**Batch: B3 Roll No.: 121**

**Experiment / assignment / tutorial No.\_\_\_3\_\_\_**

**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)**

CO1 – Describe and define the structure of a computer with buses structure and detail working of the arithmetic logic unit and its sub modules.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.

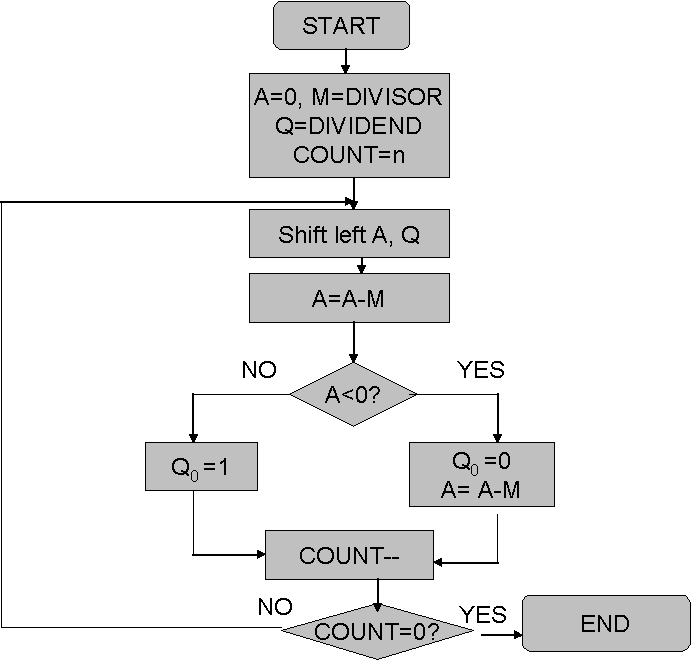
**3**. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

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

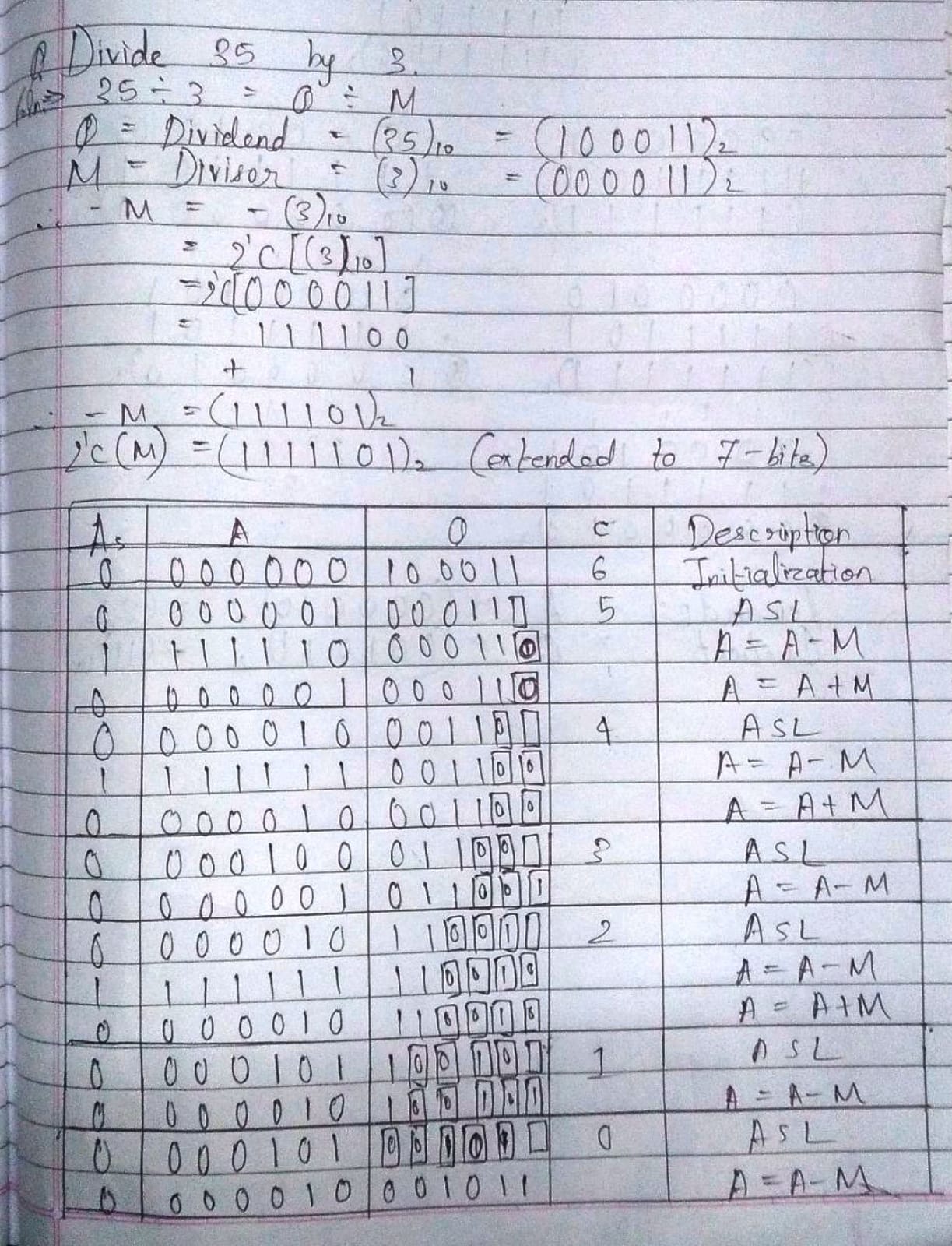
**Flowchart for Restoring of Division:**

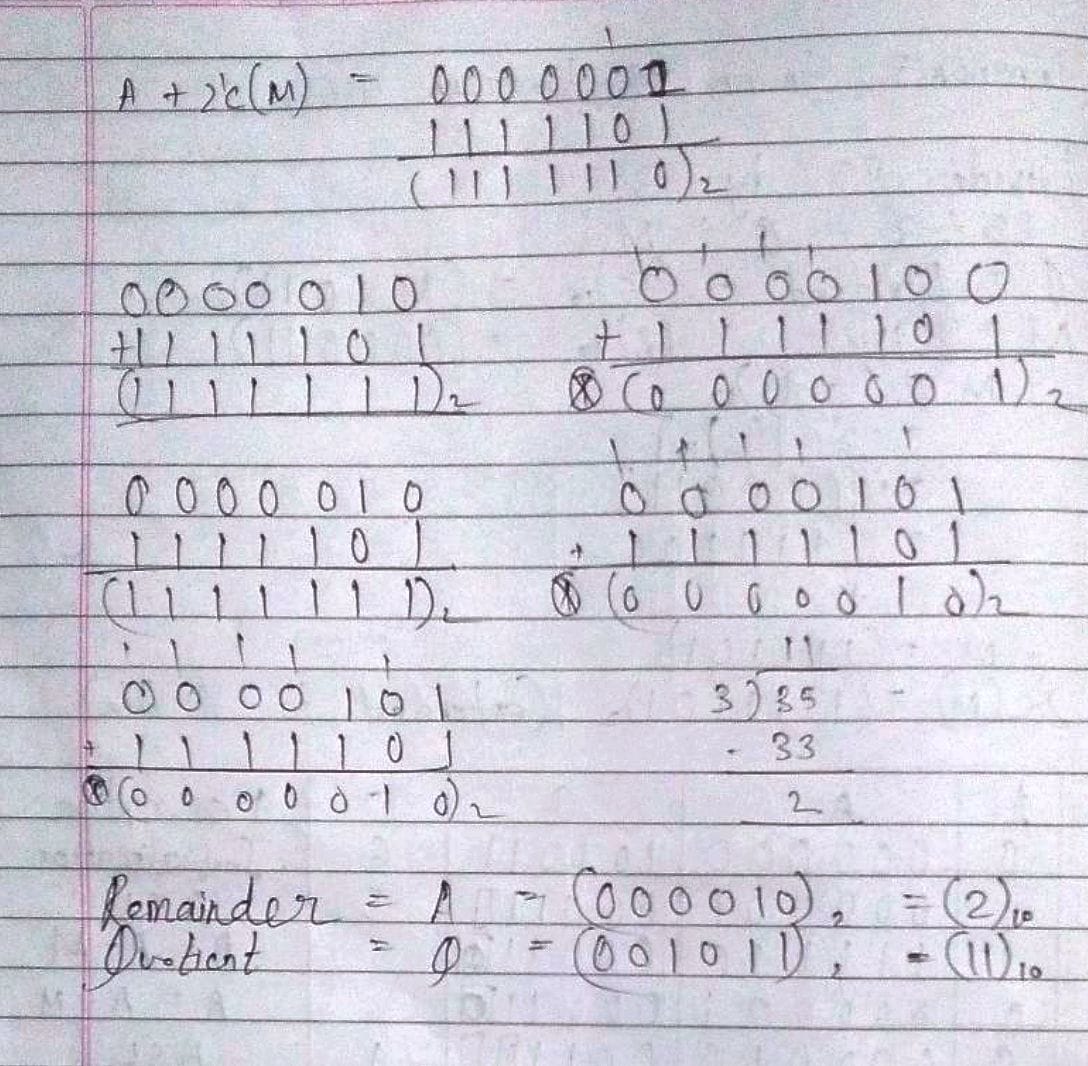


**Design Steps**:

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. Q0=0 & store present A
9. Else Q0=0 & restore previous A
10. Decrement count.
11. If count=0 go to 11
12. Else go to 3
13. STOP

**Example:- (Handwritten solved problems needs to be uploaded)**





Code:

#include<stdio.h>

#include<conio.h>

int dec\_bin(int d, int m[])

{

int b = 0, i = 0;

for(i = 0; i < 4; i++)

{

m[i] = d % 2;

d = d / 2;

}

return 0;

}

int twos(int m[], int m2[])

{

int i, m1[4];

for(i = 0; i < 4; i++)

{

if(m[i]==0)

{

m1[i] = 1;

}

else

{

m1[i] = 0;

}

}

for(i = 0; i < 4; i++)

{

m2[i] = m1[i];

}

if(m2[0] == 0)

{

m2[0] = 1;

}

else

{

m2[0] = 0;

if(m2[1] == 0)

{

m2[1] = 1;

}

else

{

m2[1] = 0;

if(m2[2] == 0)

{

m2[2] = 1;

}

else

{

m2[2] = 0;

if(m2[3] == 0)

{

m2[3] = 1;

}

else

{

m2[3] = 0;

}

}

}

}

return 0;

}

int left(int acc[], int q[])

{

int i;

for(i = 3; i > 0; i--)

{

acc[i] = acc[i-1];

}

acc[0] = q[3];

for(i = 3; i > 0; i--)

{

q[i] = q[i-1];

}

}

int add(int acc[], int m[])

{

int i, carry = 0;

for(i = 0; i < 4; i++)

{

if(acc[i]+m[i]+carry == 0)

{

acc[i] = 0;

carry = 0;

}

else if(acc[i]+m[i]+carry == 1)

{

acc[i] = 1;

carry = 0;

}

else if(acc[i]+m[i]+carry == 2)

{

acc[i] = 0;

carry = 1;

}

else if(acc[i]+m[i]+carry == 3)

{

acc[i] = 1;

carry = 1;

}

}

return 0;

}

int main()

{

int a, b, m[4] = {0, 0, 0, 0}, q[4] = {0, 0, 0, 0}, acc[4] = {0, 0, 0, 0}, m2[4], i, n = 4;

printf("\nRESTORING\tDIVISION\tALGORITHM");

printf("\nNote:\nBoth Dividend as well as Divisor\nmust be less than 15");

do{

printf("\nEnter Dividend: ");

scanf("%d", &a);

printf("\nEnter Divisor: ");

scanf("%d", &b);

}while(a > 15 || b > 15);

dec\_bin(a, q);

dec\_bin(b, m);

twos(m, m2);

printf("\nA\tQ\tComments\n");

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

}

printf("\t");

for(i = 3; i >= 0; i--)

{

printf("%d", q[i]);

}

printf("\tStart\n");

while(n > 0)

{

left(acc, q);

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

}

printf("\t");

for(i = 3; i >= 0; i--)

{

printf("%d", q[i]);

}

printf("\_\tLeft Shift A,Q\n");

add(acc, m2);

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

}

printf("\t");

for(i = 3; i >= 1; i--)

{

printf("%d", q[i]);

}

printf("\_\tA=A-M\n");

if(acc[3] == 0)

{

q[0] = 1;

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

}

printf("\t");

for(i = 3; i >= 0; i--)

{

printf("%d", q[i]);

}

printf("\tQo = 1\n");

}

else

{

q[0] = 0;

add(acc, m);

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

}

printf("\t");

for(i = 3; i >= 0; i--)

{

printf("%d", q[i]);

}

printf("\tQ0=0; A=A+M\n");

}

n--;

}

printf("\nQuotient = ");

for(i = 3; i >= 0; i--)

{

printf("%d", q[i]);

}

printf("\tRemainder = ");

for(i = 3; i >= 0; i--)

{

printf("%d", acc[i]);

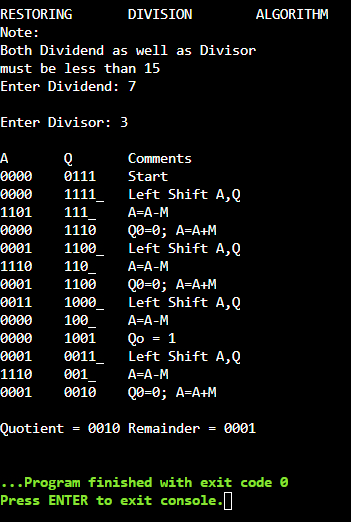
}

printf("\n");

return 0;

}

Output:



**Conclusion**

Thus, in this experiment, we have learnt about the restoring division algorithm. Using this algorithm, the hardware of earlier computers used to perform division of two unsigned binary numbers.

**Post Lab Descriptive Questions**

1. **What are the advantages of restoring division over non restoring division?**

Ans. The non-restoring division algorithm is more complex as compared to the restoring division algorithm. But when this algorithm is implemented in hardware, it has an advantage, i.e., it contains only one decision and addition/subtraction per quotient bit. After performing the subtraction operation, there will not be any restoring steps. Due to this, the number of operations basically are reduced to half. Because of less number of operations, this algorithm gets executed faster and less CPU processing power and memory is required.

**Date: \_\_**07-11-22\_\_ **Signature of faculty in-charge**