Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No. 4	

TITLE : To study and implement Non Restoring method of division

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.

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

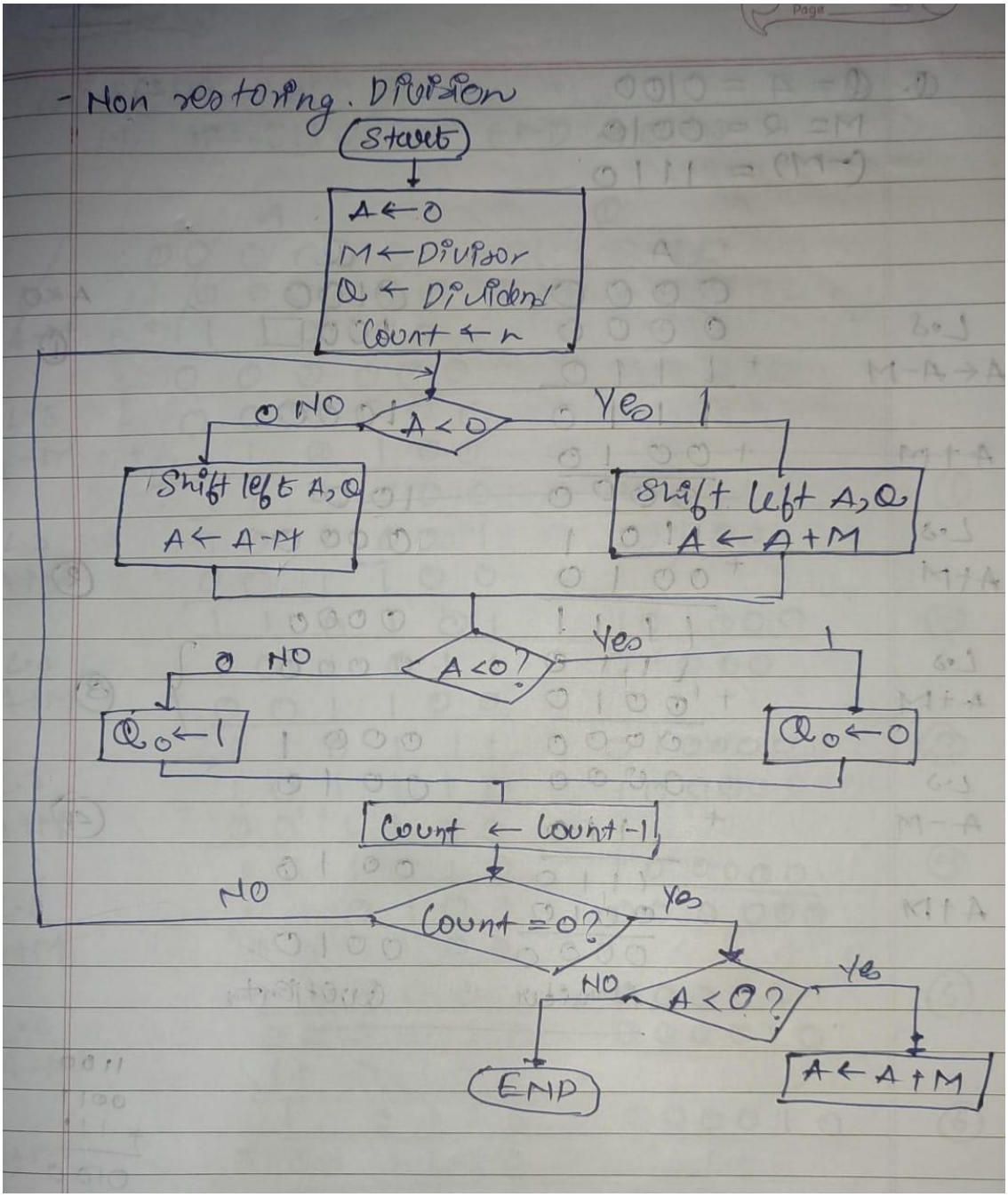
Books/ Journals/ Websites referred:

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.

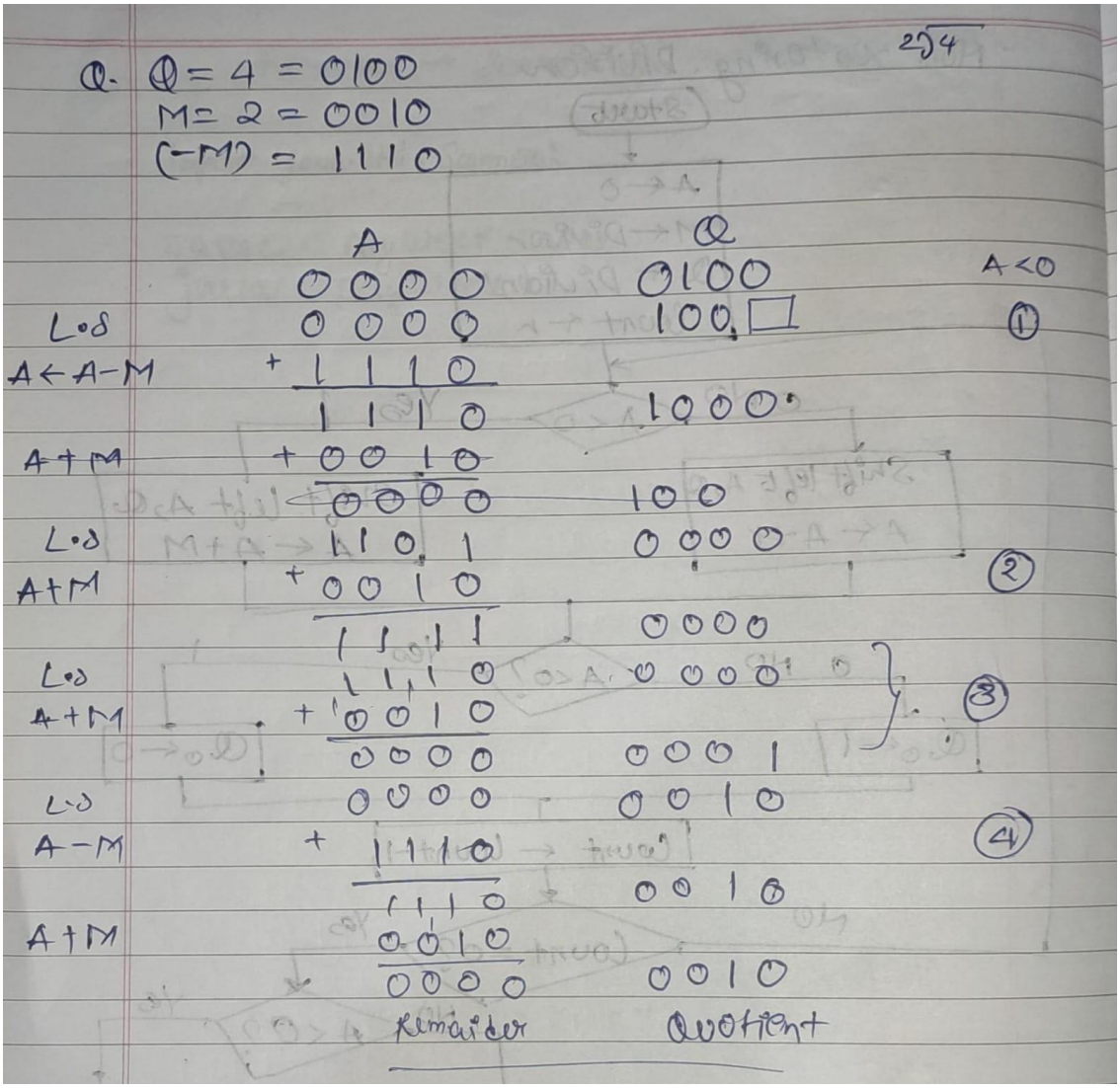
Pre Lab/ Prior Concepts:

The Non Restoring algorithm works with any combination of positive and negative numbers.

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



Example: (Handwritten solved problem needs to uploaded)



Code for Non Restoring Division:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

void integerToBinary(int number, int bitCount, char* binaryStr) {
  binaryStr[bitCount] = '\0';
  for (int i = bitCount - 1; i >= 0; i--) {
     binaryStr[i] = (number % 2) + '0';
     number /= 2;
```

```
    }
    if (number > 0) {
        strncpy(binaryStr, &binaryStr[strlen(binaryStr) - bitCount], bitCount);
    }
}

void computeComplement(char* binaryStr) {
    for (int i = 0; i < strlen(binaryStr); i++) {
        binaryStr[i] = binaryStr[i] == '0' ? '1' : '0';
    }
}

void binaryAddition(char* binary1, char* binary2, char* result) {
    int length = strlen(binary1);
    int carry = 0;
    result[length] = '\0';

    for (int i = length - 1; i >= 0; i--) {
        int bit1 = binary1[i] - '0';
        int bit2 = binary2[i] - '0';
        int sum = bit1 + bit2 + carry;
        carry = sum / 2;
        result[i] = (sum % 2) + '0';
    }
}

void leftShiftArithmetic(char* accumulator, char* quotient) {
    memmove(accumulator, &accumulator[1], strlen(accumulator) - 1);
    accumulator[strlen(accumulator) - 1] = quotient[0];
    memmove(quotient, &quotient[1], strlen(quotient) - 1);
    quotient[strlen(quotient) - 1] = '0';
}

int main() {
    int dividend, divisor;
    printf("Enter the dividend: ");
    scanf("%d", &dividend);
    printf("Enter the divisor: ");
    scanf("%d", &divisor);

    if (divisor == 0) {
        printf("Error: Division by zero is not allowed.\n");
        return 1;
    }
```

```
int bitCount = fmax(ceil(log2(abs(divisor) + 1)), ceil(log2(abs(dividend) + 1))) + 1;

char accumulator[bitCount + 1];
char binaryDivisor[bitCount + 1];
char binaryDividend[bitCount + 1];
char negDivisor[bitCount + 1];
char temp[bitCount + 1];
accumulator[0] = '0';
for (int i = 1; i < bitCount; i++) accumulator[i] = '0';
accumulator[bitCount] = '\0';

integerToBinary(divisor, bitCount, binaryDivisor);
integerToBinary(dividend, bitCount, binaryDividend);

strcpy(negDivisor, binaryDivisor);
computeComplement(negDivisor);
integerToBinary(1, bitCount, temp);
binaryAddition(negDivisor, temp, negDivisor);

for (int i = 0; i < bitCount; i++) {
   leftShiftArithmetic(accumulator, binaryDividend);

   if (accumulator[0] == '1') {
      binaryAddition(accumulator, binaryDivisor, temp);
      strcpy(accumulator, temp);
   } else {
      binaryAddition(accumulator, negDivisor, temp);
      strcpy(accumulator, temp);
   }

   if (accumulator[0] == '1') {
      binaryDividend[bitCount - 1] = '0';
   } else {
      binaryDividend[bitCount - 1] = '1';
   }
}

if (accumulator[0] == '1') {
   binaryAddition(accumulator, binaryDivisor, temp);
   strcpy(accumulator, temp);
}

printf("Decimal Quotient: %d\n", dividend / divisor);
printf("Decimal Remainder: %d\n", dividend % divisor);
printf("Binary Quotient: %s\n", binaryDividend);
```

```
  printf("Binary Remainder: %s\n", accumulator);

  return 0;
}
```

Output

```
Enter the dividend: 4
Enter the divisor: 2
Decimal Quotient: 2
Decimal Remainder: 0
Binary Quotient: 0010
Binary Remainder: 0000
```

Conclusion

In this experiment, we explored the non-restoring division algorithm and its application in performing division. We also confirmed the algorithm's functionality by implementing code that executes division using the non-restoring method.

Post Lab Descriptive Questions

What are the advantages of non-restoring division over restoring division?

Non-restoring division and restoring division are distinct algorithms utilized for integer division. Non-restoring division holds several advantages over restoring division:

1. Simplified Implementation: Non-restoring division is often considered more straightforward to implement in hardware as it requires fewer control signals and is less complex to design.
2. Reduced Iterations: Non-restoring division often requires fewer iterations than restoring division because it doesn't involve restoring the remainder when a negative result occurs. This efficiency can result in quicker execution, particularly when dealing with large operands.
3. Faster in Special Cases: In situations where the divisor is much smaller than the dividend, non-restoring division can complete faster because it avoids unnecessary restoration operations, unlike restoring division.
4. Lower Latency Due to its simpler control logic and fewer steps, non-restoring division typically has lower latency, meaning it can deliver faster results in both hardware and software implementations. This reduced latency contributes to faster results.
5. No test subtraction required: The sign bit determines whether an addition or subtraction is used, so a test subtraction is not required.

Date: 02 / 08 / 2024