

Laboratory Journal
of
COMPUTER ORGANIZATION AND SYSTEM
SOFTWARE

*For completion of term work of 8th semester
curriculum program*

Bachelor of Technology
In
ELECTRONICS AND TELECOMMUNICATION ENGINEERING



DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION
ENGINEERING

Dr. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY

Lonere-402 103, Tal. Mangaon, Dist. Raigad (MS)

INDIA

Computer Organization and System Software

Lab Manual

INDEX

Exp. No.	Title	Page No.
1	Decimal to Binary Conversion	2
2	Decimal to Octal Conversion	5
3	Decimal to hexadecimal conversion	8
4	Binary to Decimal Conversion	12
5	Octal to Decimal Conversion	15
6	Hexadecimal to Decimal Conversion	18
7	2's complement of binary number	21
8	Half adder and Full adder	24
9	Restoring division method	31
10	Non-restoring division method	38
11	Booth Algorithm to multiply two signed binary numbers	42

Experiment No.: 1

Title: Decimal to Binary Conversion.

Aim: C Program to Convert Decimal number to Binary number.

Theory: To convert a decimal number into binary number it requires successive division by 2 writing down each quotient and its remainder. The remainders are taken in the reverse order, which is the binary equivalent of the decimal number. For example, let it is required to convert the decimal number 25 to its binary equivalent.

$$\begin{array}{r|rr} 2 & 25 & \\ \hline 2 & 12 & -1 \\ \hline 2 & 6 & -0 \\ \hline 2 & 3 & -0 \\ \hline & 1 & -1 \end{array}$$

The binary equivalent for $25_{10} = 11001_2$

To convert decimal fractions into equivalent binary fractions repeatedly double the decimal fraction. The number (0 or 1) that appears on the left is written separately. The bits that are written in this manner are read from top to bottom with a decimal point on the left.

Algorithm:

- 1) Enter decimal number in a.
- 2) Use while loop upto ($a > 0$), perform division by 2.
- 3) Increment number.
- 4) $d[i] = b$.
- 5) Print binary number using for loop for $d[i]$.
- 6) Exit.

//C Program for Decimal to Binary conversion.

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

    int i=0,a,b,d[10];

    clrscr();

    printf("enter the decimal number");

    scanf("%d",&a);

    while(a>0)

    {

        b=a%2;

        a=a/2;

        d[i]=b;

        i++;

    }

    printf("Binary number is ");

    for (i=i-1;i>=0;i--)

    {

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

    }

    getch();

}
```

Output:



```
C:\TC\BIND2B.EXE
enter the decimal number15
Binary number is 1111
```

Conclusion:

Experiment No.: 2

Title: Decimal to Octal Conversion.

Aim: C Program to convert decimal number to octal number.

Theory: Conversion from decimal to octal can be performed by repeatedly dividing the decimal number by 8 and using each remainder as a digit in the octal number being formed. For instance, to convert decimal number 200 to an octal representation, we divide as follows.

$$\begin{array}{r} 8 \overline{) 200} \\ 8 \overline{) 25} \quad - 0 \\ \quad 3 \quad - 1 \end{array}$$

Therefore $(200)_{10} = (310)_8$

Algorithm:

- 1) Enter the decimal number in a.
- 2) Use while loop upto $(a > 0)$, perform division by 8.
- 3) Increment counter.
- 4) $d[i] = a$.
- 5) Print octal number using for loop for $d[i]$.
- 6) Exit.

//C Program for Decimal to Octal conversion.

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

    int i=0,a,b,d[10];

    clrscr();

    printf("enter the decimal number");

    scanf("%d",&a);

    while(a>0)

    {

        b=a%8;

        a=a/8;

        d[i]=b;

        i++;

    }

    printf("Equivalent Octal number is ");

    for(i=i-1;i>=0;i--)

    {

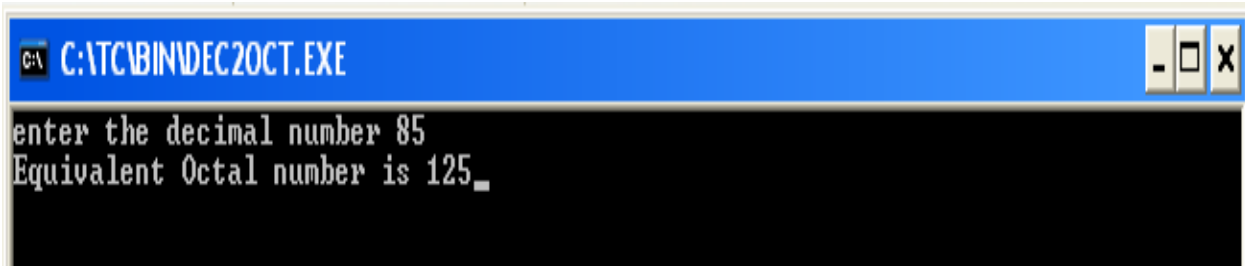
        printf("%d",d[i]);

    }

    getch();

}
```

Output:



```
C:\TC\BIN\DEC2OCT.EXE
enter the decimal number 85
Equivalent Octal number is 125_
```

Conclusion:

Experiment No.: 3

Title: Decimal to hexadecimal conversion.

Aim: C Program to convert decimal number to hexadecimal number.

Theory: One way to convert decimal to Hexadecimal is the hex dabbles. The idea is as divide successively by 16, writing down the remainders. Here is a sample of how it is done. To convert decimal 2429 to hexadecimal,

$$\begin{array}{r|l} 16 & 2429 \\ 16 & 154 - 15 \text{ ————— } F \\ & 9 - 10 \text{ ————— } A \end{array}$$

Therefore $(2429)_{10} = (9AF)_{16}$

Algorithm:

- 1) Enter the decimal number.
- 2) Use while loop upto $(dec_no > 7)$, perform division by 16
- 3) Increment counter
- 4) Use case statement for character hex number.
- 5) Print hexadecimal number.
- 6) Exit.

//C Program for Decimal to Hexadecimal conversion.

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int main(void)

{

    long dec_no,iter=0;

    char hex[100];

    printf("\nEnter the decimal number :");

    scanf("%ld",&dec_no);

    while(dec_no>7)

    {

        switch(dec_no%16)

        {

            case 10: hex[iter++]='A';

                break;

            case 11: hex[iter++]='B';

                break;

            case 12: hex[iter++]='C';

                break;

            case 13: hex[iter++]='D';

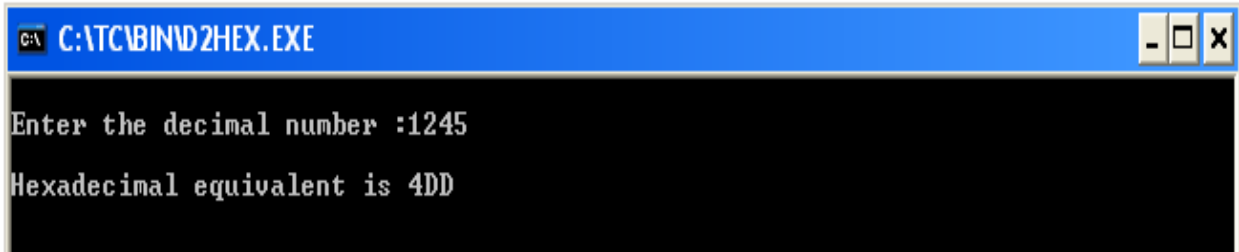
                break;

            case 14: hex[iter++]='E';

                break;
```

```
        case 15: hex[iter++]='F';  
                break;  
        default: hex[iter++]=(dec_no%16)+48;  
    }  
    dec_no/=16;  
}  
  
    hex[iter++]=dec_no+48;  
    hex[iter++]='\0';  
    printf("\nHexadecimal equivalent is %s\n",strrev(hex));  
    getch();  
}
```

Output:



```
C:\TC\BIN\D2HEX.EXE
Enter the decimal number :1245
Hexadecimal equivalent is 4DD
```

Conclusion:

Experiment No.: 4

Title: Binary to Decimal Conversion.

Aim: C Program to convert binary number to decimal number.

Theory: For converting the value of Binary numbers to decimal equivalent we have to find its quantity, which is found by multiplying a digit by its place value. The following example illustrates the conversion of binary numbers to decimal system.

$$\begin{aligned}101 &= 1*2^{3-1} + 0*2^{3-2} + 1*2^{3-3} \\&= 1*2^2 + 0*2^1 + 1*2^0 \\&= 4 + 0 + 1 \\&= 5 \\1001 &= 1*2^{4-1} + 0*2^{4-2} + 0*2^{4-3} + 1*2^{4-4} \\&= 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 \\&= 8 + 1 \\&= 9\end{aligned}$$

Algorithm:

- 1) Get a binary number from the user.
- 2) Use for loop to reverse the number and put all reversed numbers in array a[i].
- 3) Count to count the number of times this loop runs.
- 4) Count -1 condition is used to run the loop till the previous loop run.
- 5) Use pow function to raise the power of 2 to no of times previous loop run.
- 6) Multiply a[i] or reversed binary no with b[i] or increasing pow of 2 to count-1.
- 7) Add the c[i] elements with each other n put into sum variable.
- 8) Print the sum to get the decimal form.

//C Program for Binary to Decimal Conversion.

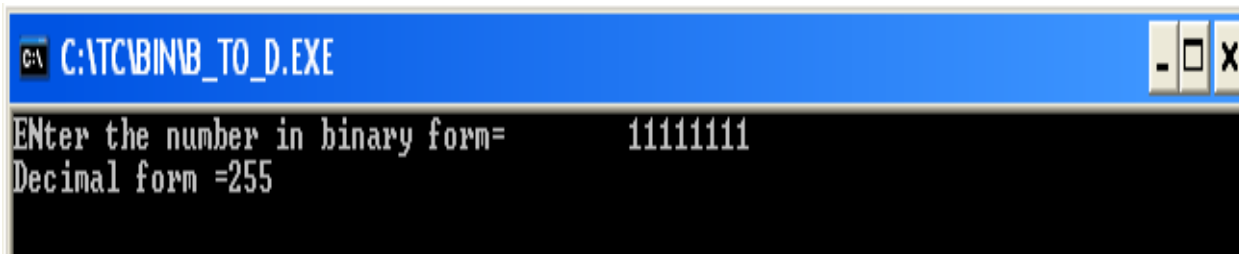
```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
long int a[20],i,n,count=0,b[20],c[20],sum=0;
printf("ENter the number in binary form=\t");
scanf("%ld",&n);

for (i=0;n>=1;i++)
{
a[i]=n%10;
n=n/10;

count=count + 1; }
for (i=0;i<=count-1;i++)

{
b[i]=pow(2,i);
}
for (i=0;i<=count-1;i++)
{
c[i]=a[i] * b[i];

sum=sum +c[i];
}
printf("Decimal form =%ld",sum);
getch();
}
```

Output:

```
C:\TC\BINB_TO_D.EXE
Enter the number in binary form= 11111111
Decimal form =255
```

Conclusion:

Experiment No.: 5

Title: Octal to Decimal Conversion.

Aim: C Program to convert octal number to decimal number.

Theory: To convert an octal number to a decimal number, we use the same sort of polynomial as was used in the binary case, except that we now have a radix of 8 instead of 2. Therefore 1213 in octal is,

$$\begin{aligned} &= 1*8^3 + 2*8^2 + 1*8^1 + 3*8^0 \\ &= 512 + 128 + 8 + 3 = 651 \end{aligned}$$

in decimal. Also, 1.123 in octal is $1*8^0 + 1*8^{-1} + 2*8^{-2} + 3*8^{-3} = 1.83/512$ in decimal

Algorithm:

1. Enter octal number a.
2. Calculate multiplying factor as
 $k = k + (s[i] * (\text{pow}(8, i)))$
3. Print the equivalent decimal number.

//C Program for Octal to Decimal conversion.

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

int a,s[100],b,i=0,k=0;

clrscr();

printf("Enter a octal number: ");

scanf("%d",&a);

while(a>0)

{

b=a%10;

a=a/10;

s[i]=b;

i++;

}

for (i=i-1;i>=0;i--)

{

k=k+(s[i]*(pow(8,i)));

}

printf("Decimal equivalent is %d.",k);

getch();

}
```

Output:



```
C:\TC\BIN\OCT2DEC.EXE
Enter a octal number: 125
Decimal equivalant is 85.
```

Conclusion:

Experiment No.: 6

Title: Hexadecimal to Decimal Conversion.

Aim: C Program to convert hexadecimal number to decimal number.

Theory: The conversion of Hexadecimal to decimal is straightforward but time consuming. In Hexadecimal number system each digit position corresponds to a power of 16. The weights of the digit positions in a hexadecimal number are as follows: For instance, BB represents,

$$\begin{aligned} \text{BB} &= \text{B} * 16^1 + \text{B} * 16^0 \\ &= 11 * 16 + 11 * 1 \\ &= 176 + 11 \\ &= 187 \end{aligned}$$

Algorithm:

1. Enter the hexadecimal number
2. Using ASCII values of characters and numbers the equivalent decimal number is determined.
3. Print the equivalent decimal number.

//C Program for Hexadecimal to Decimal conversion.

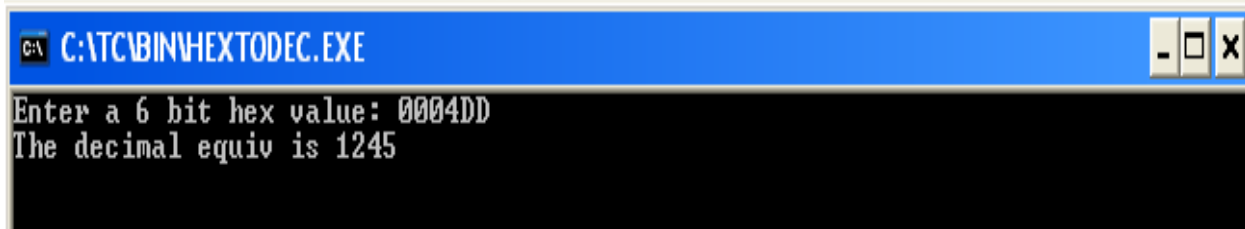
```
#include<stdio.h>

#include<conio.h>

void main()
{
    char ch[5];
    int i,b=0;
    long unsigned dec=0,p=1;
    clrscr();
    printf("Enter a 6 bit hex value: ");
    scanf("%s",&ch);
    for(i=5;i>=0;i--)
    {
        if((47<ch[i])&&(ch[i]<58))
            b=ch[i]-48;
        else
            if((64<ch[i])&&(ch[i]<72))
                b=ch[i]-55;
            else
                b=0;

        dec=dec+(b*p);
        p=p*16;
    }
}
```

```
printf("The decimal equiv is %lu",dec);  
  
getch();  
  
}
```

Output:**Conclusion:**

Experiment No.: 7

Title: 2's complement of binary number.

Aim: C Program for 2's complement of binary number.

Theory: 2's complement is used for subtraction of two binary numbers. To find the 2's complement of given binary number, number is checked from LSB. When first one occurs, the bits after that 1 are complemented or inverted.

Algorithm:

1. Take 8 bit binary number in bin[i]
2. To check occurrence of first one from LSB if statement is used.
3. Until the first one occur all bits in input bin[i] are copied in two's complemented array output bin[i]
4. When first one occur i.e. bin[i]=1.
5. Then bits after that one are complemented
6. Print 2's complement bin[i]

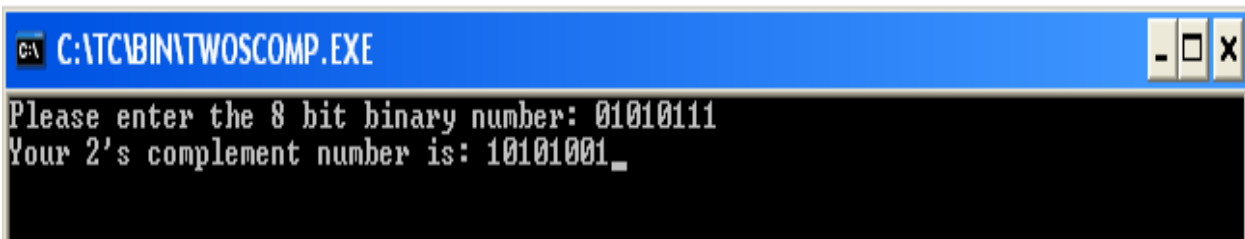
//C Program for 2's Complement.

```
#include<stdio.h>

#include<conio.h>

Void main()
{
    clrscr();
    int bin[8],c=0,i,j;
    printf("Please enter the 8 bit binary number: ");
    for(i=7;i>=0;i--)
    {
        bin[i]=getche();
    }
    for(i=0;i<=7;i++)
    {
        if(bin[i]==49)
        {
            bin[i]=1;
            c++;
            break;
        }
        bin[i]=0;
        c++;
    }
    for(i=7;i>=c;i--)
```

```
{  
    if(bin[i]==48)  
    {  
        bin[i]=1;  
    }  
    else  
        bin[i]=0;  
}  
printf("\nYour 2's complement number is: ");  
for(i=7;i>=0;i--)  
{  
    printf("%d",bin[i]);  
}  
getch();  
}
```

Output:

```
C:\TC\BIN\TWOSECOMP.EXE  
Please enter the 8 bit binary number: 01010111  
Your 2's complement number is: 10101001_
```

Conclusion:

Experiment No.: 8

Title: Half adder and Full adder.

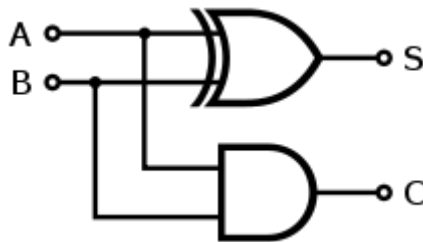
Aim: C Program for half adder and full adder.

Theory:

In electronics, an **adder** or **summer** is a digital circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit(s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar.

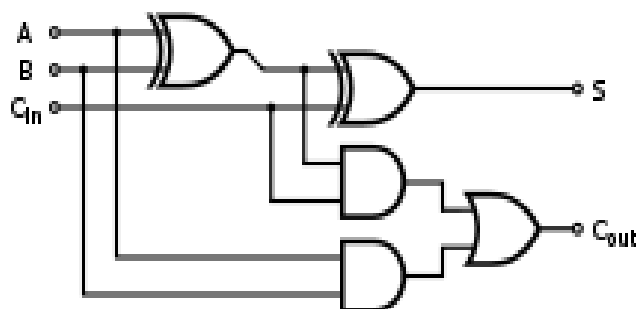
Although adders can be constructed for many numerical representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder–subtractor. Other signed number representations require a more complex adder.

Half adder



Example half adder logic diagram

The half adder is an example of a simple, functional digital circuit built from two logic gates. A half adder adds two one-bit binary numbers A and B . It has two outputs, S and C (the value theoretically carried on to the next addition); the final sum is $2C + S$. The simplest half-adder design, pictured on the right, incorporates an XOR gate for S and an AND gate for C . Half adders cannot be used compositely, given their incapacity for a carry-in bit.

Full adder

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A , B , and C_{in} ; A and B are the operands, and C_{in} is a bit carried in (in theory from a past addition). The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers. The circuit produces a two-bit output sum typically represented by the signals C_{out} and S , where $\text{sum} = 2 \times C_{out} + S$. The one-bit full adder's truth table is:

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Truth Table

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates. One example implementation is with $S = A \oplus B \oplus C_{in}$ and $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$.

In this implementation, the final OR gate before the carry-out output may be replaced by an XOR gate without altering the resulting logic. Using only two types of gates is convenient if the circuit is being implemented using simple IC chips which contain only one gate type per chip. In this light, C_{out} can be implemented as $C_{out} = (A \cdot B) \oplus (C_{in} \cdot (A \oplus B))$.

A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting the sum from that to an input to the second adder, connecting C_i to the other input and OR the two carry outputs. Equivalently, S could be made the three-bit XOR of A , B , and C_i , and C_{out} could be made the three-bit majority function of A , B , and C_i .

//C Program for Half Adder

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

int a,b,c,add;

clrscr();

printf("Enter the binary digit");

scanf("%d%d",&a,&b);

if (a>1||b>1)

printf("number is not binary");

else

{

add=a^b;

c=a&&b;

printf("addition=%d\ncarry=%d",add,c);

}

getch();

}
```

Output:

```
C:\TC\BIN\HALFADDE.EXE
Enter the binary digit 1
1
addition=0
carry=1
```

//C Program for Full Adder

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

int a,b,c,add;

clrscr();

printf("Enter the binary digits");

scanf("%d%d%d",&a,&b,&c);

if (a>1||b>1||c>1)

printf("number is not binary");

else

{

add=a^b^c;

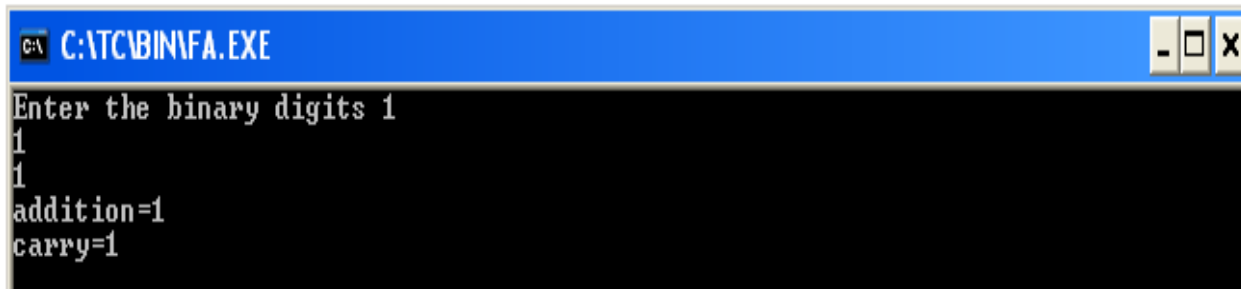
c=(a&&b)||(b&&c)||(c&&a);

printf("addition=%d\ncarry=%d",add,c);

}

getch();

}
```

Output:

```
C:\TC\BIN\F.A.EXE
Enter the binary digits 1
1
1
addition=1
carry=1
```

Conclusion:

Experiment No.: 9

Title: Restoring division method.

Aim: C Program for restoring method of division.

Theory: This is the method for division of two fixed point binary numbers. But it requires more number of steps than non – restoring method of division.

Algorithm:

1. Initial conditions are set that are M is Divisor (having one bit extra than dividend) Q is Dividend.
Count= No. of bits of dividend
A= zero (same number of bits as divisor has)
Shift left A, Q.
- 3) Subtract M from A i.e. $A=A-M$
- 4) Let C is MSB of A
- 5) If C is 1, then set Q_0 bit 0 and Add M into A (i.e. $A=A+M$) it is called as restoring step
Else if C is 0, then set Q_0 bit 1.
- 6) Count is decremented by one.
- 7) If count is not equal to zero then all the steps from (2) are repeated
- 8) If count is equal to zero then program is terminated.

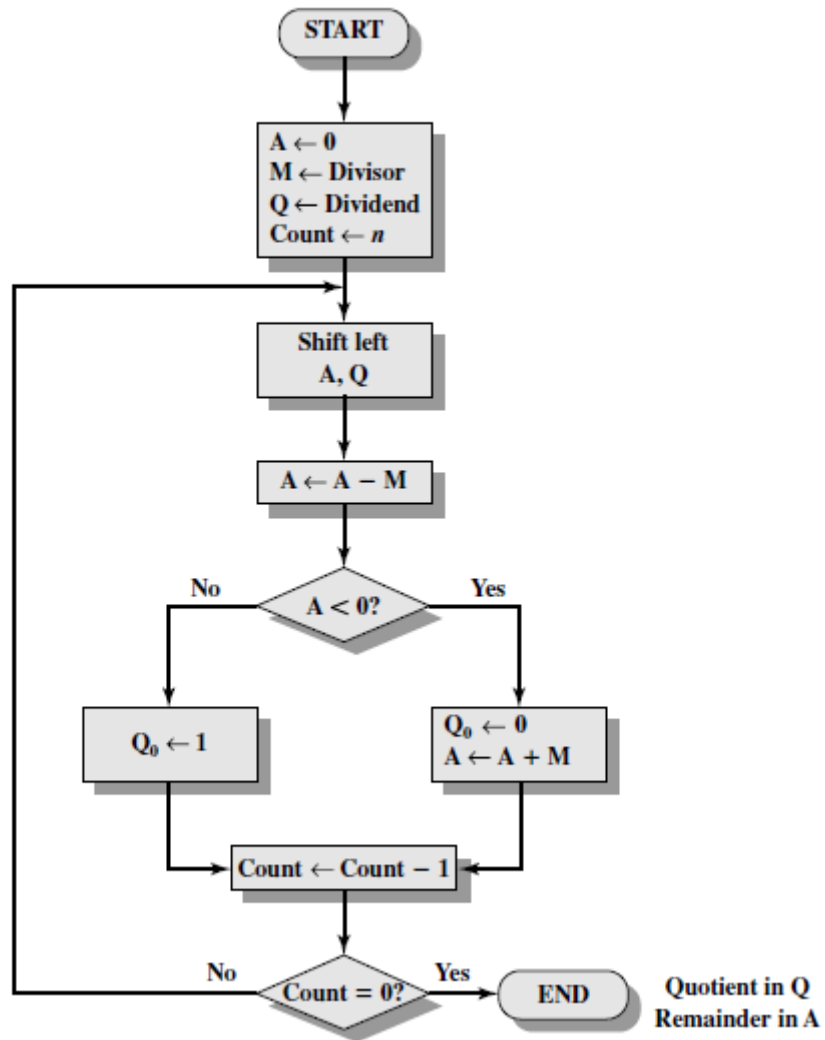
Flowchart:

Figure: Flowchart for Restoring Division Method

```
//C Program for Restoring Division Method
#include<stdio.h>
#include<math.h>

int getsize(int x)
{
    int c;
    if (x <= 1)
        c = 2;
    else if (x < 4)
        c = 2;
    else if (x < 8)
        c = 3;
    else if (x < 16)
        c = 4;
    else if (x < 32)
        c = 5;
    else if (x < 64)
        c = 6;
    else if (x < 128)
        c = 7;
    else if (x < 256)
        c = 8;
    else if (x < 512)
        c = 9;
    return c;
}

int max(int x, int y)
{
    if (x < y)
        return (y);
    else
        return (x);
}

void main()
{
    int B, Q, Z, M, c, c1, e, f, g, h, i, j, x, y, ch, in, S, G, P;
    int a[24], b[12], b1[12], q[12], carry = 0, count = 0, option;
    long num;
    do
    {
        printf("|-----|\n");
        printf("| \t\t\tPROGRAM FOR DIVISION\t\t\t|\n");
        printf("|-----|");
        printf("\n\nENTER DIVIDEND\t: ");
        scanf("%d", &Q);
        y = getsize(Q);
        printf("ENTER DIVISOR\t: ");
        scanf("%d", &M);
        x = getsize(M);
        Z = max(x, y);
        printf("\n\tTOTAL BITS CONSIDERED FOR RESULT => %d", 2 * Z + 1);
    }
```

```
printf("\n\tINITiALLY A IS RESET TO ZERO:");
for (i = 0; i <= Z; i++)
    printf("%d ", a[i] = 0);
for (i = Z; i >= 0; i--)
{
    b1[i] = b[i] = M % 2;
    M = M / 2;
    b1[i] = 1-b1[i];
}
carry = 1;
for (i = Z; i >= 0; i--)
{
    c1 = b1[i] ^ carry;
    carry = b1[i] && carry;
    b1[i] = c1;
}
for (i = 2 * Z; i > Z; i--)
{
    a[i] = Q % 2;
    Q = Q / 2;
}
printf("\n\n\tDivisor\t\t(M)\t\t: ");
for (i = 0; i <= Z; i++)
    printf("%d ", b[i]);
printf("\n\t2'C Divisor\t(M)\t\t: ");
for (i = 0; i <= Z; i++)
    printf("%d ", b1[i]);
printf("\n\tDividend\t(Q)\t\t: ");
for (i = Z + 1; i <= 2 * Z; i++)
    printf("%d ", a[i]);
printf("\n\n\tBITS CONSIDERED:[ A ] [ M ]");
printf("\n\t\t\t");
for (i = 0; i <= Z; i++)
    printf("%d ", a[i]);
printf(" ");
for (i = Z + 1; i <= 2 * Z; i++)
    printf("%d ", a[i]);
count = Z;
do
{
    for (i = 0; i < 2 * Z; i++)
        a[i] = a[i + 1];
    printf("\n\nLeft Shift\t\t");
    for (i = 0; i <= Z; i++)
        printf("%d ", a[i]);
    printf(" ");
    for (i = Z + 1; i < 2 * Z; i++)
        printf("%d ", a[i]);
    carry = 0;
    for (i = Z; i >= 0; i--)
    {
        S = a[i] ^ (b1[i] ^ carry);
        G = a[i] && b1[i];
        P = a[i] ^ b1[i];
        carry = G || (P && carry);
    }
}
```

```

    a[i] = S;
}
printf("\nA< -A-M \t\t");
for (i = 0; i <= Z; i++)
    printf("%d ", a[i]);
printf(" ");
for (i = Z + 1; i < 2 * Z; i++)
    printf("%d ", a[i]);
ch = a[0];
printf("\nBIT Q:%d", ch);
switch (ch)
{
    case 0:
        a[2 * Z] = 1;
        printf(" Q0< -1\t\t");
        for (i = 0; i <= Z; i++)
            printf("%d ", a[i]);
        printf(" ");
        for (i = Z + 1; i <= 2 * Z; i++)
            printf("%d ", a[i]);
        break;

    case 1:
        a[2 * Z] = 0;
        printf(" Q0< -0\t\t");
        for (i = 0; i <= Z; i++)
            printf("%d ", a[i]);
        printf(" ");
        for (i = Z + 1; i < 2 * Z; i++)
            printf("%d ", a[i]);
        carry = 0;
        for (i = Z; i >= 0; i--)
        {
            S = a[i] ^ (b[i] ^ carry);
            G = a[i] && b[i];
            P = a[i] ^ b[i];
            carry = G || (P && carry);
            a[i] = S;
        }
        printf("\nA< -A+M");
        printf("\t\t\t");
        for (i = 0; i <= Z; i++)
            printf("%d ", a[i]);
        printf(" ");
        for (i = Z + 1; i <= 2 * Z; i++)
            printf("%d ", a[i]);
        break;
}
count--;
}
while (count != 0);
num = 0;
printf("\n\t\t< < QUOTIENT IN BITS>> :");
for (i = Z + 1; i <= 2 * Z; i++)
{

```

```
        printf("%d ", a[i]);
        num = num + pow(2, 2 *Z - i) *a[i];
    }
    printf("\n\t\tQUOTIENT IN DECIMAL :%d", num);
    num = 0;
    printf("\n\t\t< < REMAINDER IN BITS>>:");
    for (i = 0; i <= Z; i++)
    {
        printf("%d ", a[i]);
        num = num + pow(2, Z - i) *a[i];
    }
    printf("\n\t\tREMAINDER IN DECIMAL :%d", num);
    getch();
    printf("\n\tDO YOU WANT TO CONTINUE PRESS 0-ESC 1-CONT.:");
    scanf("%d", &option);
}
while (option != 0)
    ;
}
```

Output:

```

a-----a
a          PROGRAM FOR DIVISION          a
a-----a

ENTER DIVIDEND  : 15
ENTER DIVISOR   : 2

TOTAL BITS CONSIDERED FOR RESULT => 9
INITIALLY A IS RESET TO ZERO:0 0 0 0 0

Divisor      (M)      : 0 0 0 1 0
2'C Divisor  (M)      : 1 1 1 1 0
Dividend     (Q)      : 1 1 1 1

BITS CONSIDERED:[ A ] [ M ]
                0 0 0 0 0 1 1 1 1

Left Shift      0 0 0 0 1 1 1 1
A< -A-M         1 1 1 1 1 1 1 1
BIT Q:1 Q0< -0  1 1 1 1 1 1 1 1
A< -A+M         0 0 0 0 1 1 1 0

Left Shift      0 0 0 1 1 1 1 0
A< -A-M         0 0 0 0 1 1 1 0
BIT Q:0 Q0< -1  0 0 0 0 1 1 1 0 1

Left Shift      0 0 0 1 1 1 0 1
A< -A-M         0 0 0 0 1 1 0 1
BIT Q:0 Q0< -1  0 0 0 0 1 1 0 1 1

Left Shift      0 0 0 1 1 0 1 1
A< -A-M         0 0 0 0 1 0 1 1
BIT Q:0 Q0< -1  0 0 0 0 1 0 1 1 1

< < QUOTIENT IN BITS>> :0 1 1 1
QUOTIENT IN DECIMAL :7
< < REMAINDER IN BITS>>:0 0 0 0 1
REMAINDER IN DECIMAL :1

```

Conclusion:

Experiment No.: 10

Title: Non-restoring division method.

Aim: C Program for non-restoring method of division.

Theory: This is the method for division of two fixed point binary numbers. But it requires less number of steps than restoring method of division.

Algorithm:

1. Initial conditions are set that are M is divisor (having one bit extra than dividend)
Q is Dividend.
Count= No. of bits of dividend
A=Accumulator.
C= One bit register.
Set AC and C initially at zero.
2. During each cycle, C is examined.
3. If C=1, means AC is negative, shift C, AC, Q left by one bit position and $AC=AC+M$.
Else if C=0, means AC is positive, shift C, AC, Q left by one bit position and $AC=AC-M$
4. Again C is examined.
5. If C=1, means AC is negative, set Q_0 bit 0
Else if C=0, means AC is positive, set Q_0 bit 1.
6. Follow the same procedure for n bits.
7. If count is not equal to zero then all the steps from (2) are repeated.
8. If count is equal to zero then again C is examined and
 - a. If C=1, means AC is negative, then $AC=AC+M$
 - b. Else if C=0, AC is positive, result is stored and program is terminated.

```
// C Program for the Non Restoring Division.
```

```
#include<stdio.h>
int a[5] = {0, 0, 0, 0, 0 }, q[4], b[5], b2c[5];
comp()
{
    int i = 4;
    do
    {
        b2c[i] = b[i];
        i--;
    }
    while (b[i + 1] != 1);
    while (i >= 0)
    {
        b2c[i] = (b[i] + 1) % 2;
        i--;
    }
    printf("\n\tB's complement:");
    for (i = 0; i < 5; i++)
        printf("%d", b2c[i]);
    printf("\n");
}
```

```
nonresdiv()
{
    shiftleft();
    if (a[0] == 0)
        a_minus_b();
    else
        a_plus_b();
    q[3] = (a[0] + 1) % 2;
}
```

```
shiftleft()
{
    int i;
    for (i = 0; i < 4; i++)
        a[i] = a[i + 1];
    a[4] = q[0];
    for (i = 0; i < 3; i++)
        q[i] = q[i + 1];
}
```

```
a_minus_b()
{
    int i, carry = 0, sum = 0;
    for (i = 4; i >= 0; i--)
    {
        sum = (a[i] + b2c[i] + carry);
        a[i] = sum % 2;
        carry = sum / 2;
    }
}
```

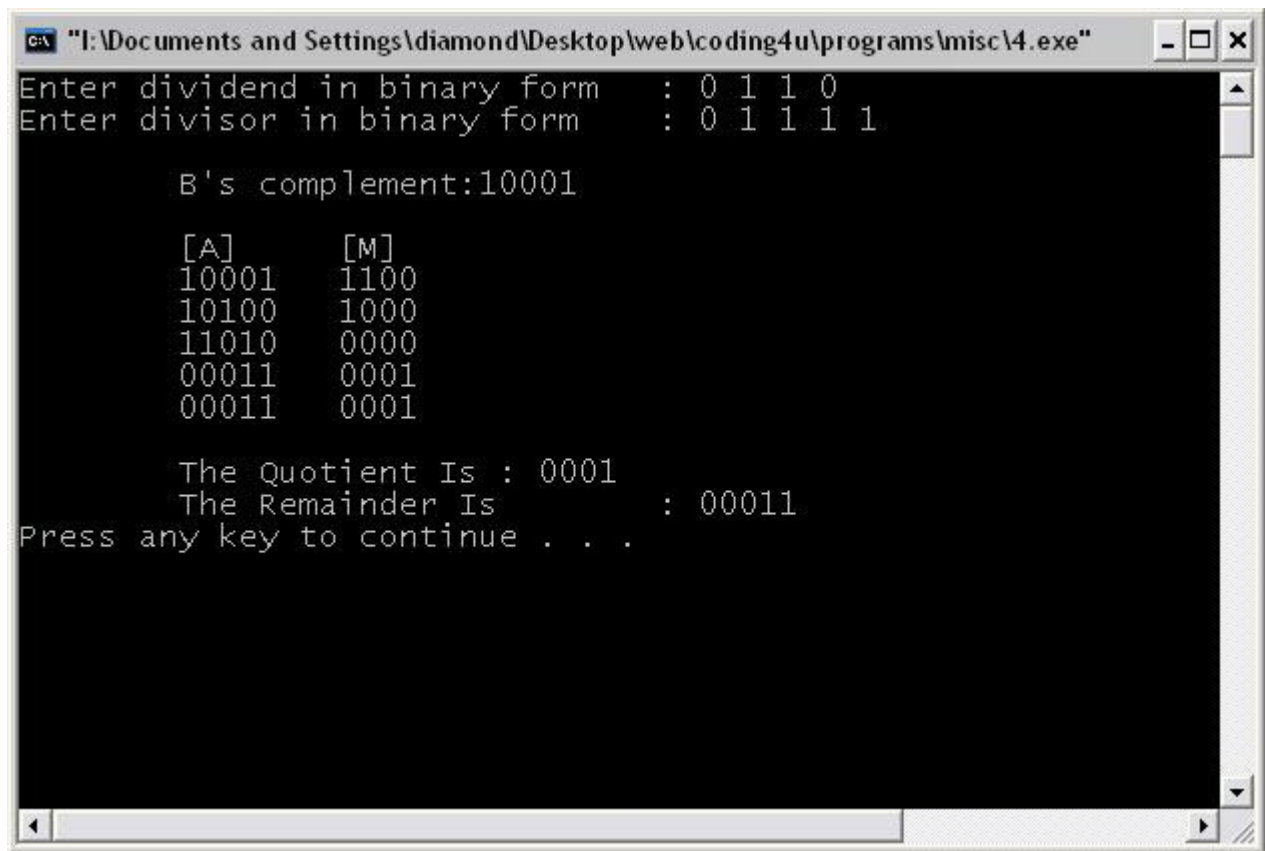
```
    }
}

a_plus_b()
{
    int i, carry = 0, sum = 0;
    for (i = 4; i >= 0; i--)
    {
        sum = (a[i] + b[i] + carry);
        a[i] = sum % 2;
        carry = sum / 2;
    }
}

void main()
{
    int i, j, k;

    printf("Enter dividend in binary form\t: ");
    for (i = 0; i < 4; i++)
        scanf("%d", &q[i]);
    printf("Enter divisor in binary form\t: ");
    for (i = 0; i < 5; i++)
        scanf("%d", &b[i]);
    comp();
    printf("\n\t[A]\t[M]\n");
    for (i = 0; i < 4; i++)
    {
        nonresdiv();
        printf("\t");
        for (j = 0; j < 5; j++)
            printf("%d", a[j]);
        printf("\t");
        for (k = 0; k < 4; k++)
            printf("%d", q[k]);
        printf("\n");
    }
    if (a[0] == 1)
        a_plus_b();
    printf("\t");
    for (j = 0; j < 5; j++)
        printf("%d", a[j]);
    printf("\t");
    for (k = 0; k < 4; k++)
        printf("%d", q[k]);
    printf("\n");
    printf("\n\tThe Quotient Is\t: ");
    for (k = 0; k < 4; k++)
        printf("%d", q[k]);
    printf("\n\tThe Remainder Is\t: ");
    for (j = 0; j < 5; j++)
        printf("%d", a[j]);
    printf("\n");
}
```

}

Out Put:

```
C:\ "I:\Documents and Settings\diamond\Desktop\web\coding4u\programs\misc\4.exe"
Enter dividend in binary form : 0 1 1 0
Enter divisor in binary form : 0 1 1 1 1

    B's complement:10001

    [A]    [M]
    10001   1100
    10100   1000
    11010   0000
    00011   0001
    00011   0001

    The Quotient Is : 0001
    The Remainder Is : 00011
Press any key to continue . . .
```

Conclusion:

Experiment No.:11

Title: Booth Algorithm to multiply two signed binary numbers.

Aim: C Program for Booth algorithm to multiply two signed binary numbers.

Theory: In this method least significant bit of multiplier Q i.e. Q_0 and Q_{-1} bits are examined.

According to these bits addition, subtraction or shifting is performed. And final answer will be stored in AC and Q registers.

Algorithm:

1. Assign Q =Multiplier, M =Multiplicand and Q_{-1} is one bit register placed to right side of Q_0 bit of Multiplier Q.
2. Initially Q_{-1} and A =Accumulator are set to zero.
3. During each cycle, Q_0 and Q_{-1} bits are examined as follows

If $Q_0=1$ and $Q_{-1}=0$, then $A=A-M$ and then right shift the bits of A, Q, and Q_{-1} by one bit position

Else if $Q_0=0$ and $Q_{-1}=1$, then $A=A+M$ and then right shift the bits of A, Q, and Q_{-1} by one bit position

Else if $Q_0=0$ and $Q_{-1}=0$, or $Q_0=1$ and $Q_{-1}=1$, then right shift the bits of A, Q, and Q_{-1} by one bit position.
4. Follow the same procedure for n cycles.
5. The result will be stored in A and Q.

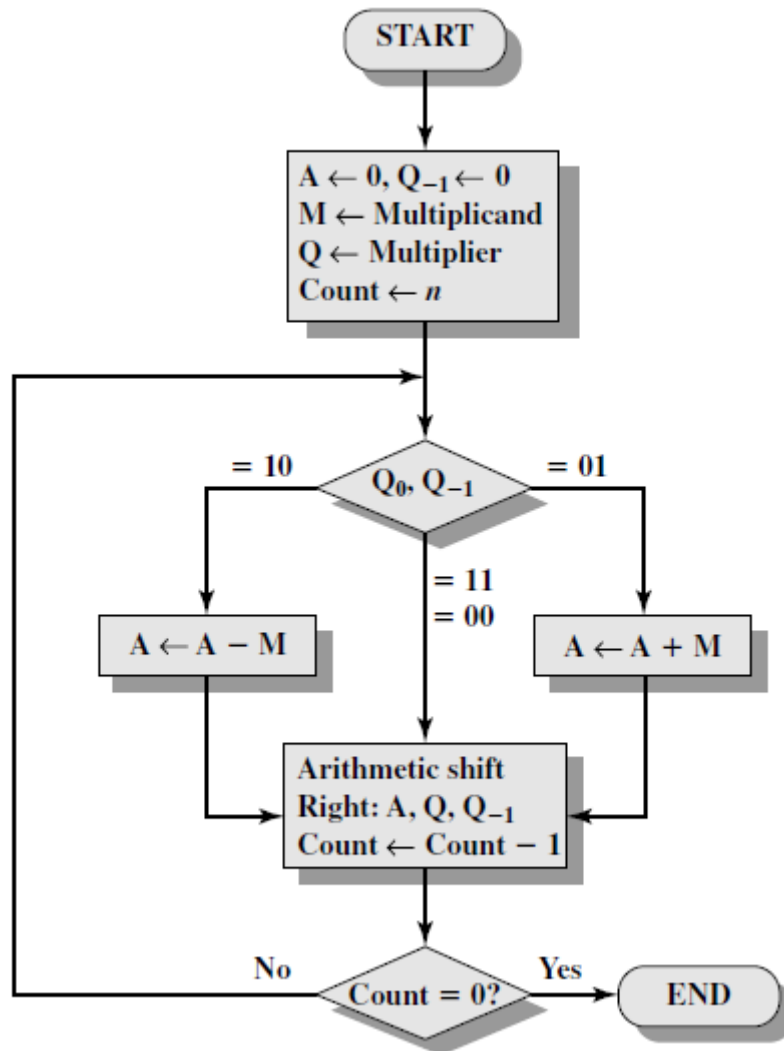
Flowchart:

Figure: Flowchart for Booth's algorithm.

```
// C Program to implement Booths Algorithm.
#include<stdio.h>
#include<process.h>
#include<math.h>

int get(int a)
{
    char ch = 'B';
    int flag = 0;
    if (a == 1)
        ch = 'A';
    do
    {
        printf("? ENTER VALUE OF %c: ", ch);
        scanf("%d", &a);
        if (a < 0)
        {
            a = a * - 1;
            flag = 1;
        }
        if (9 <= a)
            printf("\n\t!INVALID NUMBER.ENTER VALUE (-9 < A < 9)!");
    }
    while (9 <= a);
    if (flag)
        a = a * - 1;
    return (a);
}

void add(int *a, int *b)
{
    int x, i, c = 0;
    for (i = 3; i >= 0; i--)
    {
        x = a[i];
        a[i] = c ^ x ^ b[i];
        if (((c == 1) && (x == 1)) || ((x == 1) && (b[i] == 1)) || ((b[i]
== 1) &&
        (c == 1)))
            c = 1;
        else
            c = 0;
    }
}

void binary(int x, int *arr)
{
    int i, p = x, c[4] =
    {
        0, 0, 0, 1
    }
}
```

```
};
for (i = 0; i < 4; i++)
    arr[i] = 0;
if (x < 0)
    x = x * - 1;
i = 3;
do
{
    arr[i] = x % 2;
    x = x / 2;
    i--;
}
while (x != 0);
if (p < 0)
{
    for (i = 0; i < 4; i++)
        arr[i] = 1-arr[i];
    add(arr, c);
}
printf("\n\nTHE BINARY EQUIVALENT OF %d IS : ", p);
for (i = 0; i < 4; i++)
    printf("%d", arr[i]);
}

void rshift(int x, int *y)
{
    int i;
    for (i = 3; i > 0; i--)
        y[i] = y[i - 1];
    y[0] = x;
}

void main()
{
    int q = 0, i, j, a, b, A[4] =
    {
        0, 0, 0, 0
    }
    , C[4] =
    {
        0, 0, 0, 1
    }
    , C1[8] =
    {
        0, 0, 0, 0, 0, 0, 0, 1
    };
    int s = 0, z = 0, Q[4], M[4], temp, temp1[4], ans[8], y, x = 0, c =
0;
    printf("\n?-----\n");
    a = get(1);
```

```

b = get(0);
printf("\n?-----\n");
binary(a, M);
binary(b, Q);
printf("\n\n-----\n");
printf(" OPERATION\t\t A\t Q\tQ'\t M");
printf("\n\n INITIAL\t\t");
for (i = 0; i < 4; i++)
    printf("%d", A[i]);
printf("\t");
for (i = 0; i < 4; i++)
    printf("%d", Q[i]);
printf("\t");
printf("%d\t", q);
for (i = 0; i < 4; i++)
    printf("%d", M[i]);
for (j = 0; j < 4; j++)
{
    if ((Q[3] == 0) && (q == 1))
    {
        printf("\n A:=A+M \t\t");
        add(A, M);
        for (i = 0; i < 4; i++)
            printf("%d", A[i]);
        printf("\t");
        for (i = 0; i < 4; i++)
            printf("%d", Q[i]);
        printf("\t%d\t", q);
        for (i = 0; i < 4; i++)
            printf("%d", M[i]);
    }
    if ((Q[3] == 1) && (q == 0))
    {
        printf("\n A:=A-M \t\t");
        for (i = 0; i < 4; i++)
            temp1[i] = 1-M[i];
        add(temp1, C);
        add(A, temp1);
        for (i = 0; i < 4; i++)
            printf("%d", A[i]);
        printf("\t");
        for (i = 0; i < 4; i++)
            printf("%d", Q[i]);
        printf("\t%d\t", q);
        for (i = 0; i < 4; i++)
            printf("%d", M[i]);
    }
    printf("\n Shift \t\t\t");
    y = A[3];
    q = Q[3];
}

```

```

    rshift(A[0], A);
    rshift(y, Q);
    for (i = 0; i < 4; i++)
        printf("%d", A[i]);
    printf("\t");
    for (i = 0; i < 4; i++)
        printf("%d", Q[i]);
    printf("\t");
    printf("%d\t", q);
    for (i = 0; i < 4; i++)
        printf("%d", M[i]);
}
printf("\n\n-----\n");
printf("\nTHE ANSWER IN BINARY IS : ");
for (i = 0; i < 4; i++)
    ans[i] = A[i];
for (i = 0; i < 4; i++)
    ans[i + 4] = Q[i];
if (((a < 0) && (b > 0)) || ((a > 0) && (b < 0)))
{
    for (i = 0; i < 8; i++)
        ans[i] = 1-ans[i];
    for (i = 7; i >= 0; i--)
    {
        x = ans[i];
        ans[i] = c ^ x ^ C1[i];
        if (((c == 1) && (x == 1)) || ((x == 1) && (C1[i] == 1)) ||
((C1[i] == 1)
        && (c == 1)))
            c = 1;
        else
            c = 0;
    }
}
for (i = 0; i < 8; i++)
    printf("%d", ans[i]);
for (i = 7; i >= 0; i--)
{
    s = s + (pow(2, z) *ans[i]);
    z = z + 1;
}
if (((a < 0) && (b > 0)) || ((a > 0) && (b < 0)))
    printf("\nTHE ANSWER IN DECIMAL IS : -%d\n", s);
else
    printf("\nTHE ANSWER IN DECIMAL IS : %d\n", s);
}

```

Output:

```
?-----
? ENTER VALUE OF A: 6
? ENTER VALUE OF B: 4
?-----

THE BINARY EQUIVALENT OF 6 IS : 0110
THE BINARY EQUIVALENT OF 4 IS : 0100

-----
OPERATION          A      Q      Q'      M
INITIAL            0000    0100    0       0110
Shift              0000    0010    0       0110
Shift              0000    0001    0       0110
A:=A-M             1010    0001    0       0110
Shift              1101    0000    1       0110
A:=A+M             0011    0000    1       0110
Shift              0001    1000    0       0110
-----

THE ANSWER IN BINARY IS : 00011000
THE ANSWER IN DECIMAL IS : 24
Press any key to continue . . .
```

Conclusion: