

ASSIGNMENT 6:

ARITHMETIC DIVISION

U19CS012 [D-12]

1.) Write a C Code to Perform Division of Two Unsigned Binary Numbers Using Restoring Method.

Input: Two Binary Numbers

Output: Quotient and Remainder {In Binary & Equivalent Decimal}

Code:

```
#include <stdio.h>

// Maximum Number Of Bits
#define MAX 8

typedef long long int ll;

//Function Declarations

//Checks for Valid Dividendt and Divisor Entered By User
int check(ll Dividendt, ll divisor);

// 2's Complement for Negative Number
ll complement(ll n);

// Bit by Bit Addition
ll Bin_Add(ll n1, ll n2);

// Converts the Array to Equivalent Decimal(Long Long int)
//Eg: arr[] = 1011 -> return 1011(Number)
ll To_Decimal(int arr[MAX + 1]);

// Binary to decimal conversion [Accumulator]
int Bin_to_Dec_1(int arr[MAX + 1]);

// Binary to decimal conversion
int Bin_to_Dec(int arr[MAX]);

int main()
{
    printf("ARITHEMATIC DIVISION USING RESTORING METHOD\n");
    // Input Two Numbers
    ll q;
    printf("\nEnter Dividend in Binary [p/q] {p} [8 MAX]: ");
    scanf("%lld", &q);
```

```

11 b;
printf("Enter Divisor in Binary [p/q] {q} [8 MAX]: ");
scanf("%lld", &b);

// 3 Checks for Valid Input By User!

if (b == 0)
{
    printf("Division by Zero Error!\n");
    return 0;
}

if (b < 0 || b > 11111111 || q < 0 || q > 11111111)
{
    printf("Enter Valid {8-Bit Divisor(!=0) or Dividend} in Range -> [00000000(0)-11111111(255)]!\n");
    return 0;
}

if (check(q, b))
{
    printf("Enter Only 0 & 1 in Binary Format!\n");
    return 0;
}
// Checks Over -> User Entered Valid Input (Go Ahead!)

// Accumulator MAX = 1 + no of MAX of Dividendt
int A[MAX + 1] = {0};

// Q -> Dividendt Array of Binary
int Q[MAX] = {0};

// Complement of B -> Minus_B
11 Minus_B = complement(b);

int i;

for (i = MAX - 1; i >= 0; i--)
{
    int rem = q % 10;
    Q[i] = rem;
    q /= 10;
}

int count = MAX;

while (count)
{
    int j;

```

```

//>>> STEP 1 : SHIFT LEFT A,Q
// Shift Left A

for (j = 0; j < MAX; j++)
    A[j] = A[j + 1];

A[j] = Q[0];

//Shift Left Q
for (j = 0; j < MAX - 1; j++)
    Q[j] = Q[j + 1];

//>>> STEP 2 : A <- (A - B)

ll a = To_Decimal(A);

a = Bin_Add(a, Minus_B);

// -> Update the Accumulator Array
//Intialize Accumulator Array
for (j = 0; j < MAX + 1; j++)
    A[j] = 0;

j--;
// j -> MAX

while (a > 0)
{
    int rem = a % 10;
    A[j--] = rem;
    a /= 10;
}

//>>> STEP 3 : Check [A < 0]
//[NO] {Qo <- 1}
//[YES] -> {Qo <- 0; & A <- (A+B);}

if (A[0] == 0)
{
    // A is Positive [{Qo <- 1}]
    Q[MAX - 1] = 1;
}
else
{
    // A is Negative
    // {Qo <- 0}
    Q[MAX - 1] = 0;

    // {A <- (A+B)}
    ll a = To_Decimal(A);

```

```

        a = Bin_Add(a, b);

        // Update the Accumulator Array
        //Initialize Accumulator Array
        int k;
        for (k = 0; k < MAX + 1; k++)
            A[k] = 0;
        k--;
        // k -> MAX
        while (a > 0)
        {
            int rem = a % 10;
            A[k--] = rem;
            a /= 10;
        }
    }

    // Decrement the Count
    count = count - 1;
}

printf("\nA1.) Quotient Output          : ");

for (i = 0; i < MAX; i++)
{
    printf("%d", Q[i]);
}
printf("\n");

printf("A2.) Quotient Output [Final Answer] : %d\n", Bin_to_Dec(Q));

printf("B1.) Remainder Output          : ");

for (i = 0; i < MAX + 1; i++)
{
    printf("%d", A[i]);
}
printf("\n");

printf("B2.) Remainder Output [Final Answer] : %d\n", Bin_to_Dec_1(A));

return 0;
}

//Function Definations

int check(11 Dividend, 11 divisor)
{
    while (Dividend)
    {

```

```

    if ((Dividend % 10) > 1)
    {
        printf("Enter Valid Dividend!\n");
        return 1;
    }
    Dividend /= 10;
}
while (divisor)
{
    if ((divisor % 10) > 1)
    {
        printf("Enter Valid Divisor!\n");
        return 1;
    }
    divisor /= 10;
}
return 0;
}

11 Bin_Add(11 num1, 11 num2)
{
    11 sum = 0, carry = 0, pow = 1;

    while (num1 > 0 || num2 > 0)
    {
        sum += ((num1 % 10 + num2 % 10 + carry) % 2) * pow;

        // 1 1 1 -> 3/2 -> carry = 1 [11]
        // 1 1 0 -> 2/2 -> carry = 1 [10]
        // 1 0 0 -> 1/2 -> carry = 0
        carry = (num1 % 10 + num2 % 10 + carry) / 2;

        num1 /= 10;
        num2 /= 10;

        pow *= 10;
    }

    return sum;
}

11 To_Decimal(int arr[MAX + 1])
{
    long long ans = 0, i, pow = 1;
    for (i = MAX; i >= 0; i--)
    {
        ans += arr[i] * pow;
        pow *= 10;
    }
    return ans;
}

```

```

}

ll complement(ll n)
{
    ll ans = 0, pow = 1, arr[MAX + 1], i;
    for (i = MAX; i >= 0; i--)
    {
        int rem = n % 10;
        if (rem)
            arr[i] = 0;
        else
            arr[i] = 1;
        n /= 10;
    }
    for (i = MAX; i >= 0; i--)
    {
        ans += arr[i] * pow;
        pow *= 10;
    }
    return Bin_Add(ans, 1);
}

int Bin_to_Dec_1(int arr[MAX + 1])
{
    int ans = 0, pow = 1, i;
    for (i = MAX; i >= 0; i--)
    {
        ans += arr[i] * pow;
        pow *= 2;
    }
    return ans;
}

int Bin_to_Dec(int arr[MAX])
{
    int ans = 0, pow = 1, i;
    for (i = MAX - 1; i >= 0; i--)
    {
        ans += arr[i] * pow;
        pow *= 2;
    }
    return ans;
}

```

Test Cases:

1.) Invalid Input Entered by User

A.) [1010111(87) / 000000(0)] -> Divide by Zero Error!

```
ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 1010111
Enter Divisor in Binary [p/q] {q} [8 MAX]: 000000
Division by Zero Error!
```

B.) [111101011(491) / 010101(21)] -> Dividend not 8 bit [0-255]

```
ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 111101011
Enter Divisor in Binary [p/q] {q} [8 MAX]: 010101
Enter Valid {8-Bit Divisor(!=0) or Dividend} in Range -> [00000000(0)-11111111(255)]!
```

C.) [001210 / 000101] -> Binary Number only "0 & 1"

```
ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 001210
Enter Divisor in Binary [p/q] {q} [8 MAX]: 000101
Enter Valid Dividend!
Enter Only 0 & 1 in Binary Format!
```

2.) Valid Input Entered by User

A.) [11111111(255) / 11001(25)] -> $255 = 25 \times \{10[1010]\} + \{5[101]\}$

```
ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 11111111
Enter Divisor in Binary [p/q] {q} [8 MAX]: 11001

A1.) Quotient Output           : 00001010
A2.) Quotient Output [Final Answer] : 10
B1.) Remainder Output          : 000000101
B2.) Remainder Output [Final Answer] : 5
```

B.) $[11110111(247) / 1101(13)] \rightarrow 247 = 13 * \{19[10011]\} + \{0[0]\}$

ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 11110111

Enter Divisor in Binary [p/q] {q} [8 MAX]: 1101

A1.) Quotient Output : 00010011

A2.) Quotient Output [Final Answer] : 19

B1.) Remainder Output : 00000000

B2.) Remainder Output [Final Answer] : 0

C.) $[1111000(120) / 1111000(120)] \rightarrow 120 = 120 * \{1[1]\} + \{0[0]\}$

ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 1111000

Enter Divisor in Binary [p/q] {q} [8 MAX]: 1111000

A1.) Quotient Output : 00000001

A2.) Quotient Output [Final Answer] : 1

B1.) Remainder Output : 00000000

B2.) Remainder Output [Final Answer] : 0

D.) $[1011(11) / 1111001(121)] \rightarrow 11 = 121 * \{0[0]\} + \{11[1011]\}$

ARITHMETIC DIVISION USING RESTORING METHOD

Enter Dividend in Binary [p/q] {p} [8 MAX]: 1011

Enter Divisor in Binary [p/q] {q} [8 MAX]: 1111001

A1.) Quotient Output : 00000000

A2.) Quotient Output [Final Answer] : 0

B1.) Remainder Output : 000001011

B2.) Remainder Output [Final Answer] : 11