ASSIGNMENT 6:

ARITHEMETIC DIVISION

*U19CS012 [D-12]*

1.) Write a C Code to Perform Division of Two Unsigned Binary Numbers Using Non-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);

    ll 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 Dividend*

    int A[MAX + 1] = {0};

*// Q -> Dividend Array of Binary*

    int Q[MAX] = {0};

*// Complement of B -> Minus\_B*

    ll 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 : Check [A < 0]*

*//[NO]  ->  {A <- (A-B);}*

*//[YES] ->  {A <- (A+B);}*

*if* (A[0] == 0)

        {

*// A is Positive [{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;

            }

        }

*else*

        {

*// A is Negative [{A <- (A+B);}]*

            ll a = To\_Decimal(A);

            a = Bin\_Add(a, b);

*// Update the Accumulator Array*

*//Intialize 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;

            }

        }

*//>>> STEP 3 : Check [A < 0]*

*//[NO]  ->  {Qo <- 1;}*

*//[YES] ->  {Qo <- 0;}*

*if* (A[0] == 0)

        {

*// A is Positive [{Qo <- 1}]*

            Q[MAX - 1] = 1;

        }

*else*

        {

*// A is Negative*

*// {Qo <- 0}*

            Q[MAX - 1] = 0;

        }

*// Decrement the Count*

        count = count - 1;

    }

*//>>> STEP 4 : Check [A < 0]*

*//[NO]  ->  {STOP}*

*//[YES] ->  {A <- (A+B);}*

*if* (A[0] == 0)

    {

*// STOP*

    }

*else*

    {

*// A is Negative [{A <- (A+B);}]*

        ll a = To\_Decimal(A);

        a = Bin\_Add(a, b);

*// Update the Accumulator Array*

*//Intialize 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;

        }

    }

    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(ll Dividend, ll 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;

}

ll Bin\_Add(ll num1, ll num2)

{

    ll 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;

}

ll 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;

    }

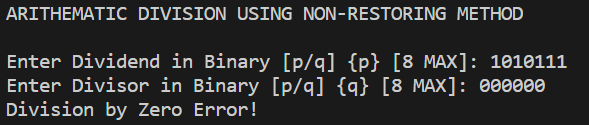
*return* ans;

}

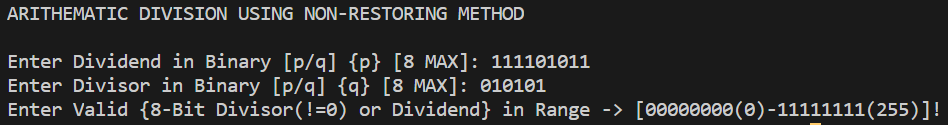
**Test Cases:**

1.) Invalid Input Entered by User

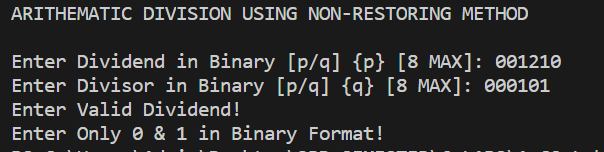
A.) [1010111(87) / 000000(0)] -> Divide by Zero Error!



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

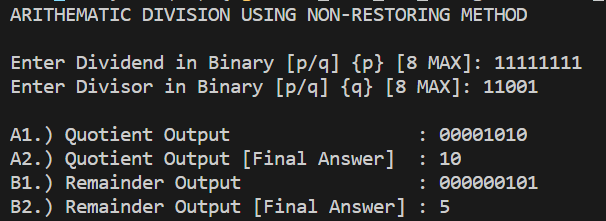


C.) [001210 / 000101] -> Binary Number only “0 & 1”

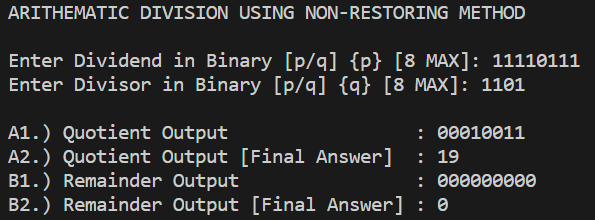


2.) Valid Input Entered by User

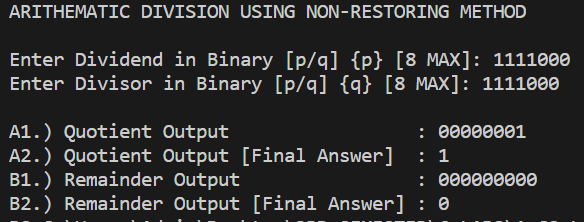
A.) [11111111(255) / 11001(25)] -> 255 = 25\*{10[1010]} + {5[101]}



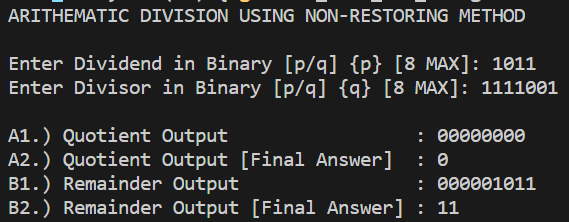
B.) [11110111(247) / 1101(13)] -> 247 = 13\*{19[10011]} + {0[0]}



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



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



2A.) Write a C Code to Perform Division of Two Signed 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 7

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\n");

    printf("SMR = Signed Magnitude Representation\n");

    printf("Sign : [1]-> Negative, [0]-> Positive\n");

    printf("Magnitude : Binary Form of Number{[-127 to 127]} [1010 <- {10}]\n");

*// Input Two Numbers*

*// q hold only 7 bits Divident*

    ll q;

*// b hold only 7 bits Divident*

    ll b;

    int q1;

    printf("\nEnter Dividend in SMR Binary [p/q] {p} [S(1) M(7)]: ");

    scanf("%d %lld", &q1, &q);

    int b1;

    printf("Enter Divisor in SMR Binary [p/q] {q}  [S(1) M(7)]: ");

    scanf("%d %lld", &b1, &b);

*//Sign -> -ve = 1*

*//Sign -> +ve = 0*

    int dividend\_sign = 0;

*if* (q1 == 1)

    {

*// Sign is -ve since q1 = 1*

        dividend\_sign = 1;

    }

*else*

    {

*if* (q1 > 1)

        {

*// Invalid Sign*

            printf("Enter Valid Sign Bit of Divident!\n");

            printf("Enter Only 0 & 1 in Binary Format!\n");

*return* 0;

        }

*else*

        {

*// Sign is +ve since q1 = 0*

            dividend\_sign = 0;

        }

    }

    int divisor\_sign = 0;

*if* (b1 == 1)

    {

*// Sign is -ve since b1 = 1*

        divisor\_sign = 1;

    }

*else*

    {

*if* (b1 > 1)

        {

*// Invalid Sign*

            printf("Enter Valid Sign Bit of Divisor!\n");

            printf("Enter Only 0 & 1 in Binary Format!\n");

*return* 0;

        }

*else*

        {

*// Sign is +ve since b1 = 0*

            divisor\_sign = 0;

        }

    }

*// 3 Checks for Valid Input By User!*

*if* (b == 0)

    {

        printf("Division by Zero Error!\n");

*return* 0;

    }

*if* (b < 0 || b > 1111111 || q < 0 || q > 1111111)

    {

        printf("Enter Valid {7-Bit Divisor(!=0) or Dividend} in Range -> [1\_1111111(-127)-0\_1111111(127)]!\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 Dividend*

    int A[MAX + 1] = {0};

*// Q -> Dividend Array of Binary*

    int Q[MAX] = {0};

*// Complement of B -> Minus\_B*

    ll 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*

*//Intialize 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 [SMR]           : ");

*if* ((dividend\_sign ^ divisor\_sign) == 0)

    {

*// If the both[divisor & divident] are positive or both negative,*

*//  then the remainder will be positive.*

        printf("%d ", 0);

    }

*else*

    {

*// Otherwise, it will be negative.*

        printf("%d ", 1);

    }

*for* (i = 0; i < MAX; i++)

    {

        printf("%d", Q[i]);

    }

    printf("\n");

    ll quo\_decimal = Bin\_to\_Dec(Q);

*if* ((dividend\_sign ^ divisor\_sign) == 0)

    {

*// If the both[divisor & divident] are positive or both negative,*

*//  then the remainder will be positive.*

        quo\_decimal \*= 1;

    }

*else*

    {

*// Otherwise, it will be negative.*

        quo\_decimal \*= -1;

    }

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

    printf("B1.) Remainder Output [SMR]          : ");

*if* (dividend\_sign == 0)

    {

*// If the dividend is positive, then the remainder will be positive.*

        printf("%d ", 0);

    }

*else*

    {

*// If the dividend is negative, then the remainder will be negative.*

        ll rem = Bin\_to\_Dec\_1(A);

*if* (rem != 0)

            printf("%d ", 1);

*else*

        {

            printf("%d ", 0);

        }

    }

*for* (i = 0; i < MAX + 1; i++)

    {

        printf("%d", A[i]);

    }

    printf("\n");

    ll rem\_decimal = Bin\_to\_Dec\_1(A);

*if* (dividend\_sign == 0)

    {

*// If the dividend is positive, then the remainder will be positive.*

        rem\_decimal \*= 1;

    }

*else*

    {

*// If the dividend is negative, then the remainder will be negative.*

        rem\_decimal \*= -1;

    }

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

*return* 0;

}

*//Function Definations*

int check(ll Dividend, ll 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;

}

ll Bin\_Add(ll num1, ll num2)

{

    ll 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;

}

ll 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;

    }

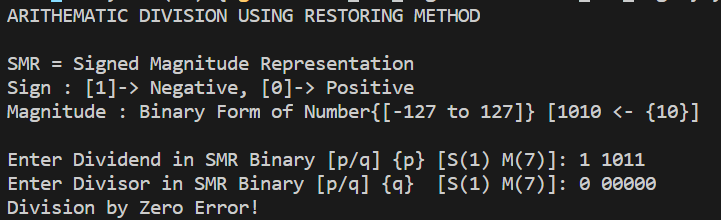
*return* ans;

}

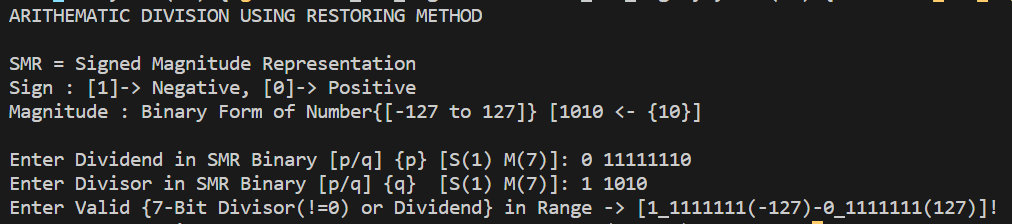
**Test Cases:**

1.) Invalid Input Entered by User

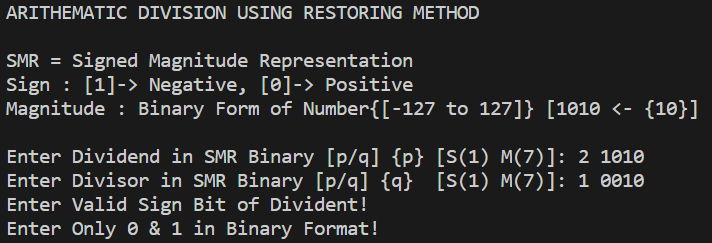
A.) [1 1011 (-11) / 0 00000 (0)] -> Divide by Zero Error!



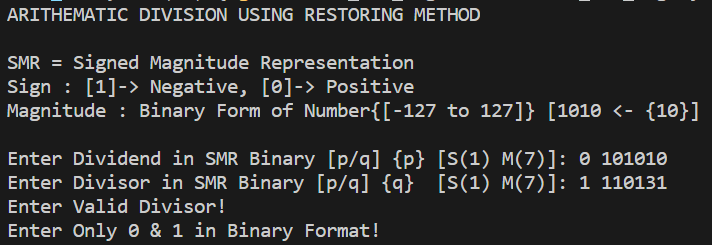
B.) [0 11111110 (254) / 1 1010 (-10)] -> Dividend not 7 bit [-127 to +127]



C.) [2 1010 (?) / 1 0010 (-2)] -> Binary Number only “0 & 1”

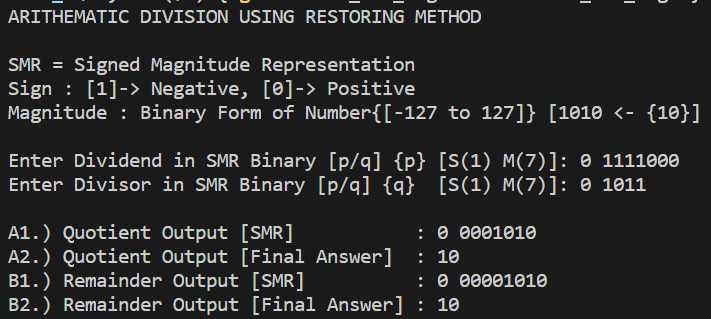


D.) [0 101010 (42) / 1 110131 (?)] -> Binary Number only “0 & 1”

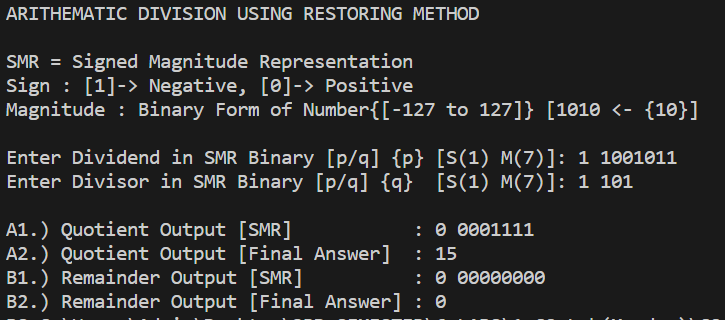


2.) Valid Input Entered by User

A.) [0 1111000 (+120) /0 1011 (+11)] -> 120 = 11\*{10[1010]} + {10[1010]}

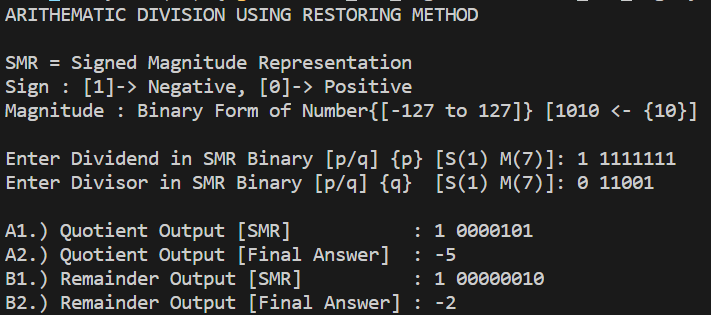


B.) [1 1001011 (-75) / 1 101 (-5)] -> -75 = -5\*{15[1111]} + {0[0000]}



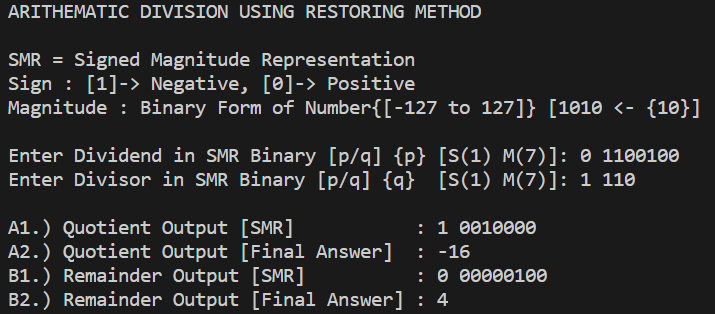
C.) [1 1111111 (-127) / 0 11001 (+25)]

-> -127 = 25\*{-5[1 0000101]} + {-2[1 00000010]}



D.) [0 1100100 (100) / 1 110 (-6)]

-> 100 = -6\*{-16[1 0010000]} + {4[0 00000100]}



2B.) Write a C Code to Perform Division of Two Signed Binary Numbers Using Non-Restoring Method.

Input: Two Binary Numbers

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

**Code:**

*#include* <stdio.h>

*// Maximum Number Of Bits*

*#define* MAX 7

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 NON-RESTORING METHOD\n");

    printf("SMR = Signed Magnitude Representation\n");

    printf("Sign : [1]-> Negative, [0]-> Positive\n");

    printf("Magnitude : Binary Form of Number{[-127 to 127]} [1010 <- {10}]\n");

*// q hold only 7 bits Divident*

    ll q;

*// b hold only 7 bits Divident*

    ll b;

    int q1;

    printf("\nEnter Dividend in SMR Binary [p/q] {p} [S(1) M(7)]: ");

    scanf("%d %lld", &q1, &q);

    int b1;

    printf("Enter Divisor in SMR Binary [p/q] {q}  [S(1) M(7)]: ");

    scanf("%d %lld", &b1, &b);

*//Sign -> -ve = 1*

*//Sign -> +ve = 0*

    int dividend\_sign = 0;

*if* (q1 == 1)

    {

*// Sign is -ve since q1 = 1*

        dividend\_sign = 1;

    }

*else*

    {

*if* (q1 > 1)

        {

*// Invalid Sign*

            printf("Enter Valid Sign Bit of Divident!\n");

            printf("Enter Only 0 & 1 in Binary Format!\n");

*return* 0;

        }

*else*

        {

*// Sign is +ve since q1 = 0*

            dividend\_sign = 0;

        }

    }

    int divisor\_sign = 0;

*if* (b1 == 1)

    {

*// Sign is -ve since b1 = 1*

        divisor\_sign = 1;

    }

*else*

    {

*if* (b1 > 1)

        {

*// Invalid Sign*

            printf("Enter Valid Sign Bit of Divisor!\n");

            printf("Enter Only 0 & 1 in Binary Format!\n");

*return* 0;

        }

*else*

        {

*// Sign is +ve since b1 = 0*

            divisor\_sign = 0;

        }

    }

*// 3 Checks for Valid Input By User!*

*if* (b == 0)

    {

        printf("Division by Zero Error!\n");

*return* 0;

    }

*if* (b < 0 || b > 1111111 || q < 0 || q > 1111111)

    {

        printf("Enter Valid {7-Bit Divisor(!=0) or Dividend} in Range -> [1\_1111111(-127)-0\_1111111(127)]!\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 Dividend*

    int A[MAX + 1] = {0};

*// Q -> Dividend Array of Binary*

    int Q[MAX] = {0};

*// Complement of B -> Minus\_B*

    ll 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 : Check [A < 0]*

*//[NO]  ->  {A <- (A-B);}*

*//[YES] ->  {A <- (A+B);}*

*if* (A[0] == 0)

        {

*// A is Positive [{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;

            }

        }

*else*

        {

*// A is Negative [{A <- (A+B);}]*

            ll a = To\_Decimal(A);

            a = Bin\_Add(a, b);

*// Update the Accumulator Array*

*//Intialize 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;

            }

        }

*//>>> STEP 3 : Check [A < 0]*

*//[NO]  ->  {Qo <- 1;}*

*//[YES] ->  {Qo <- 0;}*

*if* (A[0] == 0)

        {

*// A is Positive [{Qo <- 1}]*

            Q[MAX - 1] = 1;

        }

*else*

        {

*// A is Negative*

*// {Qo <- 0}*

            Q[MAX - 1] = 0;

        }

*// Decrement the Count*

        count = count - 1;

    }

*//>>> STEP 4 : Check [A < 0]*

*//[NO]  ->  {STOP}*

*//[YES] ->  {A <- (A+B);}*

*if* (A[0] == 0)

    {

*// STOP*

    }

*else*

    {

*// A is Negative [{A <- (A+B);}]*

        ll a = To\_Decimal(A);

        a = Bin\_Add(a, b);

*// Update the Accumulator Array*

*//Intialize 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;

        }

    }

    printf("\nA1.) Quotient Output [SMR]           : ");

*if* ((dividend\_sign ^ divisor\_sign) == 0)

    {

*// If the both[divisor & divident] are positive or both negative,*

*//  then the remainder will be positive.*

        printf("%d ", 0);

    }

*else*

    {

*// Otherwise, it will be negative.*

        printf("%d ", 1);

    }

*for* (i = 0; i < MAX; i++)

    {

        printf("%d", Q[i]);

    }

    printf("\n");

    ll quo\_decimal = Bin\_to\_Dec(Q);

*if* ((dividend\_sign ^ divisor\_sign) == 0)

    {

*// If the both[divisor & divident] are positive or both negative,*

*//  then the remainder will be positive.*

        quo\_decimal \*= 1;

    }

*else*

    {

*// Otherwise, it will be negative.*

        quo\_decimal \*= -1;

    }

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

    printf("B1.) Remainder Output [SMR]          : ");

*if* (dividend\_sign == 0)

    {

*// If the dividend is positive, then the remainder will be positive.*

        printf("%d ", 0);

    }

*else*

    {

*// If the dividend is negative, then the remainder will be negative.*

        ll rem = Bin\_to\_Dec\_1(A);

*if* (rem != 0)

            printf("%d ", 1);

*else*

        {

            printf("%d ", 0);

        }

    }

*for* (i = 0; i < MAX + 1; i++)

    {

        printf("%d", A[i]);

    }

    printf("\n");

    ll rem\_decimal = Bin\_to\_Dec\_1(A);

*if* (dividend\_sign == 0)

    {

*// If the dividend is positive, then the remainder will be positive.*

        rem\_decimal \*= 1;

    }

*else*

    {

*// If the dividend is negative, then the remainder will be negative.*

        rem\_decimal \*= -1;

    }

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

*return* 0;

}

*//Function Definations*

int check(ll Dividend, ll 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;

}

ll Bin\_Add(ll num1, ll num2)

{

    ll 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;

}

ll 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;

    }

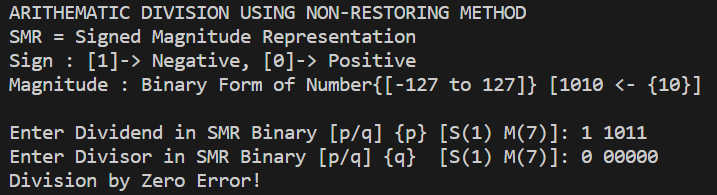
*return* ans;

}

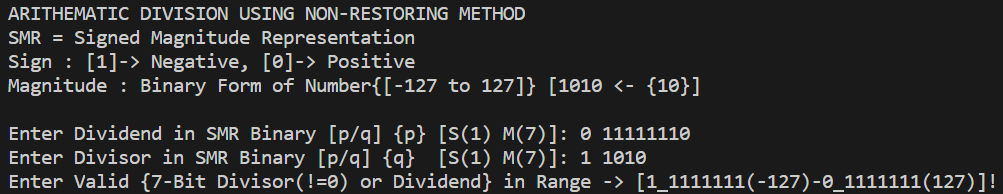
**Test Cases:**

1.) Invalid Input Entered by User

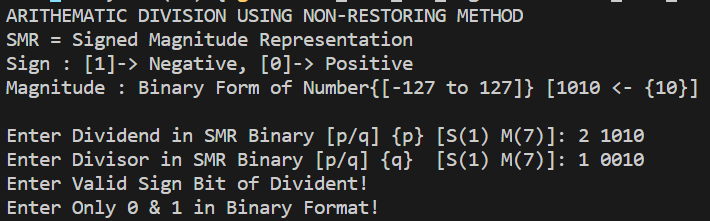
A.) [1 1011 (-11) / 0 00000 (0)] -> Divide by Zero Error!



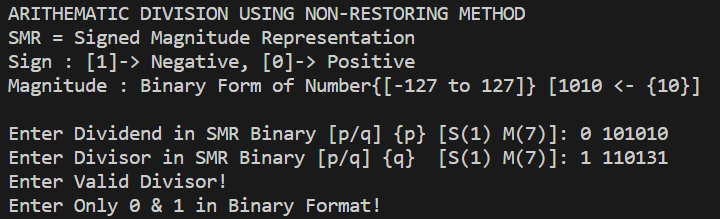
B.) [0 11111110 (254) / 1 1010 (-10)] -> Dividend not 7 bit [-127 to +127]



C.) [2 1010 (?) / 1 0010 (-2)] -> Binary Number only “0 & 1”

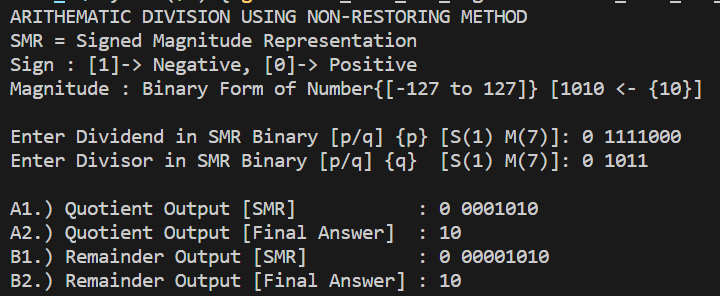


D.) [0 101010 (42) / 1 110131 (?)] -> Binary Number only “0 & 1”

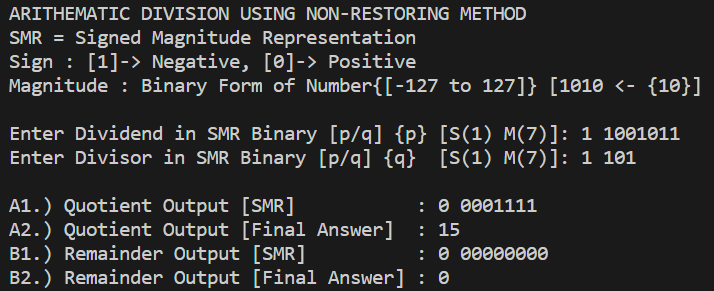


2.) Valid Input Entered by User

A.) [0 1111000 (+120) /0 1011 (+11)] -> 120 = 11\*{10[1010]} + {10[1010]}

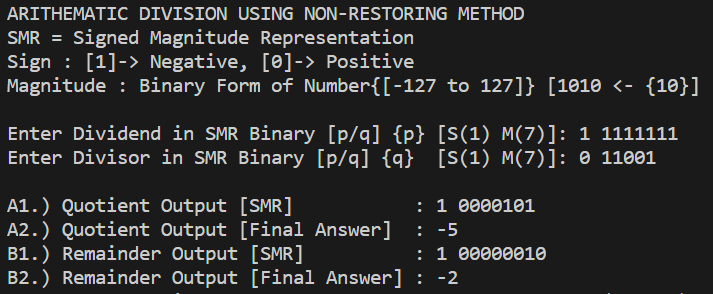


B.) [1 1001011 (-75) / 1 101 (-5)] -> -75 = -5\*{15[1111]} + {0[0000]}



C.) [1 1111111 (-127) / 0 11001 (+25)]

-> -127 = 25\*{-5[1 0000101]} + {-2[1 00000010]}



D.) [0 1100100 (100) / 1 110 (-6)]

-> 100 = -6\*{-16[1 0010000]} + {4[0 00000100]}

