first N numbers

        END
```

TASK 02: Implement a multiplication operation using iterative addition.

```
            AREA MultiplyIteratively, CODE, READONLY
    ENTRY

start
    ; Load multiplicand (A) into R0
    LDR     R0, =multiplicand   ; Load the address of multiplicand into R0
    LDR     R0, [R0]            ; Load the value of multiplicand into R0

    ; Load multiplier (B) into R1
    LDR     R1, =multiplier     ; Load the address of multiplier into R1
    LDR     R1, [R1]            ; Load the value of multiplier into R1

    ; Initialize result to 0 (R2 will store the result)
    MOV     R2, #0              ; R2 = 0 (result of multiplication)

    ; Initialize counter to 0 (used to count iterations)
    MOV     R3, #0              ; R3 = 0 (counter)
```

```
loop
      CMP     R3, R1            ; Compare counter (R3) with multiplier (R1)
      BGE     done              ; If counter >= multiplier, end the loop

      ADD     R2, R2, R0        ; Add multiplicand (R0) to result (R2)
      ADD     R3, R3, #1        ; Increment the counter (R3)

      B       loop              ; Repeat the loop

done
      ; Store the result in memory
      LDR     R0, =result       ; Load the address of result
      STR     R2, [R0]          ; Store the result in the result memory location

      ; End of program, loop indefinitely
loop_end
      B loop_end                ; Infinite loop to end the program

      ; Data Section
multiplicand DCD    5           ; Multiplicand = 5 (change as needed)
multiplier   DCD    3           ; Multiplier = 3 (change as needed)
result       DCD    0           ; result will store the product of multiplicand and multiplier

      END
```

Arrays in Assembly

TASK 01: Write a program to find the maximum value in an array.

```
    AREA MaxArrayValue, CODE, READONLY
    ENTRY

start
    ; Load the address of the array into R0
    LDR     R0, =array          ; Load the address of array into R0

    ; Load the length of the array into R1
    LDR     R1, =array_length    ; Load the address of array_length into R1
    LDR     R1, [R1]            ; Load the value of array_length into R1

    ; Load the first element of the array into R2 (initialize maximum value)
    LDR     R2, [R0]            ; R2 = array[0] (initial max value)

    ; Initialize counter R3 to 1 (we've already processed the first element)
    MOV     R3, #1              ; Start with the second element

loop
    CMP     R3, R1              ; Compare counter R3 with array length (R1)
    BGE     done                ; If counter >= array_length, we're done

    ; Load the current array element into R4
    LDR     R4, [R0, R3, LSL #2]  ; Load array[R3] into R4 (R3 is the index)

    ; Compare current element (R4) with current max (R2)
    CMP     R4, R2              ; Compare R4 with R2 (current max)
    BGT     update_max          ; If R4 > R2, update the max
```

```
        ; Increment the counter and loop
        ADD    R3, R3, #1          ; Increment the index (R3)
        B      loop                ; Repeat the loop

    update_max
        MOV    R2, R4              ; If R4 > R2, update max (R2 = R4)
        ADD    R3, R3, #1          ; Increment the index (R3)
        B      loop                ; Continue loop

    done
        ; Store the result (maximum value) in the result memory location
        LDR    R0, =result         ; Load the address of result
        STR    R2, [R0]            ; Store the maximum value in result

        ; End of program, loop indefinitely
    loop_end
        B loop_end                 ; Infinite loop to end the program

        ; Data Section
    array        DCD   12, 34, 23, 56, 89, 45, 72, 99, 13, 64 ; Array of numbers
    array_length   DCD   10                          ; Length of the array (10 elements)
    result       DCD   0                             ; Store the maximum value here

        END
```

TASK 02: Sort an array using the bubble sort algorithm.

```
            AREA BubbleSort, CODE, READONLY
      ENTRY
start
      ; Load the address of the array into R0
```

```
    LDR     R0, =array          ; Load the address of the array into R0

    ; Load the length of the array into R1

    LDR     R1, =array_length    ; Load the address of the array length into R1

    LDR     R1, [R1]             ; Load the value of array length into R1


    ; Decrement the length to get the last index (length - 1)

    SUB     R1, R1, #1          ; R1 = array_length - 1


outer_loop

    MOV     R2, #0              ; Initialize the swapped flag to 0 (no swap)

    MOV     R3, #0              ; Initialize the index to 0 (R3 is the counter)


inner_loop

    CMP     R3, R1              ; Compare index R3 with (array_length - 1)

    BGE     outer_done          ; If R3 >= array_length - 1, exit inner loop


    ; Load array[R3] into R4 and array[R3+1] into R5

    LDR     R4, [R0, R3, LSL #2]  ; R4 = array[R3]

    LDR     R5, [R0, R3, LSL #2 + 4] ; R5 = array[R3+1]


    ; Compare array[R3] with array[R3+1]

    CMP     R4, R5              ; Compare R4 (array[R3]) with R5 (array[R3+1])

    BGT     swap_elements       ; If array[R3] > array[R3+1], swap


    ; Move to the next index

    ADD     R3, R3, #1          ; Increment the index (R3)

    B       inner_loop          ; Repeat the inner loop


swap_elements

    ; Swap array[R3] and array[R3+1]

    STR     R5, [R0, R3, LSL #2]  ; Store array[R3+1] in array[R3]
```

```
    STR     R4, [R0, R3, LSL #2 + 4] ; Store array[R3] in array[R3+1]

    ; Set the swapped flag to 1
    MOV     R2, #1              ; Set swapped flag to 1
    ADD     R3, R3, #1          ; Increment the index (R3)
    B       inner_loop          ; Repeat the inner loop

outer_done
    ; If no elements were swapped, the array is sorted
    CMP     R2, #0              ; Check if swapped flag is 0
    BEQ     done                ; If no swaps, exit

    ; Decrement the length (R1) and repeat the outer loop
    SUB     R1, R1, #1          ; Decrease the array length
    B       outer_loop          ; Repeat the outer loop
done
    ; End of sorting, the array is now sorted
    ; Store the sorted array in memory
    LDR     R0, =result         ; Load the address of result
    STR     R0, [R0]            ; Store the sorted array in result

    ; Infinite loop to end the program
loop_end
    B loop_end                  ; Infinite loop to halt the program

    ; Data Section
array           DCD    12, 34, 23, 56, 89, 45, 72, 99, 13, 64 ; Array of numbers
array_length    DCD    10                        ; Length of the array (10 elements)
result          DCD    0                         ; Placeholder to store the result (sorted array)

    END
```