Experiment – 5

AIM- To implement Cycle Redundancy Check.

Description - CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in communication channel. CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Another example is x2 + 1 that represents key 101.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Append the remainder at the end of the data to form the encoded data and send the same
3. Use modulo-2 binary division to divide binary data by the key and store remainder of division.

Receiver Side (Check if there are errors introduced in transmission):

Perform modulo-2 division again and if remainder is 0, then there are no errors.

In this article we will focus only on finding the remainder i.e. check word and the code word.

Modulo 2 Division:

The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is taken XOR with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Algorithms:

1. Input the data bits
2. nput the generator polynomial bits (e.g. 1 0 1 for x^2 + 1)
3. Initialise k = number of bits in generator polynomials
4. Initialise quotient[], remainder[]
5. Append k – 1 zeroes to the data
6. For each window of size k in data, take XOR with the generator polynomial
7. If the first bit in window is 1, append 1 to quotient, else 0
8. Splice the last k – 1 bits from data to get the remainder
9. Append the remainder to the original copy of the data to get the encoded data

Code:

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int n;
    cout<<"Enter the number of bits in data: ";
    cin>>n;
    int data[2*n];
    cout<<"Enter the data (space separated bits):  ";
    for(int i=0 ; i<n; i++){
        cin>>data[i];
    }
    int m;
    cout<<"Enter the number of bits in generator polynomial: ";
    cin>>m;
    int genPol[m];
    cout<<"Enter the generator polynomial (space separated bits): ";
    for(int i=0; i<m; i++){
        cin>>genPol[i];
    }
    cout<<endl;

    int sz = n;
    int temp[n];
    for(int i=0; i<n; i++){
        temp[i] = data[i];
    }
```

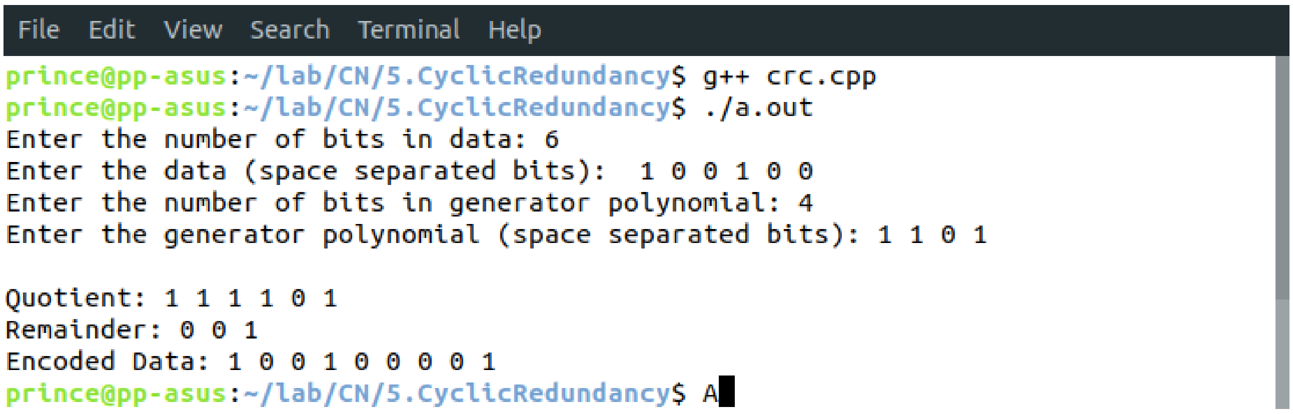
```
for(int i=0; i<m-1; i++){
    data[n++] = 0;
}
int rem[n+1];
int quo[n+1];
int jr = 0;
int jq = 0;
for(int i = 0; i < sz; ++i){
    int start = i;
    if(data[start] == 0){
        quo[jq++] = 0;
        continue;
    }
    quo[jq++] = 1;
    for(int k = 0 ; k < m; ++k){
        data[start] = data[start]^genPol[k];
        start++;
    }
}
cout<<"Quotient: ";
for(int i = 0 ; i < jq; ++i){
    cout<<quo[i]<<" ";
}
cout<<endl;

int i = 0;
for(i=0; i<m-1; i++){
    rem[jr++] = data[i+sz];
}
cout<<"Remainder: ";
for(i=0; i<jr; i++){
    cout<<rem[i]<<" ";
}
cout<<endl;

for(i=0 ; i<sz; i++){
    data[i] = temp[i];
}
for(i=0 ; i<jr; i++){
    data[sz++] = rem[i];
}
```

```
    cout<<"Encoded Data: ";
    for( i=0; i<sz; i++){
        cout<<data[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

Output:



```
prince@pp-asus:~/lab/CN/5.CyclicRedundancy$ g++ crc.cpp
prince@pp-asus:~/lab/CN/5.CyclicRedundancy$ ./a.out
Enter the number of bits in data: 6
Enter the data (space separated bits):  1 0 0 1 0 0
Enter the number of bits in generator polynomial: 4
Enter the generator polynomial (space separated bits): 1 1 0 1

Quotient: 1 1 1 1 0 1
Remainder: 0 0 1
Encoded Data: 1 0 0 1 0 0 0 0 1
prince@pp-asus:~/lab/CN/5.CyclicRedundancy$ A
```

Learnings –

1. Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if any error has occurred during transmission of the message.
2. Unlike other schemes, which are based on addition, CRC is based on binary division.
3. CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of messages delivered.
4. The device may take corrective action, such as rereading the block or requesting that it be sent again. Otherwise, the data is assumed to be error-free (though, with some small probability, it may contain undetected errors; this is the fundamental nature of error-checking).