Batch: SY-IT(B3) **Experiment Number:6**

Roll Number: 16010423076 **Name:Ritesh Jha**

Aim of the Experiment: To interpret the concept of redundancy in data message for error detection and Correction using Hamming Code.

Program/ Steps:

```
def calculate_parity_bits(data):
    if len(data) != 7 or not all(bit in '01' for bit in data):
        raise ValueError("Data must be a 7-bit binary string")

    # Calculate parity bits
    p1 = int(data[0]) ^ int(data[1]) ^ int(data[3]) ^ int(data[4]) ^ int(data[6])
    p2 = int(data[0]) ^ int(data[2]) ^ int(data[3]) ^ int(data[5]) ^ int(data[6])
    p4 = int(data[1]) ^ int(data[2]) ^ int(data[3])
    p8 = int(data[4]) ^ int(data[5]) ^ int(data[6])

    # Construct the Hamming codeword
     return str(p1) + str(p2) + data[0] + str(p4) + data[1] + data[2] + data[3] + str(p8) + data[4] + data[5] + data[6]

def Transmitter():
    data = input('Enter 7-bit data code: ')

    if len(data) != 7 or not all(bit in '01' for bit in data):
        print("Invalid data input")
```

```
        return


    try:
        hamming_code = calculate_parity_bits(data)
        print(f"Generated Hamming Code: {hamming_code}")
    except ValueError as e:
        print(e)


def Receiver():
    codeword = input('Enter the Hamming code: ')
    rev_codeword = codeword[::-1]


    if len(rev_codeword) != 11:
        print("Invalid code")
        return
    if not all(bit in '01' for bit in rev_codeword):
        print("Invalid bit value")
        return


        p1_check = str(int(rev_codeword[2]) ^ int(rev_codeword[4]) ^ int(rev_codeword[6]) ^
int(rev_codeword[8]) ^ int(rev_codeword[10]) ^ int(rev_codeword[0]))

        p2_check = str(int(rev_codeword[2]) ^ int(rev_codeword[5]) ^ int(rev_codeword[6]) ^
int(rev_codeword[9]) ^ int(rev_codeword[10]) ^ int(rev_codeword[1]))

        p4_check = str(int(rev_codeword[4]) ^ int(rev_codeword[5]) ^ int(rev_codeword[6]) ^
int(rev_codeword[3]))

        p8_check = str(int(rev_codeword[8]) ^ int(rev_codeword[9]) ^ int(rev_codeword[10]) ^
int(rev_codeword[7]))
```

```
    print(f"p1: {p1_check}\np2: {p2_check}\np4: {p4_check}\np8: {p8_check}")

    if p1_check + p2_check + p4_check + p8_check == "0000":
        data = rev_codeword[2] + rev_codeword[4:7] + rev_codeword[8:]
        print(f"Correct code: {data}")
    else:
        error_pos_bin = p8_check + p4_check + p2_check + p1_check
        error_pos = int(error_pos_bin, 2)
        print(f"Error in bit: {error_pos}")
        error_pos -= 1
        if rev_codeword[error_pos] == '1':
            rev_codeword = rev_codeword[:error_pos] + '0' + rev_codeword[error_pos + 1:]
        else:
            rev_codeword = rev_codeword[:error_pos] + '1' + rev_codeword[error_pos + 1:]

        corrected_code = rev_codeword[::-1]
        print(f"Corrected Hamming Code: {corrected_code}")
        data = rev_codeword[2] + rev_codeword[4:7] + rev_codeword[8:]
        print(f"Correct code: {data}")

def exit_program():
    print("Exiting")
    return False
```

```
while True:
    print("1. Encode data to Hamming code\n2. Decode codeword\n3. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        Transmitter()
    elif choice == '2':
        Receiver()
    elif choice == '3':
        exit_program()
        break
    else:
        print("Error: Invalid choice")
```

Output/Result:

Output

```
1. Encode data to Hamming code
2. Decode codeword
3. Exit
Enter your choice: 1
Enter 7-bit data code: 1110001
Generated Hamming Code: 11101101001
1. Encode data to Hamming code
2. Decode codeword
3. Exit
Enter your choice: 2
Enter the Hamming code: 10010110111
p1: 0
p2: 0
p4: 0
p8: 0
Correct code: 1110001
1. Encode data to Hamming code
2. Decode codeword
3. Exit
Enter your choice: 3
Exiting

=== Code Execution Successful ===
```

Post Lab Question-Answers:

1) What are the different methods used for error detection
Ans.

Parity Check: Adds an extra bit (parity bit) to data to make the total number of 1's either even (even parity) or odd (odd parity).
Cyclic Redundancy Check (CRC): Uses polynomial division to detect errors in the transmitted data. It appends a CRC checksum to the data, and the receiver verifies it using the same divisor.
Checksum: Similar to CRC but involves adding all the data units and sending the sum (checksum) along with the data. The receiver performs the same addition and checks if the result matches the checksum.
Hamming Code: Uses extra bits to allow detection and correction of single-bit errors by calculating parity bits for different positions of data.
Two-Dimensional Parity Check: Uses parity checks across both rows and columns of a data block to detect errors.
Error-Correcting Codes (ECC): Includes forward error correction mechanisms like Reed-Solomon codes that allow for the detection and correction of multiple errors.

2)If the data unit is 111111 and the divisor is 1010,wht is the dividend at the Transmitter?
Ans.

In Cyclic Redundancy Check (CRC), the dividend is formed by appending zeroes to the original data unit. The number of zeroes is one less than the number of bits in the divisor.

- **Data unit (original message)**: 111111
- **Divisor**: 1010 (4 bits)

Since the divisor is 4 bits, the number of appended zeroes is 3 (4 - 1). Hence, the dividend is:

- **Dividend = 111111000**

3)Which layer of the OSI model usually does the function of error detection?
Ans
The **Data Link Layer (Layer 2)** of the OSI model is primarily responsible for error detection. It ensures reliable data transfer by detecting errors using mechanisms like CRC,

parity checks, and frame checksums. Some error detection may also occur at the **Transport Layer (Layer 4)**, depending on the protocol (e.g., TCP).

4) What is Hamming distance ? What is minimum Hamming distance?
Ans

Hamming Distance: The Hamming distance between two binary strings of equal length is the number of positions at which the corresponding bits differ. It measures how many bits need to be changed to convert one string into the other. For example, the Hamming distance between 10101 and 10011 is 2 because two bits are different.

Minimum Hamming Distance: In coding theory, the minimum Hamming distance of a code is the smallest Hamming distance between any two valid codewords in the set. This determines the code's error-detecting and error-correcting capabilities. For error detection, a code with a minimum Hamming distance of 2 can detect single-bit errors. For error correction, a minimum Hamming distance of 3 is required to correct single-bit errors.

Outcomes:

CO4: Execute their knowledge of computer communication principles, including Error detection and correction, multiplexing, flow control, and error control.

Conclusion (based on the Results and outcomes achieved):

From experiment 6, I learned how to detect and correct errors in data transmission using Hamming Code. I also understood the importance of redundancy bits in ensuring data accuracy and how error detection works in communication systems.

References:

References:

Books/ Journals/ Websites:

- Behrouz A Forouzan, Data Communication and Networking, Tata Mc Graw hill, India, 4th Edition
- A. S. Tanenbaum, ”Computer Networks”, 4th edition, Prentice Hall