Batch: D 3 Roll No.: 16010123294
Experiment / assignment / tutorial No. 03
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

TITLE : To study and implement Restoring method of division

AIM : The basis of algorithm is based on paper and pencil approach and the operation involves repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

Expected OUTCOME of Experiment: (Mention CO /CO's attained here)

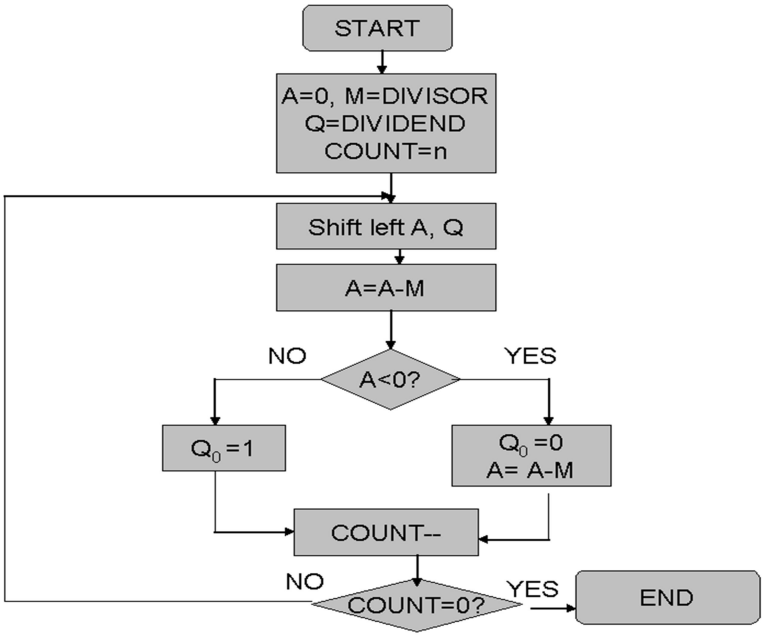
Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
2. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

The Restoring algorithm works with any combination of positive and negative numbers

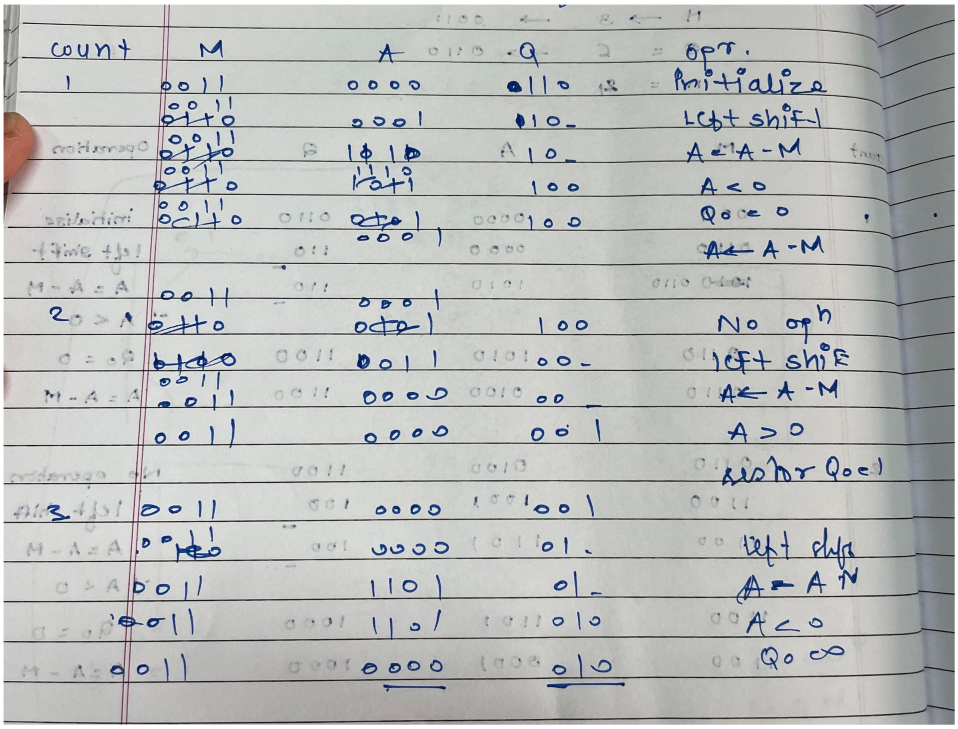
Flowchart for Restoring of Division:



Design Steps:

1. Start
2. Initialize A=0, M=Divisor, Q=Dividend and count=n (no of bits)
3. Left shift A, Q
4. If MSB of A and M are same
5. Then A=A-M
6. Else A=A+M
7. If MSB of previous A and present A are same
8. Q_0=0 & store present A
9. Else Q_0=0 & restore previous A
10. Decrement count.
11. If count=0 go to 11
12. Else go to 3
13. STOP

Example: - (Handwritten solved problem needs to be uploaded):-



CODE:-

```
#include <stdio.h>

#include <stdlib.h>


void decimalToBinary(int n, int bits, int* binary) {

   for (int i = bits - 1; i >= 0; i--) {

      binary[i] = n % 2;

      n /= 2;

   }
```

```
}

void displayBinary(int* binary, int bits) {
    for (int i = 0; i < bits; i++) {
        printf("%d", binary[i]);
    }
}

int binaryToDecimal(int* binary, int bits) {
    int decimal = 0;
    for (int i = 0; i < bits; i++) {
        decimal = (decimal << 1) | binary[i];
    }
    return decimal;
}

int main() {
    int Q, M;
    printf("Enter the Dividend (Q): ");
    scanf("%d", &Q);
    printf("Enter the Divisor (M): ");
    scanf("%d", &M);

    if (Q < 0 || M <= 0) {
        printf("Please enter a non-negative Dividend and a positive Divisor.\n");
        return 1;
```

```
}

short N = 4;
if (Q >= (1 << N) || M >= (1 << N)) {
    printf("Please enter numbers less than %d.\n", (1 << N));
    return 1;
}

int A = Q;
int q = 0;

int* binaryQ = (int*)malloc(N * sizeof(int));
int* binaryM = (int*)malloc(N * sizeof(int));

decimalToBinary(Q, N, binaryQ);
decimalToBinary(M, N, binaryM);

printf("Binary of Dividend (Q): ");
displayBinary(binaryQ, N);
printf("\nBinary of Divisor (M): ");
displayBinary(binaryM, N);
printf("\n");

M <<= N;

for (int i = N - 1; i >= 0; i--) {
```

```
    A = (A << 1) - M;

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

printf("Quotient (q): ");
decimalToBinary(q, N, binaryQ);
displayBinary(binaryQ, N);
printf("\n");

printf("Remainder (A): ");
decimalToBinary(A, N, binaryM);
displayBinary(binaryM, N);
printf("\n");
int decimalQuotient = binaryToDecimal(binaryQ, N);
int decimalRemainder = binaryToDecimal(binaryM, N);

printf("Quotient in decimal: %d\n", decimalQuotient);
printf("Remainder in decimal: %d\n", decimalRemainder);

free(binaryQ);
```

```
    free(binaryM);


    return 0;
}
```

Output:-

```
Enter the Dividend (Q): 6
Enter the Divisor (M): 3
Binary of Dividend (Q): 0110
Binary of Divisor (M): 0011
Quotient (q): 0010
Remainder (A): 0000
Quotient in decimal: 2
Remainder in decimal: 0
```

Conclusion:-

The restoring division algorithm is an efficient and user-friendly method for division tasks. It simplifies hardware design, accelerates calculations, and reduces the likelihood of errors. Its reliability and practicality make it an excellent choice for various applications, ensuring accurate results while optimizing performance in computational processes.

Post Lab Descriptive Questions

What are the advantages of restoring division over non restoring division?

Simplicity: Restoring division is simpler to implement because it consistently restores the dividend after a negative result, making the algorithm easier to understand and design.

Predictability: The restoring method's predictable steps (subtract and restore) simplify the process, whereas non-restoring division requires additional corrections for negative results.

Ease of Hardware Implementation: Restoring division's straightforward approach can simplify hardware design compared to the more complex corrections needed for non-restoring division.

Date: 29/07/2024