## Heaven's Light is Our Guide Rajshahi Universiy of Engineering and Technology



#### Course Code ECE 2214

#### Course Title

Numerical Methods and Discrete Mathematics Sessional

Experiment Date: October 9, 2023 Submission Date: October 14, 2023

Lab Report 5: Modular Exponentiation and Binary Arithmetic in Python

Submitted to
Md. Nahiduzzaman
Lecturer
Dept of ECE, Ruet

Submitted by Md. Tajim An Noor Roll: 2010025

### Contents

L		roducti	
	1.1	Modul	ar Exponentiation
	1.2	Binary	Arithmetic
2	Too	ols Used	d
3	Pro		
	3.1	Code:	
		3.1.1	Modular Exponentiation
			Binary Arithmetic
			Integrated
	3.2		t
Ĺ	Dis	cussion	

# Modular Exponentiation and Binary Arithmetic in Python.

Md Tajim An Noor

#### 1 Introduction

#### 1.1 Modular Exponentiation

Modular arithmetic is the branch of arithmetic mathematics related with the "mod" functionality. Basically, modular arithmetic is related with computation of "mod" of expressions. Expressions may have digits and computational symbols of addition, subtraction, multiplication, division or any other. Here we will discuss briefly about all modular arithmetic operations.

The result of  $(a \wedge b \mod m)$  is the modular exponentiation. [1]

#### 1.2 Binary Arithmetic

Binary number system uses only two digits, 0 & 1. The basic arithmetic operations like addition and subtraction are known as binary arithmetic. Binary arithmetic starts from the least significant bit of a binary number and gradually goes towards the most significant bit.

#### 2 Tools Used

- Python
- VS Code for running python code
- MacTeX -LATEX compiler
- VS Code with LaTeXworkshop extension as a text editor

#### 3 Process

#### 3.1 Code:

#### 3.1.1 Modular Exponentiation

```
base = int(input("Base: "))
   expo = int(input("Exponent: "))
  mod = int(input("Mod: "))
   # function to find modular exponent
   def bin_exp(base, exponent, mod):
       ans = 1
       while exponent:
           if exponent % 2 == 1:
10
               ans = (ans * base) % mod
11
           base = (base * base) % mod
12
           exponent = int(exponent) / 2
13
       return ans
14
16
  print("Modular exponent: " + str(bin_exp(base, expo, mod)))
```

#### 3.1.2 Binary Arithmetic

```
a = input("Binary A: ")
b = input("Binary B: ")

# bionary addition function
def bin_add(a, b):
max_len = max(len(a), len(b))
a = a.zfill(max_len)
b = b.zfill(max_len)
result = ""
carry = 0
```

```
11
       for i in range(max_len - 1, -1, -1):
12
            r = carry
13
            r += 1 if a[i] == "1" else 0
14
            r += 1 \text{ if } b[i] == "1" \text{ else } 0
15
            result = ("1" if r % 2 == 1 else "0") + result
16
            # carry
17
            carry = 0 if r < 2 else 1
18
19
       if carry != 0:
20
            result = "1" + result
21
       return result.zfill(max_len)
22
24
   # subtraction function, useing 2's compliment
   def bin_sub(a, b):
26
       max_len = max(len(a), len(b))
27
       a = a.zfill(max_len)
28
       b = b.zfill(max_len)
29
       ch = ""
30
31
32
       c = compliment(b)
       result = bin_add(a, c)
33
34
       if len(result) > max_len:
35
            if result[0] == "1":
36
                 compliment(result)
37
                 for i in range(1, len(result)):
38
                     ch += result[i]
39
            return ch
       else:
41
            return result
43
   # finding 2's compliment of binary number
45
   def compliment(a):
       newa = ""
47
       for i in range(0, len(a)):
48
```

```
if a[i] == "1":
    newa += "0"
    else:
    newa += "1"
    return bin_add(newa, "1")

return of A & B = " + bin_add(a, b))
    print("Difference of A & B = " + bin_sub(a, b))
```

#### 3.1.3 Integrated

```
# importing the previously defined dunctions from other files
from amodularExpo import bin_exp
from bbinaryAddSub import bin_add, bin_sub

base = int(input("Base: "))
expo = int(input("Exponent: "))
mod = int(input("Mod: "))
a = input("Binary A: ")
b = input("Binary B: ")
# using the functions directly
print("Modular exponent: " + str(bin_exp(base, expo, mod)))
print("Sum of A & B = " + bin_add(a, b))
print("Difference of A & B = " + bin_sub(a, b))
```

#### 3.2 Output

Base: 2 Base: 2 Exponent: 5 Exponent: 5 Mod: 10 Mod: 100 Modular exponent: 2 Modular exponent: 32 Base: 7 Base: 5 Exponent: 3 Exponent: 2 Mod: 49 Mod: 12 Modular exponent: 7 Modular exponent: 1

Figure 1: Outputs for Modular Exponentiation

Binary A: 10011 Binary A: 111001 Binary B: 11001 Binary B: 1101 Sum of A & B = 101100Sum of A & B = 1000110Difference of A & B = 11010 Difference of A & B = 101100Binary A: 101 Binary A: 101010 Binary B: 110 Binary B: 110011 Sum of A & B = 1011101Sum of A & B = 1011Difference of A & B = 110111 Difference of A & B = 111

Figure 2: Outputs for Contraposition

```
Base: 2
Exponent: 5
Mod: 20
Binary A: 101
Binary B: 111
Modular exponent: 12
Sum of A & B = 1100
Difference of A & B = 110
```

```
Base: 5
Exponent: 3
Mod: 25
Binary A: 111
Binary B: 1011
Modular exponent: 5
Sum of A & B = 10010
Difference of A & B = 1100
```

```
Base: 10
Exponent: 5
Mod: 100
Binary A: 11111
Binary B: 10001
Modular exponent: 0
Sum of A & B = 110000
Difference of A & B = 01110
```

```
Base: 12
Exponent: 123
Mod: 1244
Binary A: 11101111011
Binary B: 111010101001
Modular exponent: 500
Sum of A & B = 1011000100100
Difference of A & B = 100011010010
```

Figure 3: Outputs for Logical Equivalence

#### 4 Discussion

In the first code, the modular exponentiation, 3 inputs are taken: base, exponent & mod. The base is the number that exponent will be done upon, exponent is the power for the base. Using an algorithm, getting way too large number can be avoided. The result is the base to the power of exponent modded by mod.

For the binary arithmetic operations; when adding two binary number, the iteration is done from LSB to MSB. If there is a carry, its calculated accordingly.

For the subtracting operation the 2nd number is converted to its 2's compliment equivalent. And then it was added using the adding function. Then after the addition was done, it was checked if there was any carry bit. If there were any, the result was converted to its equivalent 2's compliment and shown.

#### References

[1] "Modular Arithmetic," Apr. 2023, [Online; accessed 14. Oct. 2023]. [Online]. Available: https://www.geeksforgeeks.org/modular-arithmetic