| | | | |
|---|---|---|---|
| **Course Name:** | **Computer Organization and Architecture Laboratory** | **Semester:** | **III** |
| **Date of Performance:** | **14/08/25** | **Batch No:** | **B1** |
| **Faculty Name:** | **Prof. Sheetal Pereira** | **Roll No:** | **16010124080** |
| **Faculty Sign & Date:** | | **Grade/Marks:** | **25** |

# Experiment No: 04

## Title: Non-Restoring Method of Division

**Aim and Objective of the Experiment:**

**Aim:** The basis of algorithm is based on paper and pencil approach and the operation involve repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

**Objective:** To study and implement Non Restoring method of division

**COs to be achieved:**

CO1: Describe and define the structure of a computer with buses structure and detail working of the ALU

**Books/Journals/Websites referred:**

**1.** Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
**2.** William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
**3.** Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

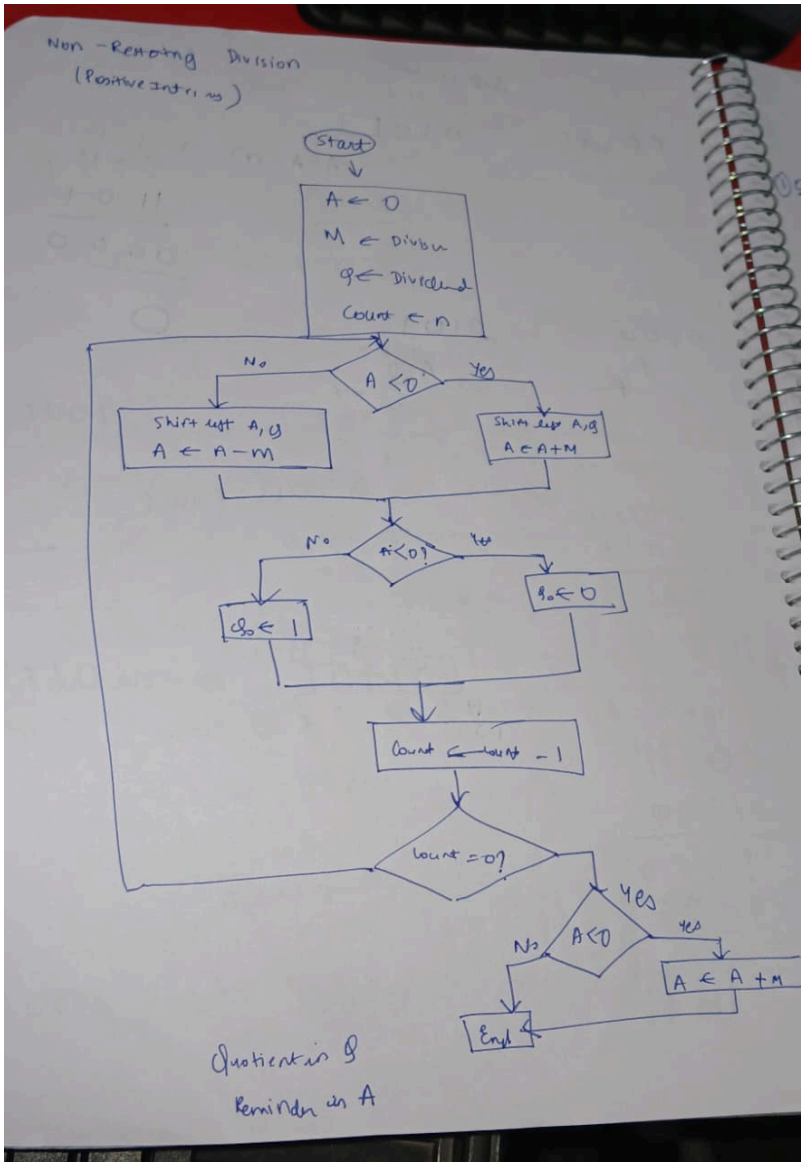
**Theory:** The **non-restoring division algorithm** is a faster alternative to the restoring method for binary division. Instead of "restoring" the accumulator when a subtraction results in a negative

value, it continues to the next step by adding the divisor back only when needed. This reduces the number of operations per iteration, improving performance.

The algorithm works by checking the sign of the accumulator after each operation. If it's positive, it subtracts the divisor in the next step; if negative, it adds the divisor. The quotient bit is determined accordingly. After all iterations, if the accumulator is negative, a final correction step is performed by adding the divisor once.

This method is more efficient in hardware implementations as it avoids unnecessary restoring operations.

**Flowchart for Restoring of Division ( Students need to draw):-**



**Design Steps**:

1. Start
2. Initialize:
   - A = 0 (Accumulator)
   - M = Divisor
   - Q = Dividend
   - count = n (number of bits)
3. Left shift (A, Q) combined by 1 bit
4. If MSB of previous A = 0 (A was positive or zero)

   Then A = A - M

   Else (A was negative)

   A = A + M
5. If MSB of A = 0

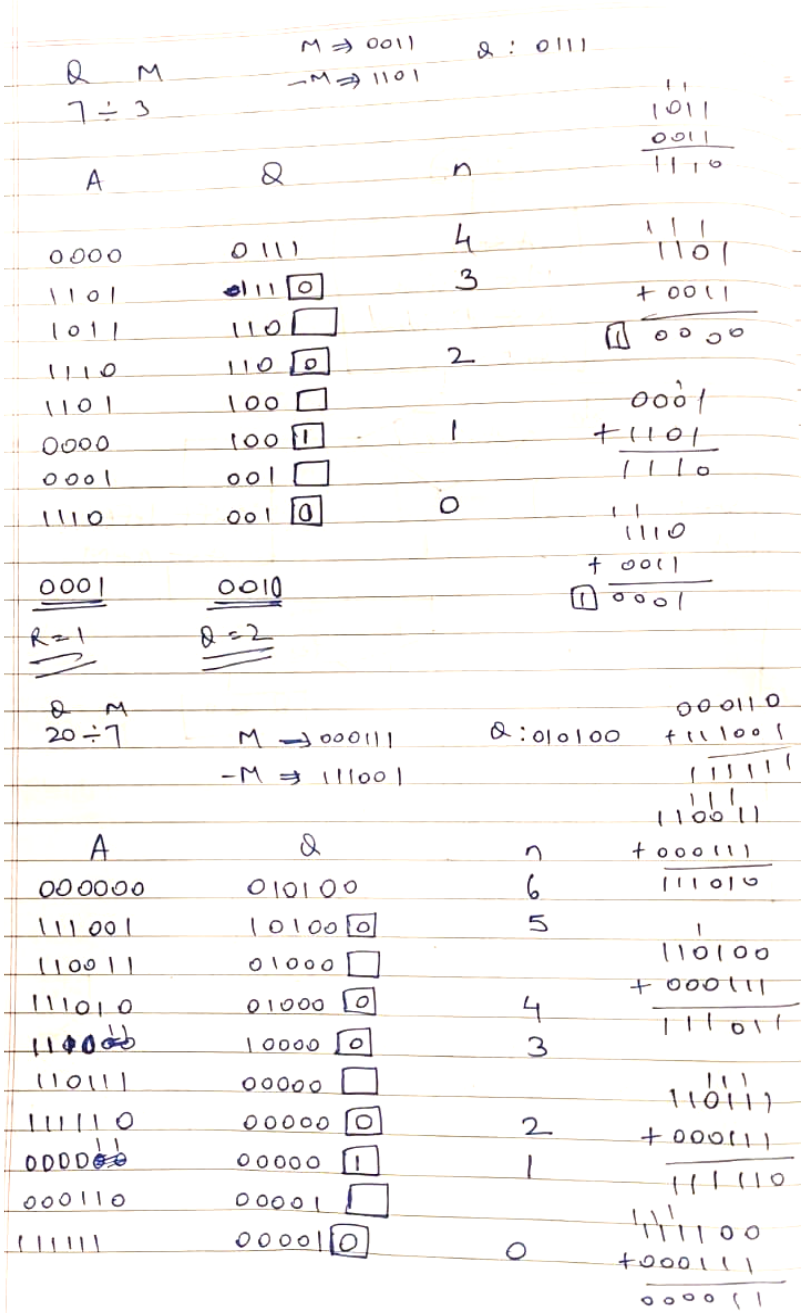
   Set Q0 = 1

   Else

   Set Q0 = 0
6. Decrement count by 1
7. If count ≠ 0, go to step 3
8. If A < 0 (negative remainder), then

   A = A + M (final correction step)
9. Stop

**Example: - (Handwritten solved problem needs to be uploaded):-**



**CODE:-**

```
def compute_bits(num1,num2):
    a = max(num1,num2)
    # Calculate minimum bits to represent max number plus one bit for sign
if needed
    # For simplicity just add some bits depending on range:
    if a <= 7:
        length = 4
    elif a <= 15:
        length = 5
    elif a <= 21:
        length = 6
    elif a <= 28:
        length = 7
    elif a <= 35:
        length = 8
    elif a <= 41:
        length = 9
    elif a <= 49:
        length = 10
```

```
    else:
        print("NOT ENOUGH TO COMPUTE YOUR RESULT")
        length = None
    return length



def to_binary(num: int, bit_length: int) -> list:
    binary_num = [0] * bit_length
    i = bit_length - 1

    while num > 0 and i >= 0:
        binary_num[i] = num % 2
        num = num // 2
        i -= 1

    return binary_num



def binary_addition(b1, b2):
    carry = 0
    final = [0] * len(b1)

    for i in range(len(b1) - 1, -1, -1):
        total = b1[i] + b2[i] + carry
```

```
        final[i] = total % 2
        carry = total // 2

    return final



def to_negative(binary):
    inverted = [1 - bit for bit in binary]
    negative = binary_addition(inverted, ([0]*(len(binary)-1)) + [1])
    if len(negative) > len(binary):
        negative = negative[1:]
    return negative

def to_set_initials():
    question = input("Enter division (format m/q): ")
    nums = question.split('/')

    Dividend, Divisor = int(nums[0]), int(nums[1])
    bits_required = compute_bits(Dividend, Divisor)
    if bits_required is None:
        exit()

    M = to_binary(Divisor, bits_required)
    M_neg = to_negative(M)
```

```
    Q = to_binary(Dividend, bits_required)
    A = [0] * bits_required
    n = bits_required

    # print(f"M: {M}   M_neg: {M_neg}   Q: {Q}   A: {A}")

    return (M, M_neg, Q, A, n)

def non_restoring(M,M_neg,Q,A,n):
    while (n!=0):
        if A[0]==0:
            A = A[1:] + [Q[0]]
            Q = Q[1:]
            A= binary_addition(A,M_neg)

        else:
            A = A[1:] + [Q[0]]
            Q = Q[1:]
            A= binary_addition(A,M)

        if A[0]==0:
            Q=Q+[1]
        else:
```

```
            Q=Q+[0]
        n=n-1
    if n==0:
        if A[0]==1:
            A=binary_addition(A,M)
        else:
            pass



    return (Q,A)




def to_decimal(binary_list):
    decimal = 0
    length = len(binary_list)
    for i, bit in enumerate(binary_list):
        decimal += bit * (2 ** (length - i - 1))
    return decimal



my_values = to_set_initials()
quotient,remainder = non_restoring(*my_values)
```

```
print(f"The answer is : Remainder {to_decimal(remainder)}\nQuotient
{to_decimal(quotient)}")
```

## Post Lab Subjective/Objective type Questions:

- **Why is the non-restoring division algorithm considered faster than the restoring method in binary division?**
  - Non-restoring division does not restore the remainder after a negative subtraction.
  - Instead of doing an extra addition to fix the remainder every time subtraction fails, it just keeps going with the next step.
  - This removes the restore (add-back) step present in restoring division, saving time.
  - Fewer operations per cycle → faster overall division.
- **What is the purpose of the final correction step in the non-restoring division algorithm?**
  - Because non-restoring division doesn't restore after every negative remainder, the final remainder might be negative at the end.
  - The final correction step adds the divisor back if the remainder is negative.
  - This ensures the final remainder is positive and correct.
  - It fixes any leftover error caused by skipping restores during the process.

- **Solve 10/3 using Non-Restoring algorithm for division operation?**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\KJSCE\Desktop\cc> & C:/Users/KJSCE/AppData/Local/Programs/Python/Python312/python.exe c:/Users/KJSCE/Desktop/cc/non_restoring.py
Enter division (format m/q): 10/3
The answer is : Remainder 1
Quotient 3
PS C:\Users\KJSCE\Desktop\cc>
```

## Conclusion:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\KJSCE\Desktop\cc> & C:/Users/KJSCE/AppData/Local/Programs/Python/Python312/python.exe c:/Users/KJSCE/Desktop/cc/non_restoring.py
Enter division (format m/q): 17/5
The answer is : Remainder 2
Quotient 3
PS C:\Users\KJSCE\Desktop\cc> & C:/Users/KJSCE/AppData/Local/Programs/Python/Python312/python.exe c:/Users/KJSCE/Desktop/cc/non_restoring.py
Enter division (format m/q): 14/6
The answer is : Remainder 2
Quotient 2
PS C:\Users\KJSCE\Desktop\cc> & C:/Users/KJSCE/AppData/Local/Programs/Python/Python312/python.exe c:/Users/KJSCE/Desktop/cc/non_restoring.py
Enter division (format m/q): 12/2
The answer is : Remainder 0
Quotient 6
PS C:\Users\KJSCE\Desktop\cc>
```

**Signature of faculty in-charge with Date:**