Experiment No. 8
Implement Restoring algorithm using c-programming
Name: Shravani Sandeep Raut
Roll Number: 48
Date of Performance:
Date of Submission:

Aim: To implement Restoring division algorithm using c-programming.

Objective -

1. To understand the working of Restoring division algorithm.
2. To understand how to implement Restoring division algorithm using c-programming.

Theory:

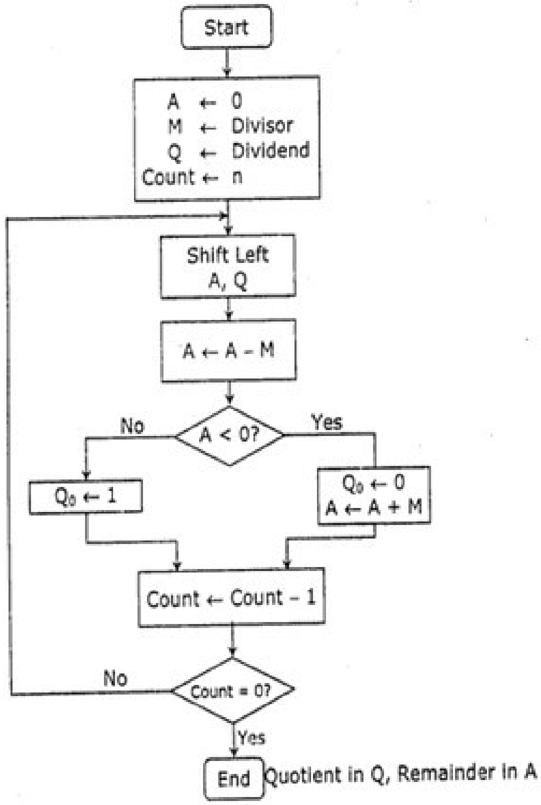
1) The divisor is placed in M register, the dividend placed in Q register.

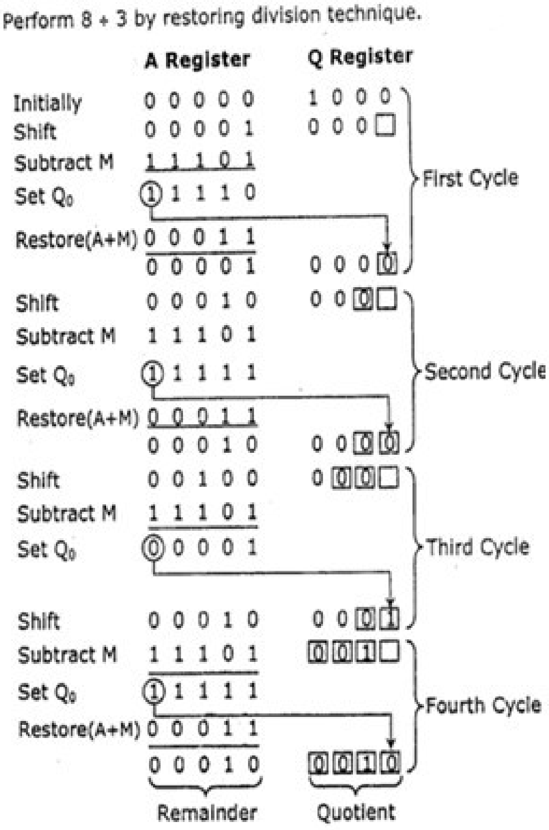
2) At every step, the A and Q registers together are shifted to the left by 1-bit

3) M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q0 set to 1-bit. Otherwise, Q0 gets a 0 bit and M must be added back to A to restore the previous value.

4) The count is then decremented and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

Flowchart





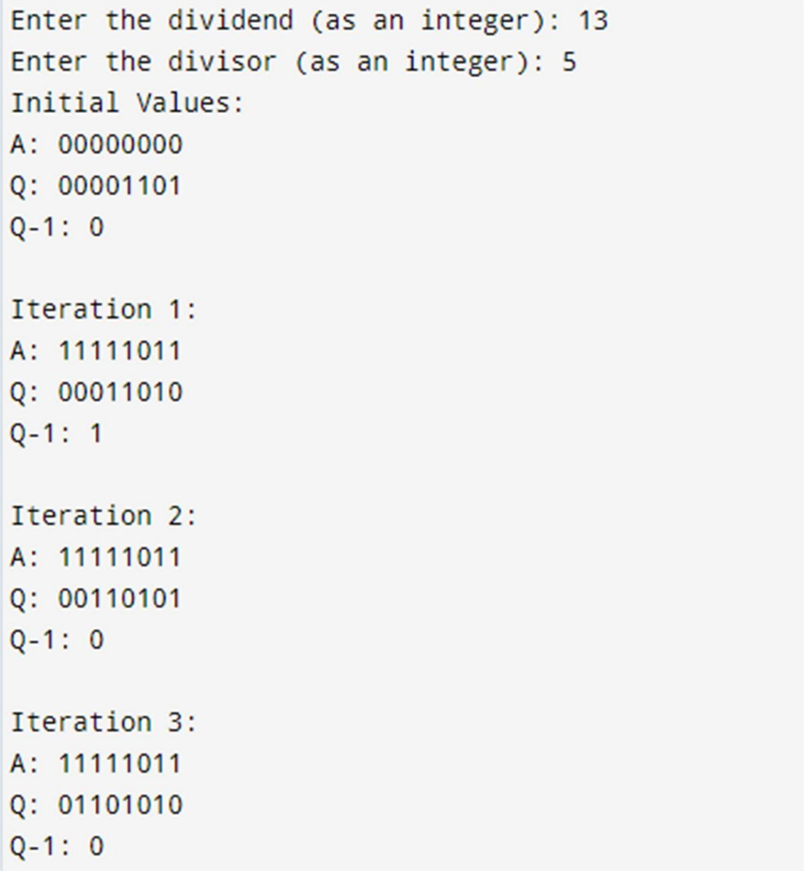
Perform 8 ÷ 3 by restoring division technique.

	A Register	Q Register	
Initially	0 0 0 0 0	1 0 0 0	First Cycle
Shift	0 0 0 0 1	0 0 0 □	
Subtract M	1 1 1 0 1		
Set Q₀	1 1 1 1 0		
Restore(A+M)	0 0 0 1 1		
	0 0 0 0 1	0 0 0 0	
Shift	0 0 0 1 0	0 0 0 □	Second Cycle
Subtract M	1 1 1 0 1		
Set Q₀	1 1 1 1 1		
Restore(A+M)	0 0 0 1 1		
	0 0 0 1 0	0 0 0 0	
Shift	0 0 1 0 0	0 0 0 □	Third Cycle
Subtract M	1 1 1 0 1		
Set Q₀	0 0 0 0 1		
Shift	0 0 0 1 0	0 0 0 1	
Subtract M	1 1 1 0 1	0 0 1 □	Fourth Cycle
Set Q₀	1 1 1 1 1		
Restore(A+M)	0 0 0 1 1		
	0 0 0 1 0	0 0 1 0	
	Remainder	Quotient	

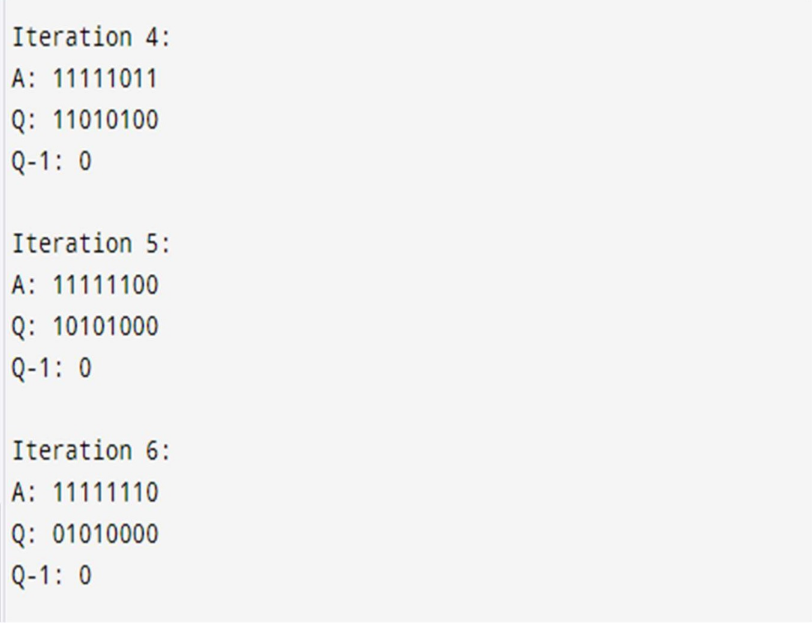
Program-

```
#include <stdio.h>
#include <string.h>

// Define the number of bits for the division
#define N 8

// Function prototypes
void restoringDivision(int dividend, int divisor);
void printBinary(int num);

int main() {
    int dividend, divisor;

    // Input the dividend and divisor
    printf("Enter the dividend (as an integer): ");
    scanf("%d", &dividend);
    printf("Enter the divisor (as an integer): ");
    scanf("%d", &divisor);

    // Perform Restoring Division Algorithm
    restoringDivision(dividend, divisor);

    return 0;
}

// Function to perform Restoring Division
void restoringDivision(int dividend, int divisor)
    {int A = 0;                    // Accumulator
    int Q = dividend;                 // Dividend
    int M = divisor;                 // Divisor
    int Q_1 = 0;                    // Q-1 (initialized to 0)
    int n = N;                     // Number of bits
    int divisor_shifted = M << (n - 1); // Initial divisor shifted left
    int quotient = 0;               // To store the quotient

    // Initialize A and Q
    A = A & ((1 << n) - 1);          // Mask to keep only N bits
    Q = Q & ((1 << n) - 1);          // Mask to keep only N bits

    printf("Initial Values:\n");
    printf("A: ");
    printBinary(A);  printf("Q:
    "); printBinary(Q);
    printf("Q-1: %d\n", Q_1);
    // Division process
    for (int i = 0; i < n; i++)
        { printf("\nIteration %d:\n", i + 1);

        // Shift left A and Q
        A = (A << 1) | ((Q >> (n - 1)) & 1);
        Q = (Q << 1) | Q_1;

        // Subtract divisor if A >= 0 after shift
        if (A >= 0) {
            A = A - divisor;
            Q_1 = 1;
        } else {
            Q_1 = 0;
            A = A + divisor; // Restore A
        }

        // Print the values after the shift
        printf("A: ");
        printBinary(A);
        printf("Q: ");
        printBinary(Q);
        printf("Q-1: %d\n", Q_1);
    }

    // The result is in Q (quotient) and A (remainder)
    printf("\nFinal Result:\n");
    printf("Quotient (Q): ");
    printBinary(Q);
    printf("\nRemainder (A): ");
    printBinary(A);
    printf("\n");
}

// Function to print the binary representation of a number
void printBinary(int num) {
    for (int i = (N - 1); i >= 0; i--) {
        printf("%d", (num >> i) & 1);
        if (i == 0) printf("\n"); // Newline at the end
    }
}
```

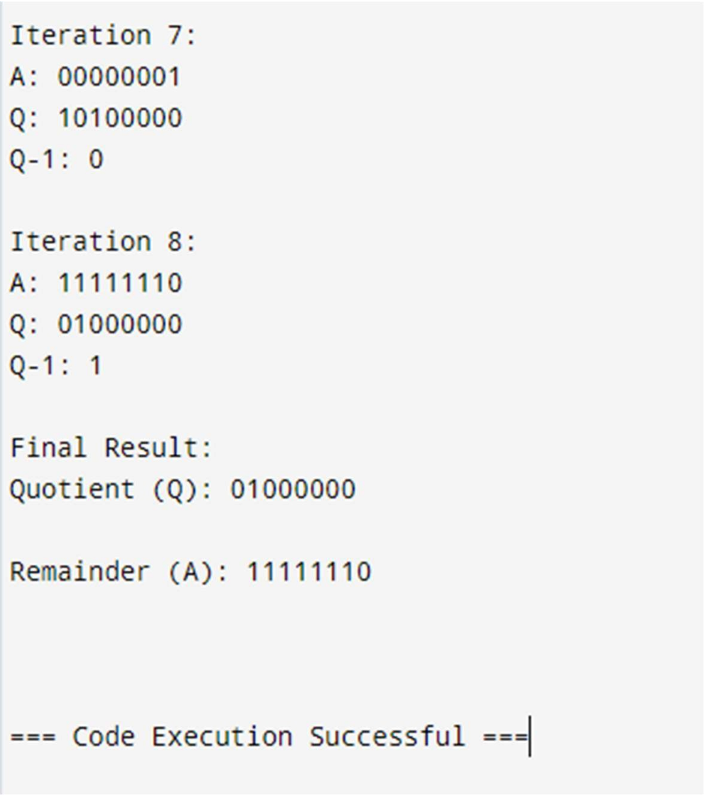
Output -



```
Enter the dividend (as an integer): 13
Enter the divisor (as an integer): 5
Initial Values:
A: 00000000
Q: 00001101
Q-1: 0

Iteration 1:
A: 11111011
Q: 00011010
Q-1: 1

Iteration 2:
A: 11111011
Q: 00110101
Q-1: 0

Iteration 3:
A: 11111011
Q: 01101010
Q-1: 0
```



```
Iteration 4:
A: 11111011
Q: 11010100
Q-1: 0

Iteration 5:
A: 11111100
Q: 10101000
Q-1: 0

Iteration 6:
A: 11111110
Q: 01010000
Q-1: 0
```



```
Iteration 7:
A: 00000001
Q: 10100000
Q-1: 0

Iteration 8:
A: 11111110
Q: 01000000
Q-1: 1

Final Result:
Quotient (Q): 01000000

Remainder (A): 11111110



=== Code Execution Successful ===
```

Conclusion - I conclude that I have understood the working of the Restoring division algorithm and how to implement Restoring division algorithm using c-programming.