



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Experiment No. 9

Implement Non-Restoring algorithm using c-programming

Name:- Sumit Patel

Roll no:- 40

Date of Performance: 19-09-2024

Date of Submission:

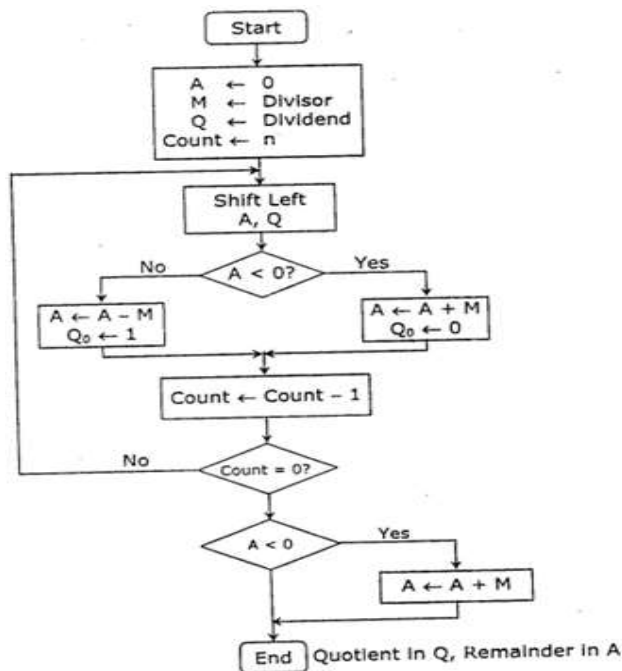
Aim - To implement Non-Restoring division algorithm using c-programming.

Objective -

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

Theory:

In each cycle content of the register, A is first shifted and then the divisor is added or subtracted with the content of register A depending upon the sign of A. In this, there is no need of restoring, but if the remainder is negative then there is a need of restoring the remainder. This is the faster algorithm of division.



Perform $8 \div 3$ by non-restoring division technique.

	A Register	Q Register	
Initially	0 0 0 0 0	1 0 0 0	
Shift	0 0 0 0 1	0 0 0 □	
Subtract	1 1 1 0 1		
Set Q ₀	① 1 1 1 0	0 0 0 ①	First Cycle
Shift	1 1 1 0 0	0 0 ① □	
Add	0 0 0 1 1		
Set Q ₀	① 1 1 1 1	0 0 ① ①	Second Cycle
Shift	1 1 1 1 0	0 ① ① □	
Add	0 0 0 1 1		
Set Q ₀	① 0 0 0 1	0 0 ① ①	Third Cycle
Shift	0 0 0 1 0	0 ① ① □	
Subtract	1 1 1 0 1		
Set Q ₀	① 1 1 1 1	0 0 ① ①	Fourth Cycle
Add	1 1 1 1 1		
	0 0 0 1 1		
	0 0 0 1 0		
			Quotient
			Remainder



Program -

```
#include <stdio.h>

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

int main() {
    int M, Q, A = 0, count;
    int n;

    printf("Enter the divisor (M): ");
    scanf("%d", &M);
    printf("Enter the dividend (Q): ");
    scanf("%d", &Q);
    printf("Enter the number of bits: ");
    scanf("%d", &n);

    count = n;

    printf("\nInitial values:\n");
    printf("A: ");
    binaryPrint(A, n);
    printf("Q: ");
    binaryPrint(Q, n);
    printf("M: ");
    binaryPrint(M, n);
    printf("\n");

    while (count > 0) {
        A = (A << 1) | ((Q >> (n - 1)) & 1);
        Q = Q << 1;

        printf("After left shift:\n");
        printf("A: ");
        binaryPrint(A, n);
        printf("Q: ");
        binaryPrint(Q, n);
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
    if (A >= 0) {
        A = A - M;
        printf("After subtraction (A >= 0):\n");
    } else {
        A = A + M;
        printf("After addition (A < 0):\n");
    }

    printf("A: ");
    binaryPrint(A, n);

    if (A >= 0) {
        Q = Q | 1;
    } else {
        Q = Q & ~(1);
    }

    printf("After updating Q0:\n");
    printf("A: ");
    binaryPrint(A, n);
    printf("Q: ");
    binaryPrint(Q, n);
    printf("\n");

    count--;
}

if (A < 0) {
    A = A + M;
    printf("Final correction (if A < 0, add M to A):\n");
    printf("A: ");
    binaryPrint(A, n);
}

printf("\nFinal quotient (Q): ");
binaryPrint(Q, n);
printf("Final remainder (A): ");
binaryPrint(A, n);
return 0;
}
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Output:

Enter the divisor (M): 8
Enter the dividend (Q): 3
Enter the number of bits: 4

Initial values:

A: 0000
Q: 0011
M: 1000

After left shift:

A: 0000
Q: 0110
After subtraction ($A \geq 0$):
A: 1000
After updating Q0:
A: 1000
Q: 0110

After left shift:

A: 0000
Q: 1100
After addition ($A < 0$):
A: 1000
After updating Q0:
A: 1000
Q: 1100

After left shift:

A: 0001
Q: 1000
After addition ($A < 0$):
A: 1001
After updating Q0:
A: 1001
Q: 1000

After left shift:

A: 0011
Q: 0000
After addition ($A < 0$):
A: 1011
After updating Q0:
A: 1011
Q: 0000

Final correction (if $A < 0$, add M to A):



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

A: 0011

Final quotient (Q): 0000

Final remainder (A): 0011

Screenshot -

```
Enter the divisor (M): 8
Enter the dividend (Q): 3
Enter the number of bits: 4

Initial values:
A: 0000
Q: 0011
M: 1000

After left shift:
A: 0000
Q: 0110
After subtraction (A >= 0):
A: 1000
After updating Q0:
A: 1000
Q: 0110

After left shift:
A: 0000
Q: 1100
After addition (A < 0):
A: 1000
After updating Q0:
A: 1000
Q: 1100

After left shift:
A: 0001
Q: 1000
After addition (A < 0):
A: 1001
After updating Q0:
A: 1001
Q: 1000

After left shift:
A: 0011
Q: 0000
After addition (A < 0):
A: 1011
After updating Q0:
A: 1011
Q: 0000

Final correction (if A < 0, add M to A):
A: 0011

Final quotient (Q): 0000
Final remainder (A): 0011
```



Conclusion -

In conclusion, the Non-Restoring Division algorithm offers a more efficient approach to binary division by eliminating the need to restore the remainder after each step. Instead of restoring the remainder when it becomes negative, the algorithm alternates between subtracting or adding the divisor based on the sign of the remainder after each shift operation. This reduces the number of operations compared to the Restoring Division method, as fewer corrective steps are needed. Additionally, a final adjustment is applied only if the remainder is negative after all iterations. This makes the Non-Restoring Division particularly useful for hardware implementations, where speed and simplicity are critical, while still delivering accurate results for both the quotient and remainder.