The University of Newcastle
School of Information and Physical Sciences

SENG2250/6250 System and Network Security
Week 2 Lab

PART 1: Binary Number System and Logic

The binary number system is a base-2 numeral system used in computing and digital electronics. Unlike the decimal system (base-10), which uses 10 digits (0-9), the binary system uses only two digits: 0 and 1.

In binary, each digit position represents a power of 2. Starting from the rightmost position, the values increase as powers of 2 from right to left. For example:

- 1 in the rightmost position represents 2^0 = 1.
- 1 in the next position to the left represents 2^1 = 2.
- 1 in the third position from the right represents 2^2 = 4.

And so on.

Converting Binary to Decimal: To convert a binary number to its decimal equivalent, multiply each binary digit by its corresponding power of 2 and sum the results. For example:

Binary number: 1101

Decimal equivalent: (1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 8 + 4 + 0 + 1 = 13

Q1: Can you convert the following binary numbers (a) 1110 and (b) 110101.

Converting Decimal to Binary: To convert a decimal number to binary, repeatedly divide the decimal number by 2 and note the remainders from right to left. The binary representation will be the sequence of remainders. For example:

Decimal number: 53

Step 1: Divide 53 by 2

Quotient: 26

Remainder: 1

Step 2: Divide 26 (the quotient from Step 1) by 2

Quotient: 13

Remainder: 0

<u>Step 3</u>: Divide 13 (the quotient from Step 2) by 2

Quotient: 6

Remainder: 1

<u>Step 4</u>: Divide 6 (the quotient from Step 3) by 2

Quotient: 3

Remainder: 0

<u>Step 5</u>: Divide 3 (the quotient from Step 4) by 2

Quotient: 1

Remainder: 1

<u>Step 6</u>: Divide 1 (the quotient from Step 5) by 2

Quotient: 0

Remainder: 1

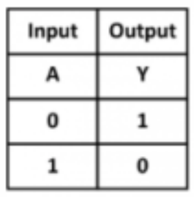
Now, read the remainders from the bottom to the top to get the binary equivalent of 53:

53 in binary is 110101.

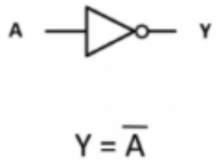
<u>Q2</u>: Compute the binary equivalent of 68? Verify your answer by applying the binary to decimal conversion technique.

<u>Binary Logic</u>: Binary logic is the foundation of digital computing and is based on two fundamental values: 0 and 1, representing false and true, respectively. In binary logic, various logical operations can be performed on binary inputs using logic gates.

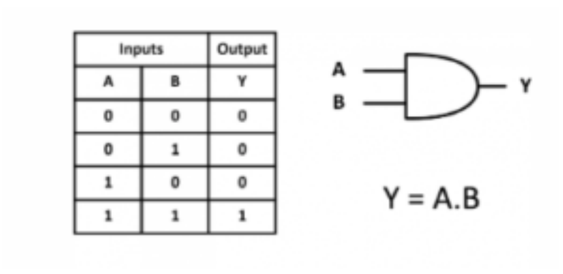
- NOT (! or ~): The NOT operation, also called inversion or complement, takes a single input and returns its opposite value. If the input is true (1), it returns false (0), and vice versa.



Input	Output
A	Y
0	1
1	0



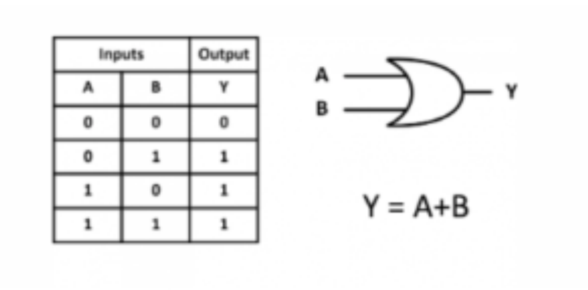
- AND (&&): The AND operation returns true (1) only if both input values are true (1); otherwise, it returns false (0).



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = A.B$

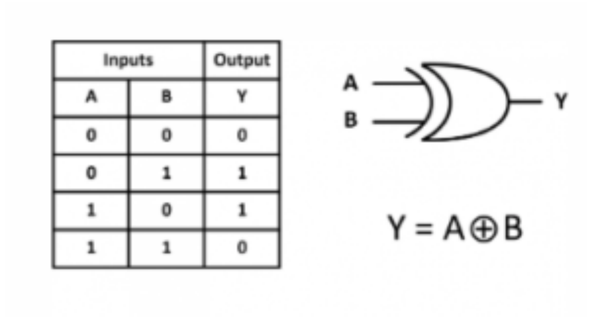
- OR (||): The OR operation returns true (1) if at least one of the input values is true (1); it returns false (0) only if both inputs are false (0).



Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

$Y = A+B$

- XOR (exclusive OR) (^): The XOR operation returns true (1) if the input values are different; otherwise, it returns false (0). In other words, it's true when there is an odd number of true inputs.



Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B$

Q3: Compute the bitwise NOT, AND, OR and XOR operations of the following binary streams: (a) 110101 and (b) 011011.

Complement of AND: NAND - Returns false (0) if both inputs are true (1), otherwise true (1).
Complement of OR: NOR - Returns true (1) only if both inputs are false (0), otherwise false (0).
Complement of XOR: XNOR - Returns true (1) if the inputs are the same (both true or both false), otherwise false (0).

Hexadecimal Number System: The hexadecimal number system is a base-16 numeral system commonly used in computing and programming. It uses sixteen digits: 0-9 and A-F, where A represents 10, B represents 11, and so on up to F representing 15.

Conversion between Binary and Hexadecimal: To convert binary to hexadecimal, group the binary digits in sets of four from right to left and then replace each set with its corresponding hexadecimal digit.

```
Example: Convert binary 11011011 to hexadecimal.
1101 1011 -> D B
Hexadecimal equivalent: DB
```

To convert hexadecimal to binary, replace each hexadecimal digit with its four-digit binary equivalent.

```
Example: Convert hexadecimal A7 to binary.
A -> 1010
7 -> 0111
Binary equivalent: 10100111
```

Using hexadecimal simplifies representing and working with large binary numbers, as each hexadecimal digit corresponds to four binary digits (bits). It is widely used in computer memory addressing, colour representation, and networking.

Q4: Convert the following binary number to hexadecimal: 1101010010101101 and the following hexadecimal number to binary: 3F8E.

Part 2: Modular Arithmetic

Modular arithmetic, also known as clock arithmetic, is a branch of mathematics that deals with numbers' remainders when divided by a fixed positive integer called the modulus.

In modular arithmetic, we are only concerned with the remainder after dividing a number by the modulus, ignoring the quotient. The set of integers modulo the modulus forms a finite set of possible remainders, ranging from 0 to (modulus - 1).

```
Example of Modular Arithmetic with Modulus 7:
Addition: (6 + 5) mod 7 = 11 mod 7 = 4
Subtraction: (8 - 3) mod 7 = 5 mod 7 = 5
Multiplication: (2 * 4) mod 7 = 8 mod 7 = 1
Division: (12 / 3) mod 7 = 4 mod 7 = 4
```

Modular arithmetic follows several important rules that govern the operations within this arithmetic system. Here are the key rules for modular arithmetic with modulus "m":

Addition	(a + b) mod m = (a mod m + b mod m) mod m
Subtraction	(a - b) mod m = (a mod m - b mod m) mod m
Multiplication	(a * b) mod m = (a mod m * b mod m) mod m
Division	(a / b) mod m ≡ (a mod m * b^(-1) mod m) mod m *(b^(-1) mod m is the modular inverse of b, such that (b * b^(-1)) mod m ≡ 1)
Exponentiation	(a^k) mod m = [(a mod m)^k] mod m
Congruence	a ≡ b (mod m) if and only if (a - b) mod m = 0

Modular arithmetic is widely used in various fields, including computer science, cryptography, and number theory. It plays a crucial role in algorithms that require large number manipulations, such as in RSA encryption and hashing functions.

Q1: Solve the followings (preferably without using a calculator): (a) (651+7213) mod 47 and (b) (651×7213) mod 47.

Fast Exponentiation: The fast exponentiation with mod algorithm efficiently calculates the power of a number to a given exponent modulo a given modulus. This algorithm is particularly useful when dealing with large numbers, such as in cryptography or number theory, where the result needs to be reduced to the modulus to prevent overflow and obtain a smaller and manageable result.

For example, 5^{15} mod 17 can be efficiently computed as follows.

$$\begin{aligned}
5^{15} \bmod 17 &= 5\times5^{14} \bmod 17 \\
&= 5\times25^{7} \bmod 17 \\
&= 5\times8^{7} \bmod 17 && \text{[cause 25 mod 17=8]} \\
&= 5\times8\times8^{6} \bmod 17 \\
&= 6\times8^{6} \bmod 17 && \text{[cause 5x8 mod 17=6]} \\
&= 6\times64^{3} \bmod 17 \\
&= 6\times13^{3} \bmod 17 && \text{[cause 64 mod 17=13]} \\
&= 6\times13\times13^{2} \bmod 17 \\
&= 10\times13^{2} \bmod 17 && \text{[cause 6x13 mod 17=10]} \\
&= 10\times16 \bmod 17 \\
&= \mathbf{7}
\end{aligned}$$

Q2: Compute 19^{11} mod 26 using fast exponentiation with mod algorithm.

Homework: Can you write a (C/C++/Java/Python) program for the modular exponentiation operation based on the following pseudocode?

```
function powmod(base b, exponent e, modulus n) {
    if n  = 1
        return 0
    t = 1
    rs = 1
    while (t <= e) {
        rs = (rs * b) mod n
        t = t + 1
    }
    return rs
}
```

Using the above implementation to find the solutions to the followings:

$3^3 \; mod \; 7 = ?$
$10^8 \; mod \; 133 = ?$
$3785^{8395} \; mod \; 65537 = ?$

Multiplicative Inverse: The multiplicative inverse modulo "m" of a number "a" is another number "x" such that their product, when taken modulo "m," yields a result of 1. In other words, "x" is the number that, when multiplied by "a" and then reduced modulo "m," gives a remainder of 1.

For example, let's find the multiplicative inverse of 5 modulo 11:

We need to find "x" such that (5 * x) mod 11 ≡ 1.

Possible values of "x" are:

x = 2: (5 * 2) mod 11 = 10 mod 11 ≡ 10 (not 1)

x = 3: (5 * 3) mod 11 = 15 mod 11 ≡ 4 (not 1)

x = 4: (5 * 4) mod 11 = 20 mod 11 ≡ 9 (not 1)

x = 5: (5 * 5) mod 11 = 25 mod 11 ≡ 3 (not 1)

x = 6: (5 * 6) mod 11 = 30 mod 11 ≡ 8 (not 1)

x = 7: (5 * 7) mod 11 = 35 mod 11 ≡ 2 (not 1)

x = 8: (5 * 8) mod 11 = 40 mod 11 ≡ 7 (not 1)

x = 9: (5 * 9) mod 11 = 45 mod 11 ≡ 1 (YES! We found the multiplicative inverse.)

So, the multiplicative inverse of 5 modulo 11 is 9, which is denoted as 5^(-1) ≡ 9 (mod 11).

Q3: What is the multiplicative inverse of 19 mod 31?

Is there any systematic way of computing the multiplicative inverse of a mod m? (How about we do some homework for it?)

Part 3: Brute Force Attacks

Brute Force Attacks tries to attempt all possible passwords until the correct one is found. It is also known as an exhaustive key search. Assume that an adversary can (randomly) try 100,000 different passwords per second.

a. If a password consists of 8 digits, what is the expected (average) time to find the correct password?
b. What is the expected time to find the correct password if a password consists of 8 characters, including numbers and/or lower-case English letters?
c. What is the expected time to find the correct password if a password consists of 6 characters, including numbers and/or lower-case English letters?
d. What are the unicity distances for (a), (b) and (c)?