**7. WAP to Implement Error Detection: LRC and Checksum**

**1. LRC (Longitudinal Redundancy Check)**

**Concept:**

* Data is divided into blocks of equal length.
* Perform bitwise XOR on corresponding bits across all blocks to generate the LRC.
* The LRC is appended to the transmitted data.

**Steps:**

1. Divide data into blocks of fixed size.
2. Calculate LRC by XORing each column of bits.
3. Append LRC to the data.

**Code Implementation for LRC:**

c

Copy code

#include <stdio.h>

#include <string.h>

void calculateLRC(char data[][9], int rows, char \*lrc) {

int colSum[8] = {0};

// Calculate LRC

for (int col = 0; col < 8; col++) {

for (int row = 0; row < rows; row++) {

colSum[col] ^= (data[row][col] - '0'); // XOR each bit column-wise

}

lrc[col] = colSum[col] + '0'; // Convert back to character

}

lrc[8] = '\0'; // Null-terminate the LRC

}

int main() {

char data[4][9] = { // Example binary data (8 bits per block)

"11001101",

"10101010",

"11110000",

"00001111"

};

char lrc[9];

calculateLRC(data, 4, lrc);

printf("Input Data Blocks:\n");

for (int i = 0; i < 4; i++) {

printf("%s\n", data[i]);

}

printf("LRC: %s\n", lrc);

return 0;

}

**2. Checksum**

**Concept:**

* Data is divided into blocks of fixed size.
* All blocks are added together (modulo 2^n, where n is the block size).
* Complement of the sum is the checksum.
* Checksum is appended to the transmitted data.

**Steps:**

1. Divide data into blocks of equal size.
2. Add all blocks together.
3. Calculate 1’s complement of the sum as the checksum.
4. Append checksum to the data.

**Code Implementation for Checksum:**

c

Copy code

#include <stdio.h>

#include <string.h>

// Function to calculate checksum

unsigned int calculateChecksum(int data[], int n) {

unsigned int sum = 0;

// Add all data blocks

for (int i = 0; i < n; i++) {

sum += data[i];

}

// Calculate the 1's complement of the sum

unsigned int checksum = ~sum;

return checksum;

}

int main() {

int data[] = {0x1234, 0x5678, 0x9ABC, 0xDEF0}; // Example data (16-bit blocks)

int n = sizeof(data) / sizeof(data[0]);

// Calculate checksum

unsigned int checksum = calculateChecksum(data, n);

printf("Input Data Blocks:\n");

for (int i = 0; i < n; i++) {

printf("0x%X\n", data[i]);

}

printf("Checksum: 0x%X\n", checksum);

return 0;

}

**3. Explanation of the Code**

**For LRC:**

1. **Data Blocks**: Input is divided into 8-bit blocks.
2. **LRC Calculation**:
   * Each column of bits is XORed across all blocks.
   * The result is the LRC, which is appended to the data.

**For Checksum:**

1. **Data Blocks**: Input is treated as 16-bit words.
2. **Sum**: All blocks are added together.
3. **Complement**: The 1’s complement of the sum is the checksum.
4. **Append**: The checksum is added to the data for verification.

**4. Compilation and Execution**

**Compile:**

bash

Copy code

gcc -o lrc lrc.c

gcc -o checksum checksum.c

**Execute:**

bash

Copy code

./lrc

./checksum

**5. Expected Output**

**For LRC:**

plaintext

Copy code

Input Data Blocks:

11001101

10101010

11110000

00001111

LRC: 10010010

**For Checksum:**

plaintext

Copy code

Input Data Blocks:

0x1234

0x5678

0x9ABC

0xDEF0

Checksum: 0xDCB2

**6. Notes**

* Both methods are basic error detection mechanisms and assume no data corruption during processing.
* **LRC** is more suitable for character-oriented data, while **Checksum** is used in block-oriented systems (e.g., TCP/IP).