NAME: ALISTAIR SHAAN SALDANHA

SAPID: 60009200024

BRANCH: DATA SCIENCE

BATCH: K1

Restoring Division

In [8]:

```
def to_binary(num):
  if(num >= 0):
    return(bin(num)[2:].zfill(n+1)) #[2:] - to remove 0b
  elif(num < 0):
    return (bin(abs(num))[2:].zfill(n+1))
# print(to_binary(9))
# print(to_binary(-2))
```

ADDING TWO BINARY NUMBERS

In [9]:

```
def add(x1, x2, n):
  result = ''
  carry = 0
  for i in range(n, -1, -1):
      carry += 1 if x1[i] == '1' else 0
      carry += 1 if x2[i] == '1' else 0
      result = ('1' if carry % 2 == 1 else '0') + result
      carry = 0 if carry < 2 else 1
  return(result.zfill(n))
add('11100','00001',4)
```

Out[9]:

```
'11101'
```

TWO'S COMPLEMENT

In [10]:

```
def twos_comp(num,n):
  x = ''
  one_add = '0'*(n)+'1'
  one_c = map(lambda x: '0' if x=='1' else '1',num) # 00011 --> 11100
  for i in one_c:
    x += i
  two_c = add(x,one_add,n)
  return two_c
print(twos_comp('00011',4))
```

```
11101
```

SHIFT LEFT

In [11]:

```
def shl(bits):
```

```
    x = bits[0]
    shift_bits = bin(int(('0b' + bits),2) << 1)[2:] # using the left right operator (adv
antage in string operations)
    bits = shift_bits.zfill(2*n+1)[:-1]  # zfill() is used to fill the starting places w
ith 0s
    return bits
# print(shl('0101'))
# print(shl('0000010001'))
```

DISPLAY THE CALCULATION TABLE

In [12]:

```
def table(bits,count,oper,n): # Display Function
    A = bits[:n+1]
    q_bin = bits[n+1:]
    print(f'  {count}     {A}   {q_bin}  {oper}')
```

In [13]:

```
def RestoringDiv(bits,count,n):
    # SHIFT LEFT
    A = bits[:n+1]
    q_bin = bits[n+1:2*n+1]
    bits = shl(bits)
    # print(f"Shift Left: {bits}")
    table(bits,count,'   Shift',n)
    # A = A - M
    A = bits[:n+1]
    q_bin = bits[n+1:2*n+1]
    A = add(A,minus_M,n)
    # print(f'A: {A}')
    # print(f'q_bin:{q_bin}')
    # CHECK
    if(A[0] == '0'):
        q_bin = q_bin + '1'
        bits = A + q_bin
        table(bits,count,'  A=A-M',n)
    else:
        q_bin = q_bin + '0'
        bits = A + q_bin
        table(bits,count,'  A=A-M',n)
        A = add(A,plus_M,n)
        bits = A + q_bin
        table(bits,count,'  A=A+M(Restore)',n)
    count -= 1
    if(count > 0):
        RestoringDiv(bits,count,n)
    else:
        A = bits[:n+1]
        q_bin = bits[n+1:2*n+1]
        print(f'A: {A}, Q: {q_bin}')
        remainder = int(('0b' + A),2)
        quotient = int(('0b' + q_bin),2)
        print(f"Remainder: {remainder} , Quotient: {quotient}")
```

RESTORING DIVISION ALGORITHM

In [14]:

```
# Check for m and q and accordingly get +M,-M,Q
q = 17 # Dividend
m = 3  # Divisor
n = 4  # Number of bits

# M - +ve Q - +ve
if(m>0 and q>0):
    plus_M = to_binary(m)
    minus_M = twos_comp(plus_M,n)
    x = list(to_binary(q))
```

```
    q_bin = ''
    if(x[0] == '0'):
        del x[0]
    for i in x:
        q_bin += i
A = '0'*(n+1)
count=n

# Trace the table
print("-----------RESTORING DIVISION-----------\n")
print('count  ','  A  ','   q  ','   Operation')

# Initialisation
bits = A + q_bin
# print(bits)
table(bits,count,'  Initialisation',n)
RestoringDiv(bits,count,n)
```

```
-----------RESTORING DIVISION-----------

count      A       q       Operation
  4      00000   10001     Initialisation
  4      00010   001       Shift
  4      11111   0010      A=A-M
  4      00010   0010      A=A+M(Restore)
  3      00100   010       Shift
  3      00001   0101      A=A-M
  2      00010   101       Shift
  2      11111   1010      A=A-M
  2      00010   1010      A=A+M(Restore)
  1      00101   010       Shift
  1      00010   0101      A=A-M
A: 00010, Q: 0101
Remainder: 2 , Quotient: 5
```

NAME: ALISTAIR SHAAN SALDANHA

SAPID: 60009200024

BRANCH: DATA SCIENCE

BATCH: K1

Non Restoring Division

In [15]:

```
# Check for m and q and accordingly get +M,-M,Q
q = 10 # Dividend
m = 3  # Divisor
count = n = 4  # Number of bits

A = '0'*(n+1)
```

CONVERSION INTO BINARY

In [16]:

```
def to_binary(num):
  if(num >= 0):
    return(bin(num)[2:].zfill(n+1)) #[2:] - to remove 0b
  elif(num < 0):
    return (bin(abs(num))[2:].zfill(n+1))
```

```
# print(to_binary(9))
# print(to_binary(-2))
```

ADDING TWO BINARY NUMBERS

In [17]:

```
def add(x1, x2, n):
  result = ''
  carry = 0
  for i in range(n, -1, -1):
      carry += 1 if x1[i] == '1' else 0
      carry += 1 if x2[i] == '1' else 0
      result = ('1' if carry % 2 == 1 else '0') + result
      carry = 0 if carry < 2 else 1
  return(result.zfill(n))
# add('11100','00001',n)
```

TWO'S COMPLEMENT

In [18]:

```
def twos_comp(num,n):
  x = ''
  one_add = '0'*(n)+'1'
  one_c = map(lambda x: '0' if x=='1' else '1',num) # 00011 --> 11100
  for i in one_c:
    x += i
  two_c = add(x,one_add,n)
  return two_c
# print(twos_comp('00011',n))
```

SHIFT LEFT

In [19]:

```
def shl(bits):
    bit_list=[]
    bit_list[:0]=bits
    for i in range(0, len(bit_list)-1):
        bit_list[i] = bit_list[i+1]
    del bit_list[-1]
    bits = ''
    for i in bit_list:
        bits += i
    return bits
```

DISPLAY THE CALCULATION TABLE

In [20]:

```
def table(bits,count,oper,n): # Display Function
    A = bits[:n+1]
    q_bin = bits[n+1:]
    print(f'  {count}      {A}   {q_bin} {oper}')
```

NON RESTORING DIVISION ALGORITHM

In [21]:

```
def NonRestoringDiv(bits,count,n):
    A = bits[:n+1]
    q_bin = bits[n+1:]
    if(A[0] == '0'):
        bits = A + q_bin
        bits = shl(bits)
        table(bits,count,'    Shift Left',n)
```

```
        A = bits[:n+1]
        q_bin = bits[n+1:]
        A = add(A,minus_M,n)
        if(A[0] == '1'):
            q_bin = q_bin + '0'
            bits = A + q_bin
            table(bits,count,'   A = A - M',n)
        else:
            q_bin = q_bin + '1'
            bits = A + q_bin
            table(bits,count,'   A = A - M',n)
    else:
        bits = A + q_bin
        bits = shl(bits)
        table(bits,count,'    Shift Left',n)
        A = bits[:n+1]
        q_bin = bits[n+1:]
        A = add(A,plus_M,n)
        if(A[0] == '1'):
            q_bin = q_bin + '0'
            bits = A + q_bin
            table(bits,count,'   A = A + M',n)
        else:
            q_bin = q_bin + '1'
            bits = A + q_bin
            table(bits,count,'   A = A + M',n)
    count -= 1
    if(count > 0):
        NonRestoringDiv(bits,count,n)
    else:
        A = bits[:n+1]
        q_bin = bits[n+1:]
        if(A[0] == '1'):
            A = add(A,plus_M,n)
            bits = A + q_bin
        A = bits[:n+1]
        q_bin = bits[n+1:2*n+1]
        print(f'A: {A}, Q: {q_bin}')
        remainder = int(('0b' + A),2)
        quotient = int(('0b' + q_bin),2)
        print(f"Remainder: {remainder} , Quotient: {quotient}")
```

INPUT

In [22]:

```
# Check for m and q and accordingly get +M,-M,Q
q = 12 # Dividend
m = 3  # Divisor
count = n = 4  # Number of bits
A = '0'*(n+1)

# M - +ve Q - +ve
if(m>=0 and q>=0):
    plus_M = to_binary(m)
    minus_M = twos_comp(plus_M,n)
    # While converting q in binary
    x = list(to_binary(q))
    q_bin = ''
    if(x[0] == '0'):
        del x[0]
    for i in x:
        q_bin += i

# Trace the table
print("\n-----------NON RESTORING DIVISION-----------\n")
print('count  ','  A  ','   q  ','    Operation')

# Initialisation
bits = A + q_bin
# print(bits)
```

```
table(bits,count,'   Initialisation',n)
NonRestoringDiv(bits,count,n)
```

```
-----------NON RESTORING DIVISION-----------

count     A      q       Operation
  4     00000   1100    Initialisation
  4     00001   100     Shift Left
  4     11110   1000    A = A - M
  3     11101   000     Shift Left
  3     00000   0001    A = A + M
  2     00000   001     Shift Left
  2     11101   0010    A = A - M
  1     11010   010     Shift Left
  1     11101   0100    A = A + M
A: 00000, Q: 0100
Remainder: 0 , Quotient: 4
```