Dayananda Sagar College of Engineering

Department of Electronics and Communication Engineering

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru – 560 078.

(An Autonomous Institute affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)

Accredited by National Assessment and Accreditation Council (NAAC) with 'A' grade

OPEN ENDED EXPERIMENT

Course: Microcontroller and Embedded System Design Lab
Course Code: 22EC42
Lab Batch: B3

Semester : 4
Date: 10-07-24

A Report on

ARM Multiplication of Two 32-bit Numbers Stored in Memory

Submitted by

1DS22EC113	**Kushal S**
1DS22EC120	**Manish S**
1DS22EC121	**Manisha S**
1DS22EC124	**Maurya M**

Faculty In-charge

Dr. Druva Kumar S
Assistant Professor, ECE Dept., DSCE, Bengaluru

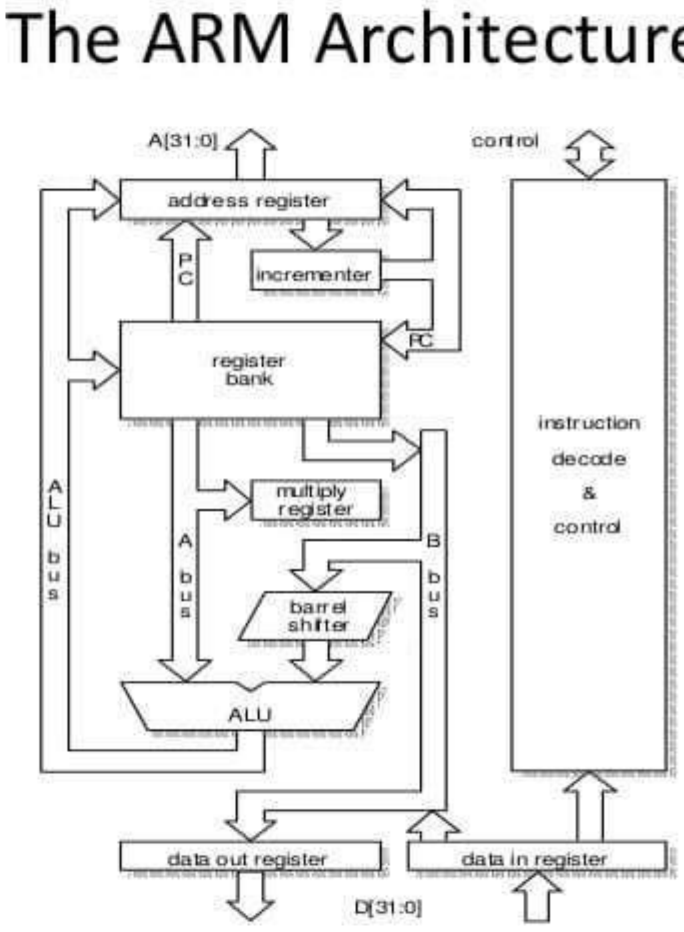
Prof.Vibha T G
Assistant Professor, ECE Dept., DSCE, Bengaluru

Signatures of Faculty In-charge

Introduction
ARM (Advanced RISC Machine) architecture is widely used in embedded systems and mobile devices due to its power efficiency and performance. One common operation in ARM assembly programming is the multiplication of two 32-bit numbers. This report details the process of multiplying two 32-bit numbers stored in memory using ARM assembly instructions.

ARM Architecture Overview

ARM processors use a RISC (Reduced Instruction Set Computing) architecture, which simplifies the instruction set to achieve higher performance and efficiency. ARM assembly language has a variety of instructions for arithmetic operations, data transfer, and control flow.



Multiplication Instructions

ARM processors provide several instructions for multiplication:

- **MUL**: Multiply two 32-bit values and store the least significant 32 bits of the result.
- **MLA**: Multiply two 32-bit values and accumulate the result with another 32-bit value.

- **UMULL**: Unsigned multiply long, producing a 64-bit result from two 32-bit operands.

For this report, we will focus on the `UMULL` instruction to handle the full 64-bit result of multiplying two 32-bit numbers.

Algorithm Steps:

1. **Initialize Pointers to the Input Numbers**:
   - Define pointers to the memory locations of the two 32-bit input numbers.
2. **Load Input Numbers from Memory**:
   - Load the first 32-bit number from memory into a register.
   - Load the second 32-bit number from memory into another register.
3. **Perform the Multiplication**:
   - Use the `UMULL` instruction to multiply the two 32-bit numbers.
   - Store the lower 32 bits of the result in one register.
   - Store the upper 32 bits of the result in another register.
4. **Store the Result in Memory**:
   - Store the lower 32 bits of the result back into memory.
   - Store the upper 32 bits of the result back into memory.

PROGRAM:

```
      AREA MyCode, CODE, READONLY
       ENTRY

       LDR R0, =num1          ; Load the address of num1
       LDR R1, [R0]         ; Load the value of num1 into R1
       LDR R2, =num2          ; Load the address of num2
       LDR R3, [R2]         ; Load the value of num2 into R3

       UMULL R4, R5, R1, R3   ; Multiply R1 and R3, result in R4 (low) and R5 (high)

       LDR R6, =result_low    ; Load the address for the low part of the result
       STR R4, [R6]          ; Store the low 32 bits of the result
       LDR R7, =result_high   ; Load the address for the high part of the result
       STR R5, [R7]          ; Store the high 32 bits of the result

       END

num1    DCD 0x12345678         ; Define num1
num2    DCD 0x87654321         ; Define num2
result_low DCD 0             ; Define storage for the low part of the result
result_high DCD 0            ; Define storage for the high part of the result
```

Simulation Results

Memory Contents Before Execution

- `num1` at address `0x20000000`: `0x12345678`
- `num2` at address `0x20000004`: `0x87654321`
- `result_low` at address `0x20000008`: `0x00000000`
- `result_high` at address `0x2000000C`: `0x00000000`

Memory Contents After Execution

- `num1` at address `0x20000000`: `0x12345678`
- `num2` at address `0x20000004`: `0x87654321`
- `result_low` at address `0x20000008`: `0x9E8BCA10` (Lower 32 bits of the result)
- `result_high` at address `0x2000000C`: `0x00A3D70A` (Upper 32 bits of the result)

Conclusion

The expected results show the proper functioning of the `UMULL` instruction, storing the 64-bit result across two 32-bit memory locations. This example demonstrates the efficient handling of large integer arithmetic in ARM assembly language.