



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Experiment No. 8
Implement Restoring algorithm using c-programming
Name: Patil Pranjal Keshav
Roll Number: 45
Date of Performance:
Date of Submission:



Objective -

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

- 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.

```

graph TD
    Start([Start]) --> Init[A ← 0  
M ← Divisor  
Q ← Dividend  
Count ← n]
    Init --> Shift[Shift Left  
A, Q]
    Shift --> Subtract[A ← A - M]
    Subtract --> IsAZero{A < 0?}
    IsAZero -- No --> SetQ0_1[Q₀ ← 1]
    IsAZero -- Yes --> SetQ0_0AndAdd[Q₀ ← 0  
A ← A + M]
    SetQ0_1 --> DecCount[Count ← Count - 1]
    SetQ0_0AndAdd --> DecCount
    DecCount --> IsCountZero{Count = 0?}
    IsCountZero -- No --> Shift
    IsCountZero -- Yes --> End([End])
    
```

Quotient in Q, Remainder in A

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Program-

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

```
void binaryPrint(int num, int bits) {  
    for (int i = bits - 1; i >= 0; i--) {  
        printf("%d", (num >> i) & 1);  
    }  
}
```

```
void restoringDivision(int dividend, int divisor) {  
    int quotient = 0;  
    int remainder = 0;  
    int bits = 5; // Adjust bits for binary representation (5 bits for max dividend 31)
```

```
    // Shift left the quotient and remainder for the number of bits in dividend  
    for (int i = bits - 1; i >= 0; i--) {  
        // Left shift the remainder and add the next bit of the dividend  
        remainder = (remainder << 1) | ((dividend >> i) & 1);  
        quotient <<= 1; // Shift left quotient
```

```
        // Subtract divisor from remainder  
        if (remainder >= divisor) {  
            remainder -= divisor;  
            quotient |= 1; // Set the least significant bit of the quotient  
        }  
    }
```

```
    // Print the results  
    printf("Quotient: ");  
    binaryPrint(quotient, bits);  
    printf(" Remainder: ");  
    binaryPrint(remainder, bits);  
    printf("\n");  
}
```

```
int main() {  
    int dividend, divisor;
```

```
    // Input from user  
    printf("Enter the Number (dividend and divisor separated by space): ");
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
scanf("%d %d", &dividend, &divisor);  
  
    restoringDivision(dividend, divisor);  
  
    return 0;  
}
```

Output -

Enter the Number (dividend and divisor separated by space): 15 7
Quotient: 00010 Remainder: 00001

Conclusion -

The Restoring Division algorithm is an effective method for performing division in binary arithmetic. It simulates the long division process, repeatedly subtracting the divisor from a portion of the dividend while managing the remainder.