Name: Jigar Siddhpura **SAPID:** 60004200155

DIV: C/C2 **Branch:** Computer Engineering

POA EXPERIMENT 2

Jigar Siddhpura
60004210155

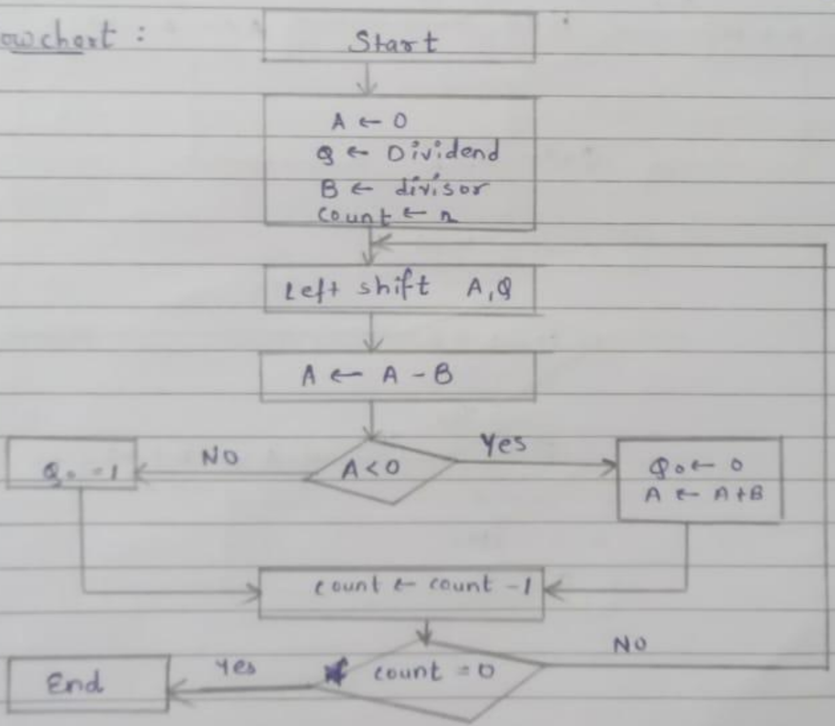
Experiment 2

Aim: To study & implement restoring & non-restoring division algorithm.

Theory:

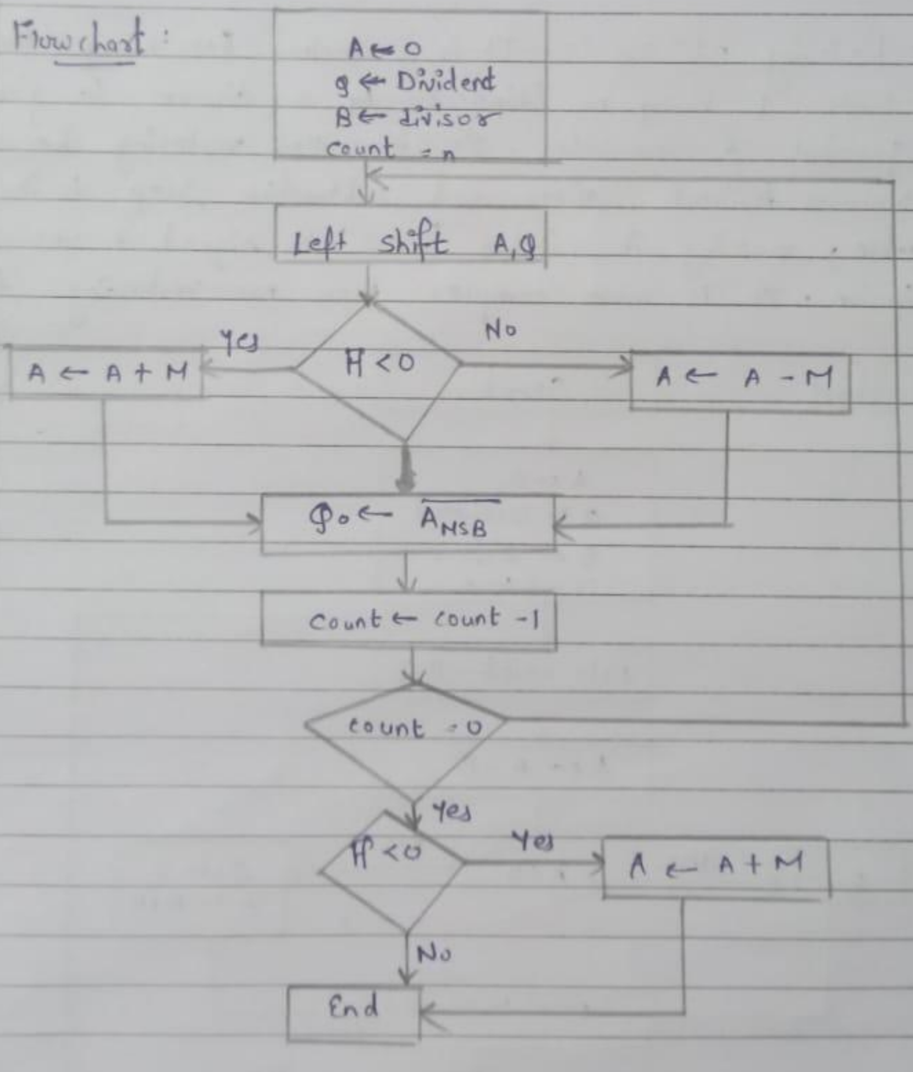
1. Restoring Division :- It is a method for binary division to divide 1 binary no. (dividend) by a divisor to give ~~get~~ quotient & remainder. It is called restoring because it restores 'dividend', after each subtraction step to its orignal value, making it suitable for both signed & unsigned division. It is more complex than non-restoring division.

Flowchart :



Non-restoring Division :- It is another method for binary division. Here it doesn't restore the orignal value of dividend after each subtraction, making it more suitable for hardware implementations where subtraction can be more complex. It's commonly used in digital arithmetic circuits.

Flowchart :



Q - Restoring division $\left(\frac{14}{Q} \Big| \frac{5}{M}\right)$

→ $Q = (14)_{10} = (01110)_2$

$B = (5)_{10} = (00101)_2$

Count = $(n+1)$ bits

= 5 bits = 5

$-B = (11011)_2$

$A = 00000$

00101 → 1's comp.

+ 11010

11011

A	Q	operation	count
00000	01110	A.L.S.	
00000	1110$\square$	$A \leftarrow A - B$	4
		00000	
		+ 11011	
11011	1110$\square$	11011	
00000	11100	$Q_0 \leftarrow 0$	
		$A \leftarrow A + B$	
		11011	
		+ 00101	
		$\boxed{1}$00000	
00000	11100	A.L.S	
00001	1100$\square$	$A \leftarrow A - B$	3
		00001	
		+ 11011	
11100	1100$\square$	11100	
00001	11000	$Q_0 = 0$ & $A \leftarrow A+B$	
		11100	
		+ 00101	
		$\boxed{1}$00001	
00001	11000	A.L.S	
00011	1000_	$A \leftarrow A - B$	
		00011	
		+ 11011	2
11110	1000_	11110	
00011	1000$\boxed{0}$	$Q_0 \leftarrow 0$ & $A \leftarrow A+B$	
		11110	
		+ 00101	
		$\boxed{1}$00011	

00011	10000	A.L.S	
00111	0000$\boxed{\ }$	$A \leftarrow A - B$ 00111 + 11011 $\boxed{1}$00010	1
00010	0000$\boxed{\ }$	$q_0 = 1$	
00010	00001		

00010	00001	A.L.S.	
00100	0001_	$A \leftarrow A - B$ 00100 + 11011 11111	0
11111	0001_	$q_0 = 0$ & $A \leftarrow A + B$ 11111 + 00101 $\boxed{1}$00100	
00100	00010		

Answer : Remainder : $A \rightarrow (00100)_2 = (4)_{10}$
Quotient : $Q \rightarrow (00010)_2 = (2)_{10}$

Answer : Quotient $= (0011)_2 = (3)_{10}$

Remainder $= \cancel{(00011)} \; (00011)_2 = (3)_{10}$

Conclusion : Hence we have implement both restoring & non-restoring division to perform binary division.

Restoring Division Code :

```python
def binary_addition(a,b):
    """a,b are binary strings"""
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)

    # Initialize the result
    result = ''

    # Initialize the carry
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = '1' + result

    result = result.zfill(max_len)

   return result[-max_len:]

def complement(b):
    # 2's complement of binary number
    b_comp = ""
    for i in b:
        if i=="1": b_comp += "0"
        elif i=="0": b_comp += "1"
    b_comp = binary_addition(b_comp,"1")
   return b_comp

def binary_subtraction(a,b):
    """a,b are binary strings"""
    b_complement = complement(b)
    max_len = len(a)
   return binary_addition(a,b_complement.zfill(max_len))




def ALS(a,q):
    """all are binary strings"""
    # x = Q not
    a = a[1:]+q[0]
    q = q[1:]+"_"
   return a,q

def isNegative(bin_number):
   return bin_number[0] == "1"
```

```
def restoring_division(dividend,divisor,count,A="00000",Q1="0"):
    binary_dividend  = bin(dividend)[2:].zfill(count) if dividend>0 else 
bin(dividend)[3:].zfill(count)
    binary_divisor = bin(divisor)[2:].zfill(count+1) if divisor>0 else 
bin(dividend)[3:].zfill(count+1)
    print(f"Dividend Q = {binary_dividend}, divisor B = {binary_divisor}, A = {A}, 
count = {count}\n")
    print("Count\tA\tQ\tOperation")
   print("----------------------------------------")

    while(count!=0):
        print(f"{count}\t{A}\t{binary_dividend}\tbefore ALS")
        A,binary_dividend = ALS(A,binary_dividend)
        print(f"{count}\t{A}\t{binary_dividend}\tA <- A-B")
        A = binary_subtraction(A,binary_divisor)
        case = isNegative(A)
        # print(f"A,Q <- {case}")
        if(case):
            binary_dividend = binary_dividend.replace("_","0")
            A = binary_addition(A,binary_divisor)
            print(f"{count}\t{A}\t{binary_dividend}\tQ. = 0 & A<-A+B")
        else:
            # A = binary_subtraction(A,binary_divisor)
            binary_dividend = binary_dividend.replace("_","1")
            print(f"{count}\t{A}\t{binary_dividend}\tQ. = 1")
        print(f"{count}\t{A}\t{binary_dividend}")
        count -= 1
        print("----------------------------------------")

    remainder = int(A,base=2)
    quotient = int(binary_dividend,base=2)

   print(f"Remainder = {remainder}, Quotient = {quotient}")

restoring_division(int(input("Enter Dividend = ")),int(input("Enter Divisor = ")),4)
```

Output :

```
PS D:\SEM 5\POA\EXPERIMENTS> python -u "d:\SEM 5\POA\EXPERIMENTS\restoringDivision.py"
Enter Dividend = 11
Enter Divisor = 3
Dividend Q = 1011, divisor B = 00011, A = 00000, count = 4

Count   A       Q       Operation
----------------------------------------
4       00000   1011    before ALS
4       00001   011_    A <- A-B
4       00001   0110    Q. = 0 & A<-A+B
4       00001   0110
----------------------------------------
3       00001   0110    before ALS
3       00010   110_    A <- A-B
3       00010   1100    Q. = 0 & A<-A+B
3       00010   1100
----------------------------------------
2       00010   1100    before ALS
2       00101   100_    A <- A-B
2       00010   1001    Q. = 1
2       00010   1001
----------------------------------------
1       00010   1001    before ALS
1       00101   001_    A <- A-B
1       00010   0011    Q. = 1
1       00010   0011
----------------------------------------
Remainder = 2, Quotient = 3
PS D:\SEM 5\POA\EXPERIMENTS>
```

Non-restoring Division Code :

```
def binary_addition(a,b):
    """a,b are binary strings"""
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)

    # Initialize the result
    result = ''

    # Initialize the carry
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = '1' + result

    result = result.zfill(max_len)

   return result[-max_len:]

def complement(b):
    # 2's complement of binary number
    b_comp = ""
    for i in b:
        if i=="1": b_comp += "0"
        elif i=="0": b_comp += "1"
    b_comp = binary_addition(b_comp,"1")
   return b_comp

def binary_subtraction(a,b):
    """a,b are binary strings"""
    b_complement = complement(b)
    max_len = len(a)
   return binary_addition(a,b_complement.zfill(max_len))

def ALS(a,q):
    """all are binary strings"""
    # x = Q not
    a = a[1:]+q[0]
    q = q[1:]+"_"
   return a,q

def isNegative(bin_number):
   return bin_number[0] == "1"

def nonRestoringDivision(dividend,divisor,count,A="00000"):
    binary_dividend  = bin(dividend)[2:].zfill(count) if dividend>0 else
bin(dividend)[3:].zfill(count)
```

```
    binary_divisor = bin(divisor)[2:].zfill(count+1) if divisor>0 else
bin(dividend)[3:].zfill(count+1)
    print(f"Dividend Q = {binary_dividend}, divisor B = {binary_divisor}, A = {A},
count = {count}\n")
    print("Count\tA\tQ\tOperation")
    print("----------------------------------------")

    while(count!=0):
        print(f"{count}\t{A}\t{binary_dividend}\tbefore ALS")
        A,binary_dividend = ALS(A,binary_dividend)
        case = isNegative(A)

        if(case):
            print(f"{count}\t{A}\t{binary_dividend}\tA <- A+B")
            A = binary_addition(A,binary_divisor)
        else:
            print(f"{count}\t{A}\t{binary_dividend}\tA <- A-B")
            A = binary_subtraction(A,binary_divisor)

        print(f"{count}\t{A}\t{binary_dividend}\tQ. = ~(Amsb)")
        binary_dividend = binary_dividend.replace("_",str(int(not(int(A[0])))))
        print(f"{count}\t{A}\t{binary_dividend}")

        count -= 1
        print("----------------------------------------")

    if isNegative(A):
        A = binary_addition(A,binary_divisor)

    remainder = int(A,base=2)
    quotient = int(binary_dividend,base=2)

    print(f"Remainder = {remainder}, Quotient = {quotient}")

nonRestoringDivision(int(input("Enter Dividend = ")),int(input("Enter Divisor =
")),4)
```

Output :

```
PS D:\SEM 5\POA\EXPERIMENTS> python -u "d:\SEM 5\POA\EXPERIMENTS\restoringDivision.py"
Enter Dividend = 11
Enter Divisor = 3
Dividend Q = 1011, divisor B = 00011, A = 00000, count = 4

Count   A       Q       Operation
----------------------------------------
4       00000   1011    before ALS
4       00001   011_    A <- A-B
4       00001   0110    Q. = 0 & A<-A+B
4       00001   0110
----------------------------------------
3       00001   0110    before ALS
3       00010   110_    A <- A-B
3       00010   1100    Q. = 0 & A<-A+B
3       00010   1100
----------------------------------------
2       00010   1100    before ALS
2       00101   100_    A <- A-B
2       00010   1001    Q. = 1
2       00010   1001
----------------------------------------
1       00010   1001    before ALS
1       00101   001_    A <- A-B
1       00010   0011    Q. = 1
1       00010   0011
----------------------------------------
Remainder = 2, Quotient = 3
PS D:\SEM 5\POA\EXPERIMENTS>
```