Experiment No. 8
Implement Restoring algorithm using c-programming
Name:
Roll Number:
Date of Performance: 19-09-2024
Date of Submission:

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

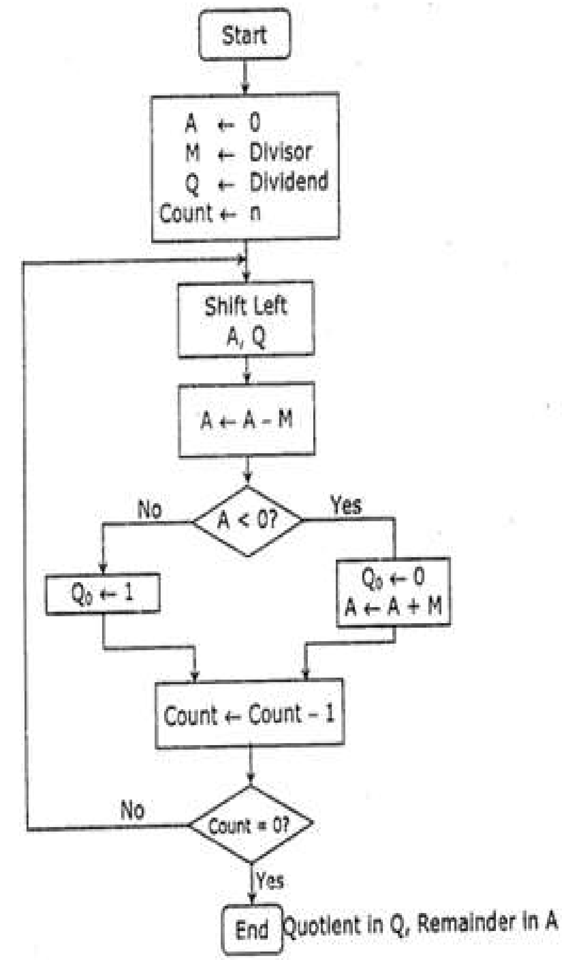
Objective -

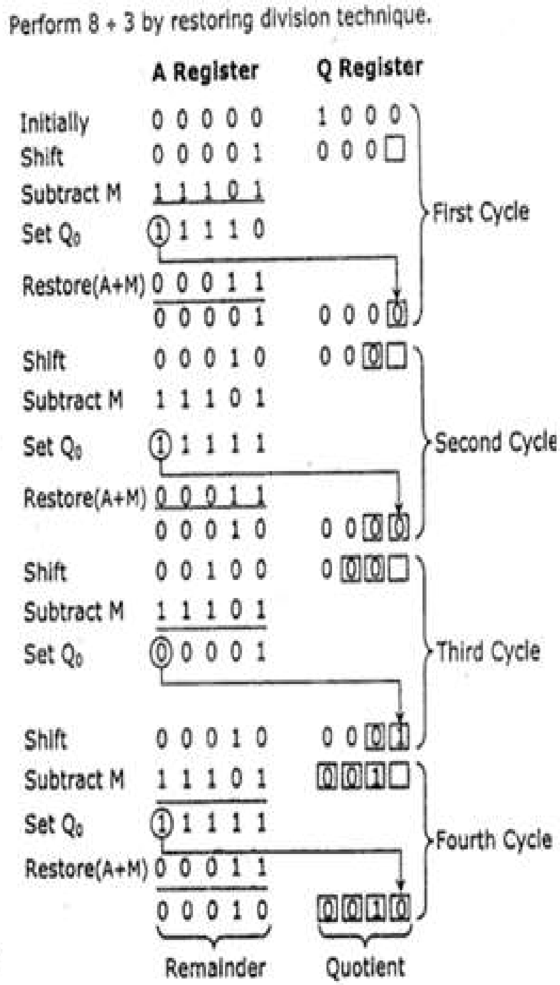
1. To understand the working of Restoring division algorithm.
2. To understand how to implement Restoring division algorithm using c-programming.

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





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);

    A = A - M;

    printf("After subtraction:\n");
    printf("A: ");
    binaryPrint(A, n);

    if (A < 0) {
        A = A + M;
        Q = Q & ~(1);
    } else {
        Q = Q | 1;
    }

    printf("After restore (if needed):\n");
    printf("A: ");
    binaryPrint(A, n);
    printf("Q: ");
    binaryPrint(Q, n);
    printf("\n");

    count--;
```

```
    }

    printf("Final quotient (Q): ");
    binaryPrint(Q, n);
    printf("Final remainder (A): ");
    binaryPrint(A, n);

    return 0;
}
```

Output -

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

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

After left shift:
A: 0001
Q: 0000
After subtraction:
A: 1110
After restore (if needed):
A: 0001
Q: 0000

After left shift:
A: 0010
Q: 0000
After subtraction:
A: 1111
After restore (if needed):
A: 0010
Q: 0000

After left shift:
A: 0100
Q: 0000
After subtraction:
A: 0001
After restore (if needed):
A: 0001
```

Q: 0001

After left shift:
A: 0010
Q: 0010
After subtraction:
A: 1111
After restore (if needed):
A: 0010
Q: 0010

Final quotient (Q): 0010
Final remainder (A): 0010

Screenshot -

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

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

After left shift:
A: 0001
Q: 0000
After subtraction:
A: 1110
After restore (if needed):
A: 0001
Q: 0000

After left shift:
A: 0010
Q: 0000
After subtraction:
A: 1111
After restore (if needed):
A: 0010
Q: 0000

After left shift:
A: 0100
Q: 0000
After subtraction:
A: 0001
After restore (if needed):
A: 0001
Q: 0001

After left shift:
A: 0010
Q: 0010
After subtraction:
A: 1111
After restore (if needed):
A: 0010
Q: 0010

Final quotient (Q): 0010
Final remainder (A): 0010
```

Conclusion -

The Restoring Division algorithm efficiently performs binary division by iteratively shifting, subtracting, and restoring the divisor and dividend. It accurately determines the quotient and remainder after a fixed number of steps corresponding to the bit-length of the divisor. This method is well-suited for hardware implementation in digital systems due to its straightforward control flow and minimal resource requirements. While it may be slower compared to non-restoring methods, its simplicity and precision make it valuable in various computational applications. The algorithm effectively demonstrates the fundamentals of binary arithmetic and logical operations.