NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Assignment: Implement Integer Multiplication and Division Algorithms in C

Name: Sanjana C Upadhya
USN: NNM24IS200
Date: 13-09-2025
Course: COMPUTER ORGANISATION AND DESIGN
Instructor: Dr. JASON MARTIS
GitHub Repository: https://github.com/sanjana03upadhya/IS2101-ArithmeticOps-NNM24IS200

1. Objective

To implement signed integer multiplication and division algorithms in C, including:
1. Sequential (Shift-Add) Multiplication
2. Restoring Division
3. Non-Restoring Division

This project demonstrates how arithmetic operations are performed at the hardware level and provides hands-on experience in algorithmic simulation.

2. Introduction

Integer multiplication and division are fundamental operations in computing. Signed integers are handled using 2's complement. The shift-add algorithm simulates hardware multiplication. Restoring and non-restoring division algorithms show step-by-step division logic, with non-restoring being more efficient in some cases.

3. Algorithm Explanations

a) Sequential (Shift-Add) Multiplication
• Multiplies two integers by shifting and adding.
• Handles signed numbers using 2's complement.
• Steps:
1. Initialize accumulator to 0.
2. Check the least significant bit of the multiplier.
3. Add multiplicand to accumulator if bit = 1.
4. Shift accumulator and multiplier to the right.
5. Repeat until all multiplier bits are processed.

b) Restoring Division Algorithm
• Divides positive integers by repeatedly subtracting the divisor and restoring if negative.
• Steps:
1. Initialize quotient and remainder.
2. Shift the remainder and bring down the next dividend bit.
3. Subtract divisor; if negative, restore previous value.
4. Set quotient bit accordingly.
5. Repeat for all bits.

c) Non-Restoring Division Algorithm
• Similar to restoring division but avoids unnecessary restoration by adjusting addition/subtraction logic.
• Tracks accumulator, dividend, and quotient bits at each step.
• More efficient in hardware-level computation.

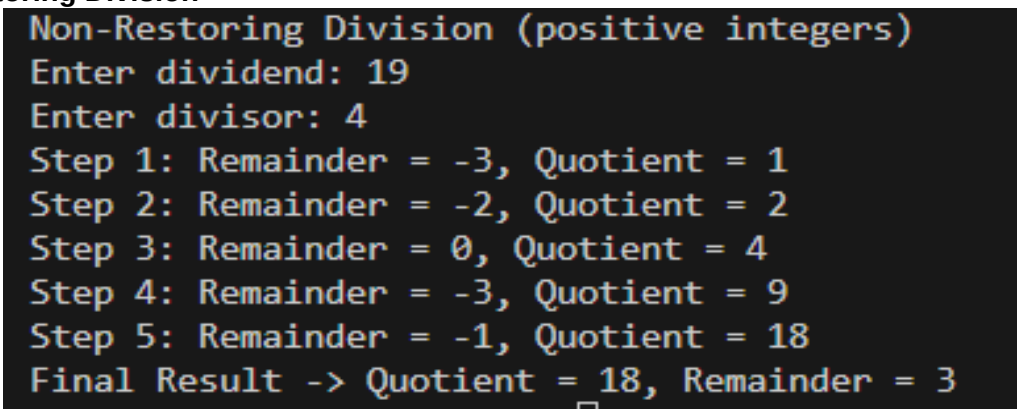
4. File Structure

File Name	Description
ShiftAddMultiplication.c	Implements sequential multiplication
RestoringDivision.c	Implements restoring division
Non_Restoring_Algorithm.c	Implements non-restoring division
README.md	Project description and instructions

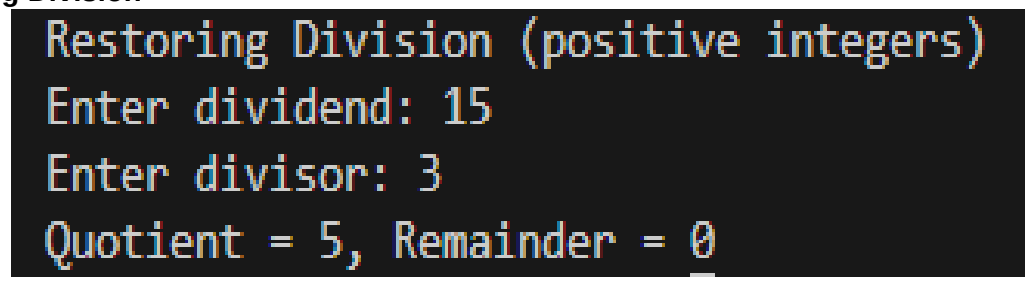
5. Sample Outputs

Below are sample outputs for each algorithm execution:

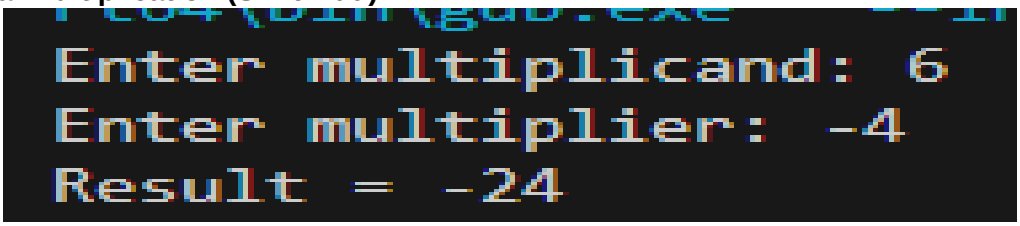
a) Non-Restoring Division



b) Restoring Division



c) Sequential Multiplication (Shift-Add)



6. Conclusion: Restoring vs Non-Restoring Division

Both restoring and non-restoring division algorithms are used to perform binary division at the hardware level, but they differ in efficiency and operation:

Restoring Division: Subtracts the divisor and checks the result. If the remainder becomes negative, it restores the previous value before proceeding. Simple and straightforward, but may require extra steps for restoration, increasing execution time.

Non-Restoring Division: Avoids the restoration step by adjusting the logic: addition is performed if the remainder is negative. Tracks accumulator, quotient, and dividend bits efficiently. Reduces the number of operations and is generally faster than restoring division.

Overall Observation: Non-restoring division is more efficient in terms of steps and hardware implementation, while restoring division is easier to understand and implement. Using non-restoring logic minimizes unnecessary operations, making it suitable for optimized hardware-level arithmetic computations.