Lab Experiment # 1

Name of the Experiment: Implementing encoding and decoding scheme using NRZ-L, NRZ-I and Manchester

NRZ-L

```
import matplotlib.pyplot as plt
import numpy as np

# NRZ-L Encoding function
def nrzl_encode(binary_sequence):
    encoded_signal = [1 if bit == '1' else 0 for bit in binary_sequence]
    return encoded_signal

# NRZ-L Decoding function
def nrzl_decode(encoded_signal):
    decoded_sequence = ''.join(['1' if level == 1 else '0' for level in
encoded_signal])
    return decoded_sequence
```

NRZ-I

```
import matplotlib.pyplot as plt
import numpy as np

# NRZ-I Encoding function
def nrzi_encode(binary_sequence):
    encoded_signal = []
    current_level = 1  # Starting level can be 0 or 1
    for bit in binary_sequence:
        if bit == '1':
            current_level = 1 - current_level  # toggle level
        encoded_signal.append(current_level)
    return encoded_signal

# Fixed NRZ-I Decoding function
def nrzi_decode(encoded_signal):
    decoded_sequence = ''
```

```python
    # Assume first bit is '0' if no transition, or '1' if there's a 
transition from starting level
    # You can also capture the first bit manually if needed
    decoded_sequence += '0'  # We'll update this based on actual encoding
    previous = encoded_signal[0]
    for current in encoded_signal[1:]:
        if current == previous:
            decoded_sequence += '0'
        else:
            decoded_sequence += '1'
        previous = current

    # Infer actual first bit by comparing the first two encoded levels
    # If there's a transition at the start, the first bit was '1'
    if len(encoded_signal) > 1 and encoded_signal[1] != encoded_signal[0]:
        decoded_sequence = '1' + decoded_sequence[1:]

    return decoded_sequence
```

Manchester:

```python
# Manchester Encoding function
def manchester_encode(binary_sequence):
    encoded_signal = []
    for bit in binary_sequence:
        if bit == '1':
            encoded_signal.extend([1, 0])  # high to low for '1'
        elif bit == '0':
            encoded_signal.extend([0, 1])  # low to high for '0'
    return encoded_signal

# Manchester Decoding function
def manchester_decode(encoded_signal):
    decoded_sequence = []
    # For Manchester decoding, check pairs of bits
    for i in range(0, len(encoded_signal), 2):
        if encoded_signal[i] == 1 and encoded_signal[i+1] == 0:
            decoded_sequence.append('1')  # high to low
        elif encoded_signal[i] == 0 and encoded_signal[i+1] == 1:
            decoded_sequence.append('0')  # low to high
    return ''.join(decoded_sequence)
```

Lab Experiment # 2

Name of the Experiment:
Implementing encoding and decoding scheme using Alternate Mark Inversion (AMI), Pseudoternary and Multi-Level Line

AMI:

```
import matplotlib.pyplot as plt
import numpy as np

def ami_encode(binary_str):
    l = -1
    encoded_signal = []

    for bit in binary_str:
        if bit == '0':
            encoded_signal.append(0)
        elif bit == '1':
            l *= -1
            encoded_signal.append(l)

    return encoded_signal

def ami_decode(encoded_signal):
    decoded_str = ''
    for value in encoded_signal:
        if value == 0:
            decoded_str += '0'
        else:
            decoded_str += '1'

    return decoded_str
```

```python
binary_str = input("Input: ")
if not all(bit in '01' for bit in binary_str):
   print("Invalid")
else:
    encoded_ami = ami_encode(binary_str)
    print(f"{encoded_ami}")

    decoded_ami = ami_decode(encoded_ami)
    print(f"AMI Decoded String: {decoded_ami}")
```

Pseudoternary

```python
def pseudoternary_encode(binary_str):
    l = 1
    encoded_signal = []

    for bit in binary_str:
        if bit == '1':
            encoded_signal.append(0)
        elif bit == '0':
            l *= -1
            encoded_signal.append(l)

    return encoded_signal

def pseudoternary_decode(encoded_signal):
    decoded_str = ''
    for value in encoded_signal:
        if value == 0:
            decoded_str += '1'
        else:
            decoded_str += '0'

    return decoded_str

binary_str = input("Input: ")
encoded_pseudoternary = pseudoternary_encode(binary_str)
```

```python
print(f"Pseudoternary Encoded Signal: {encoded_pseudoternary}")

decoded_pseudoternary = pseudoternary_decode(encoded_pseudoternary)
print(f"Pseudoternary Decoded String: {decoded_pseudoternary}")
```

MLT-3:

```python
def mlt3_encoding(data):
    encoded = []
    levels = [0, 1, 0, -1]
    current_index = 0
    current_level = 0
    for bit in data:
        if bit == '0':
            encoded.append(current_level)
        else:
            current_index = (current_index + 1) % 4
            current_level = levels[current_index]
            encoded.append(current_level)
    return encoded

def mlt3_decoding(encoded_signal):
    decoded = ''
    prev = 0
    for level in encoded_signal:
        if level == prev:
            decoded += '0'
        else:
            decoded += '1'
        prev = level
    return decoded
```

LAB EXP-3: Title: Implementing Bit Stuffing and De-stuffing Using Socket Programming

Client:

```
public class ClientTwoWay {

    public static String stuff(String message) {
        String FLAG = "@";
        String ESC = "\\";

        message = message.replace("\\", "\\\\");
        message = message.replace("@", "\\@");
        return FLAG + message + FLAG;
    }

    public static String destuff(String message) {
        String FLAG = "@";
        String ESC = "\\";

        if (message.startsWith(FLAG) && message.endsWith(FLAG)) {
            message = message.substring(1, message.length() - 1);
        }

        message = message.replace("\\@", "@");
        message = message.replace("\\\\", "\\");
        return message;
    }

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("192.168.112.78", 6000);
            System.out.println("Connected to the server.");

            DataInputStream input = new
DataInputStream(socket.getInputStream());
            DataOutputStream output = new
DataOutputStream(socket.getOutputStream());

            Thread receiveThread = new Thread(() -> {
                try {
                    String message;
                    while (!(message =
input.readUTF()).equalsIgnoreCase("stop")) {
                        System.out.println("Server: " + destuff(message));
                    }
                    System.out.println("Server ended the chat.");
                    System.exit(0);
                } catch (IOException e) {
                    System.out.println("Connection closed.");
                }
            });

            Thread sendThread = new Thread(() -> {
                try (BufferedReader console = new BufferedReader(new
InputStreamReader(System.in))) {
                    String message;
                    while (!(message =
console.readLine()).equalsIgnoreCase("stop")) {
                        output.writeUTF(stuff(message));
                        output.flush();
                    }
                    System.out.println("Client ended the chat.");
                    System.exit(0);
                } catch (IOException e) {
                    System.out.println("Error sending message.");
                }
            });

            receiveThread.start();
            sendThread.start();

        } catch (IOException e) {
            System.out.println("Unable to connect to the server: " +
e.getMessage());
        }
    }
}
```

Server:

```
public class ServerTwoWay {

    public static String stuff(String message) {
        String FLAG = "@";
        String ESC = "\\";

        message = message.replace("\\", "\\\\");
        message = message.replace("@", "\\@");
        return FLAG + message + FLAG;
    }

    public static String destuff(String message) {
        String FLAG = "@";
        String ESC = "\\";

        if (message.startsWith(FLAG) && message.endsWith(FLAG)) {
            message = message.substring(1, message.length() - 1);
        }

        message = message.replace("\\@", "@");
        message = message.replace("\\\\", "\\");
        return message;
    }

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(6000);
        System.out.println("Server started. Waiting for client...");

        Socket socket = serverSocket.accept();
        System.out.println("Client connected: " +
socket.getInetAddress());

        DataInputStream input = new
DataInputStream(socket.getInputStream());
        DataOutputStream output = new
DataOutputStream(socket.getOutputStream());

        Thread receiveThread = new Thread(() -> {
            try {
                String message;
                while (!(message =
input.readUTF()).equalsIgnoreCase("stop")) {
                    System.out.println("Client: " + destuff(message));
                }
                System.out.println("Client ended the chat.");
                System.exit(0);
            } catch (IOException e) {
                System.out.println("Connection closed.");
            }
        });

        Thread sendThread = new Thread(() -> {
            try (BufferedReader console = new BufferedReader(new
InputStreamReader(System.in))) {
                String message;
                while (!(message =
console.readLine()).equalsIgnoreCase("stop")) {
                    output.writeUTF(stuff(message));
                    output.flush();
                }
                System.out.println("Server ended the chat.");
                System.exit(0);
            } catch (IOException e) {
                System.out.println("Error sending message.");
            }
        });

        receiveThread.start();
        sendThread.start();
    }
}
```

EXP-4: Implementation of Multiplexing and Demultiplexing in Data Communication

Client:

```
public class R_53_27_Client {
    public static void main(String[] args) {
        try {

            Scanner sc = new Scanner(System.in);
            System.out.print("Server ID: ");
            String serverid = sc.nextLine();

            System.out.print("Server port: ");
            int port = sc.nextInt();

            System.out.print("Time slot: ");
            int T = sc.nextInt();
            sc.close();


            Socket socket = new Socket(serverid, port);
            DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());


            String[] savepath = {

"C:\\Users\\Anik\\Documents\\NetBeansProjects\\mux_demux\\src\\mux_demux\\
input1.txt",

"C:\\Users\\Anik\\Documents\\NetBeansProjects\\mux_demux\\src\\mux_demux\\
input2.txt",

"C:\\Users\\Anik\\Documents\\NetBeansProjects\\mux_demux\\src\\mux_demux\\
input3.txt"
            };

            int filenumber = savepath.length;


            dos.writeInt(filenumber);


            FileInputStream[] inputFiles = new
FileInputStream[filenumber];
```

```
            boolean[] finished = new boolean[filenumber];
            int fCount = 0;

            for (int i = 0; i < filenumber; i++) {
                inputFiles[i] = new FileInputStream(savepath[i]);
                finished[i] = false;
            }

            int w=0;
            while (fCount < filenumber) {
    String s = "";

    for (int i = 0; i < filenumber; i++) {
        int count = 0;

        while (count < T) {
            if (!finished[i]) {
                int data = inputFiles[i].read();
                if (data == -1) {
                    dos.writeByte('#');
                    s = s + '#';
                    finished[i] = true;
                    fCount++;
                } else {
                    dos.writeByte(data);
                    s = s + (char) data;
                }
                count++;
            } else {
                dos.writeByte('#');
                s = s + '#';
                count++;
            }
        }
    }

    System.out.println(w+" th  cycle: " + s);
    w++;
}



            for (FileInputStream fis : inputFiles) {
                fis.close();
            }
            dos.close();
            socket.close();
            System.out.println("Data sent successfully.");

        } catch (Exception e) {
            System.out.println("Connection error: " + e.getMessage());
        }
    }
}
```

Server:

```
public class R_53_27_Server {
public static void main(String[] args) throws IOException {
int port = 5000;
ServerSocket serverSocket = new ServerSocket(port);
System.out.println("Server is connected at port no: " + port);
System.out.println("Waiting for the client...");

Socket socket = serverSocket.accept();
System.out.println("Client request is accepted at port no: " +
socket.getPort());

DataInputStream dis = new DataInputStream(socket.getInputStream());

int numFiles = dis.readInt();
int T = 2;

System.out.println("Number of files: " + numFiles);


FileOutputStream[] outputFiles = new FileOutputStream[numFiles];
StringBuilder[] actualData = new StringBuilder[numFiles];
StringBuilder[] rawRounds = new StringBuilder[numFiles];

for (int i = 0; i < numFiles; i++) {
outputFiles[i] = new FileOutputStream("output" + (i + 1) + ".txt");
actualData[i] = new StringBuilder();
rawRounds[i] = new StringBuilder();
}

try {
int Count = 1;

while (true) {
StringBuilder round = new StringBuilder();
for (int i = 0; i < numFiles; i++) {
for (int t = 0; t < T; t++) {
byte b = dis.readByte();
char c = (char) b;

round.append(c);
rawRounds[i].append(c);

if (c != '#') {
actualData[i].append(c);
outputFiles[i].write(b);
}
}
}

System.out.println("Round " + Count + " received: " + round.toString());
Count++;
}

} catch (EOFException e) {
System.out.println("\nTransmission complete.");
/*
System.out.println("\nRaw data:");
for (int i = 0; i < numFiles; i++) {
System.out.println("File " + (i + 1) + ": " + rawRounds[i].toString());
}


for (int i = 0; i < numFiles; i++) {
System.out.println("output" + (i + 1) + ".txt: " +
actualData[i].toString());
}
*/
}

for (FileOutputStream fos : outputFiles) fos.close();
dis.close();
socket.close();
serverSocket.close();
}
}
```

Lab Experiment-05
Title: Implementation of CRC (Cyclic Redundancy Check) for Error Detection

Client Code:
```
import java.io.*;
import java.net.*;

public class Roll_09_27_CRCClient {
    public static void main(String[] args) throws Exception {
        String serverIP = "192.168.168.45";
        int port = 5000;
        Socket socket = new Socket(serverIP, port);
        System.out.println("Client connected to the server on Handshaking
port " + port);
        System.out.println("Client’s Communication Port: " +
socket.getLocalPort());
        System.out.println("Client is Connected");

        DataOutputStream out = new
DataOutputStream(socket.getOutputStream());

        BufferedReader reader = new BufferedReader(new
FileReader("input.txt"));
        String text = reader.readLine().trim();
        reader.close();
        System.out.println("File Content: " + text);

        StringBuilder binaryData = new StringBuilder();
        for (char c : text.toCharArray()) {
            binaryData.append(String.format("%8s",
Integer.toBinaryString(c)).replace(' ', '0'));
        }
        System.out.println("Converted Binary Data: " + binaryData);

        String generator = "1101";
        int k = generator.length();
        String dataToDivide = binaryData + "0".repeat(k - 1);
        System.out.println("After Appending zeros Data to Divide: " +
dataToDivide);

        String remainder = getRemainder(dataToDivide, generator);
        System.out.println("CRC Remainder: " + remainder);

        String codeword = binaryData + remainder;
        System.out.println("Transmitted Codeword to Server: " + codeword);
        out.writeUTF(codeword);
        socket.close();
    }

    static String getRemainder(String data, String divisor) {
        int len = divisor.length();
        String remainder = data.substring(0, len);

        for (int i = len; i < data.length(); i++) {
            remainder = xor(remainder, divisor) + data.charAt(i);
        }
        remainder = xor(remainder, divisor);
        return remainder;
    }

    static String xor(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 1; i < a.length(); i++) {
            result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
        }
        return result.toString();
    }
}
```

Server Code:

```
import java.io.*;
import java.net.*;

public class Roll_09_27_CRCServer {
    public static void main(String[] args) throws Exception {

        int port = 5000;
        ServerSocket serverSocket = new ServerSocket(port);
        System.out.println("Server is connected at port no: " + port);
        System.out.println("Server is connecting");
        System.out.println("Waiting for the client");


        Socket socket = serverSocket.accept();
        System.out.println("Client request is accepted at port no: " +
socket.getPort());
        System.out.println("Server’s Communication Port: " +
socket.getLocalPort());


        DataInputStream in = new DataInputStream(socket.getInputStream());
        String receivedCodeword = in.readUTF();

        System.out.println("Received Codeword: " + receivedCodeword);

        String generator = "1101";


        double randkey=Math.random();
        if(randkey<=0.5){
            int ids= receivedCodeword.length()-2 ;
            StringBuilder sb = new StringBuilder(receivedCodeword);

            if (sb.charAt(ids) == '0')
            {
                sb.setCharAt(ids, '1');
            }
            else
            {
                sb.setCharAt(ids, '0');
            }
            receivedCodeword = sb.toString();

        }

        String remainder = getRemainder(receivedCodeword, generator);
        System.out.println("Calculated Remainder: " + remainder);


        if (remainder.chars().allMatch(ch -> ch == '0')) {
            System.out.println("No error detected in transmission.");
        } else {
            System.out.println("Error detected in transmission!");
        }

        socket.close();
        serverSocket.close();
    }


    static String getRemainder(String data, String divisor) {
        int len = divisor.length();
        String remainder = data.substring(0, len);

        for (int i = len; i < data.length(); i++) {
            remainder = xor(remainder, divisor) + data.charAt(i);
        }
        remainder = xor(remainder, divisor);
        return remainder;
    }


    static String xor(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 1; i < a.length(); i++) {
            result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
        }
        return result.toString();
    }



    }
```

Lab Experiment-07
Title: Implementation of Simple Spreading Techniques in Data and Telecommunications

Spreader Code:

```
import java.io.*;
import java.net.*;

public class Roll_09_27_Spreader {
    public static void main(String[] args) throws Exception {
        String serverIP = "10.33.2.206";
        int port = 5000;
        Socket socket = new Socket(serverIP, port);
        DataOutputStream out = new
DataOutputStream(socket.getOutputStream());

        String chip = "10101";

        String spreaded1 = spreadFile("input1.txt", chip);
        String spreaded2 = spreadFile("input2.txt", chip);

        out.writeUTF("FILE1");
        out.writeUTF(spreaded1);

        out.writeUTF("FILE2");
        out.writeUTF(spreaded2);

        socket.close();
    }

    static String spreadFile(String filename, String chip) throws
IOException {
        BufferedReader reader = new BufferedReader(new
FileReader(filename));
        StringBuilder textBuilder = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            textBuilder.append(line);
        }
        reader.close();
        String text = textBuilder.toString();
        StringBuilder binaryData = new StringBuilder();
        for (char c : text.toCharArray()) {
            binaryData.append(String.format("%8s",
Integer.toBinaryString(c)).replace(' ', '0'));
        }

        StringBuilder spreaded = new StringBuilder();
        for (int i = 0; i < binaryData.length(); i++) {
            char bit = binaryData.charAt(i);
            String bitStr = "";
            for (int j = 0; j < chip.length(); j++) {
                bitStr += bit;
            }
            spreaded.append(xor(bitStr, chip));
        }
        return spreaded.toString();
    }

    static String xor(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < a.length(); i++) {
            result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
        }
        return result.toString();
    }
}
```

Despreader Code:

```
import java.io.*;
import java.net.*;

public class Roll_09_27_Despreader {
    public static void main(String[] args) throws Exception {
        int port = 5000;
        ServerSocket serverSocket = new ServerSocket(port);
        Socket socket = serverSocket.accept();

        DataInputStream in = new DataInputStream(socket.getInputStream());
        String received = in.readUTF();

        String chip = "101";
        int chipLength = chip.length();
        StringBuilder recoveredBits = new StringBuilder();
        for (int i = 0; i < received.length(); i += chipLength) {
            String chunk = received.substring(i, i + chipLength);
            String result = xor(chunk, chip);
            int ones = 0;
            for (int j = 0; j < result.length(); j++) {
                if (result.charAt(j) == '1') ones++;
            }
            if (ones > chipLength / 2) {
                recoveredBits.append('1');
            } else {
                recoveredBits.append('0');
            }
        }

        StringBuilder output = new StringBuilder();
        for (int i = 0; i < recoveredBits.length(); i += 8) {
            String byteStr = recoveredBits.substring(i, i + 8);
            int ascii = Integer.parseInt(byteStr, 2);
            output.append((char) ascii);
        }

        FileWriter fw = new FileWriter("recovered_input1.txt");
        fw.write(output.toString());
        fw.close();

        socket.close();
        serverSocket.close();
    }

    static String xor(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < a.length(); i++) {
            result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
        }
        return result.toString();
    }
}
```