Instituto Federal de Minas Gerais - Campus Avançado Ouro Branco
Curso: Bacharelado em Sistemas de Informação
Disciplina: Arquitetura e Organização de Computadores
Professor: Saulo Henrique Cabral Silva (saulo.cabral@ifmg.edu.br)

Trabalho Prático 1

Transmissão de dados utilizando Cyclic Redundancy Check

Nome: Felipe Leal Vieira RA: 0034372

João Victor Dutra Martins Silva RA: 0076873

1.Introdução:

No sistema a ser implementado e fundamental a transmissão de dados, a operação a ser realizada, envolve a transferência confiável de informações entre um transmissor e um receptor. No entanto como vimos na matéria, a utilização do Cyclic Redundancy Check (CRC), e fundamental para que não ocorra envio de dados errados, pode ocorrer no envio de dados uma perca desses dados, de forma que os dados cheguem de forma erronia no receptor.

No problema que foi proposto, seria implementar um código, de forma que o CRC fosse utilizado, para lidar com esses erros e garantir a integridade dos dados, são utilizados métodos de detecção e correção de erros. Imitando como um transmissor e um receptor funcionam. O CRC consiste em realizar operações usando XOR, os dados a serem transmitidos são transformados em binário, assim adicionando bits de redundância aos dados transmitidos, de modo que o receptor possa verificar se os dados recebidos estão íntegros. Um ponto importante a ser citado, e que o Polinômio que foi utilizado, e o seguinte: 1 1 0 0 1.

Portanto, o desafio consiste em implementar o algoritmo CRC para garantir que o dado recebido seja corretamente verificado e, se necessário, corrigido, a fim de obter uma transmissão confiável e livre de erros. Em caso de detecção de erro, o receptor deve ser capaz de solicitar a retransmissão do dado comprometido, visando obter a informação completa sem interferências prejudiciais.

2. Documentação:

2.1. Transmissor:

Para iniciar, na parte do transmissor, foi criado a função “StreamCaracter”, esta função converte um caractere para um vetor de booleanos, onde cada elemento do vetor representa um bit do caractere na tabela ASCII. O vetor resultante tem tamanho fixo de 8 elementos, já que um caractere na tabela ASCII é representado por 8 bits. A conversão é feita iterando sobre cada bit do valor do caractere na tabela ASCII e armazenando-o como verdadeiro se for 1 ou falso se for 0.

```
13        //convertendo um símbolo para "vetor" de boolean (bits)
14        private boolean[] streamCaracter(char simbolo) {
15
16            //cada símbolo da tabela ASCII é representado com 8 bits
17            boolean bits[] = new boolean[8];
18
19            //convertendo um char para int (encontramos o valor do mesmo na tabela ASCII)
20            int valorSimbolo = (int) simbolo;
21            int indice = 7;
22
23            //convertendo cada "bits" do valor da tabela ASCII
24            while (valorSimbolo >= 2) {
25                int resto = valorSimbolo % 2;
26                valorSimbolo /= 2;
27                bits[indice] = (resto == 1);
28                indice--;
29            }
30            bits[indice] = (valorSimbolo == 1);
31
32            return bits;
33        }
```

Nessa função “geradorRuido” recebe um “array” de booleanos representando uma sequência de bits. Ela gera um número aleatório entre 0 e 4 (inclusive). Se o número gerado for maior que 1, significa que há uma probabilidade de 50% de um erro ocorrer. Se este for o caso, um índice aleatório dentro do “array” é selecionado e o bit correspondente é invertido, simulando assim a introdução de ruído na sequência de bits.

```
35          //não modifique (seu objetivo é corrigir esse erro gerado no receptor)
36          private void geradorRuido(boolean bits[]) {
37              Random geradorAleatorio = new Random();
38
39              //pode gerar um erro ou não..
40              if (geradorAleatorio.nextInt(5) > 1) {
41                  int indice = geradorAleatorio.nextInt(8);
42                  bits[indice] = !bits[indice];
43              }
44          }
```

A função “DadoBitsCRC”, inicializa os cálculos do CRC, dessa forma ela também e responsável por retornar os valores finais dos cálculos.

```
46      private boolean[] dadoBitsCRC(boolean bits[]) {
47
48          //Retornando o bit novo ja calculado e com os numeros de controle no final, na qual foi passado pelas funcoes adicionarZeros e calcularXOR
49          return adicionarZeros(bits);
50      }
```

Na função “adicionarZeros”, ela adiciona zeros ao final de um” array” de bits para permitir a realização da operação de codificação CRC. Além disso novo “array” de bits com os zeros foi criado para auxiliar no processo.

```
52      private boolean[] adicionarZeros(boolean bits[]) {
53          //Polinomio gerador 1 1 0 0 1
54          boolean[] polinomio = {true, true, false, false, true};
55          //Criando um novo vetor para armazenar os 0's adicionais junto ao dado
56          boolean bitsComZeros[] = new boolean[bits.length + polinomio.length - 1];
57          //Logica de adicao dos 0's ao dado original
58          for (int i = 0; i < bitsComZeros.length; i++) {
59              if (i < bits.length) {
60                  bitsComZeros[i] = bits[i];
61              }
62          }
63
64          return calcularXOR(bitsComZeros);
65
66      }
```

Na função “calculaXOR”, calcula o CRC de uma sequência de bits de entrada. Primeiro, define-se um polinômio gerador fixo, que é usado para o cálculo. Em seguida, os bits de entrada são copiados para um novo vetor para evitar alterações nos dados originais. O cálculo do XOR é realizado entre os bits de entrada e o polinômio gerador, iterando sobre cada bit de entrada e aplicando o XOR correspondente. Os resultados são armazenados em um vetor separado. Por fim, os bits de controle do CRC são substituídos pelos valores originais dos bits de entrada. O vetor resultante contendo os bits calculados é então retornado como saída da função.

```java
private boolean[] calcularXOR(boolean[] bitsComZeros) {
    //Polinomio gerador 1 1 0 0 1
    boolean polinomio[] = {true, true, false, false, true};
    //Criando um novo vetor para o manuseio dos bits sem alterar o vetor original
    boolean bitsCalculados[] = new boolean[bitsComZeros.length];
    //Fazendo a copia dos bits para o vetor bitsCalculados[]
    System.arraycopy(bitsComZeros, 0, bitsCalculados, 0, bitsComZeros.length);

    //Realizando o processo do calculo do XOR
    for (int i = 0; i < bitsComZeros.length - polinomio.length + 1; i++) {
        //Verificando se o bit atual é 1
        if (bitsComZeros[i]) {
            //Inicializando uma variável k para acompanhar a posição no polinomio
            int k = 0;
            //Loop atraves dos bits do polinomio
            for (int j = i; j < (i + polinomio.length); j++) {
                //Realizando o XOR entre o bit atual e o correspondente no polinomio
                bitsComZeros[j] = !(bitsComZeros[j] == polinomio[k]);
                //Incrementando k para acompanhar o próximo bit do polinomio
                k++;
            }
        }
    }
    //Logica para a substituicao dos 0's para os bits de controle do CRC
    for (int i = (bitsComZeros.length - polinomio.length + 1); i < bitsComZeros.length; i++) {
        bitsCalculados[i] = bitsComZeros[i];
    }

    return bitsCalculados;
}
```

Na função “enviaDado”, ela recebe a mensagem que usuário informou, transforma em Bits pela função “streamCaracter”, que será enviada para função “dadoBitsCRC” onde será iniciado o processo de cálculo do CRC, retornando então o valor do cálculo para um novo vetor. Nesse novo vetor, irá passar por um gerador de ruídos, simulado uma falha na transmissão do dado, isso ira ocorre dentro da função “geradorRuido”. Enquanto o receptor não retornar que o dado esta integro, o dado será reenviado, até que o dado seja enviado da forma correta.

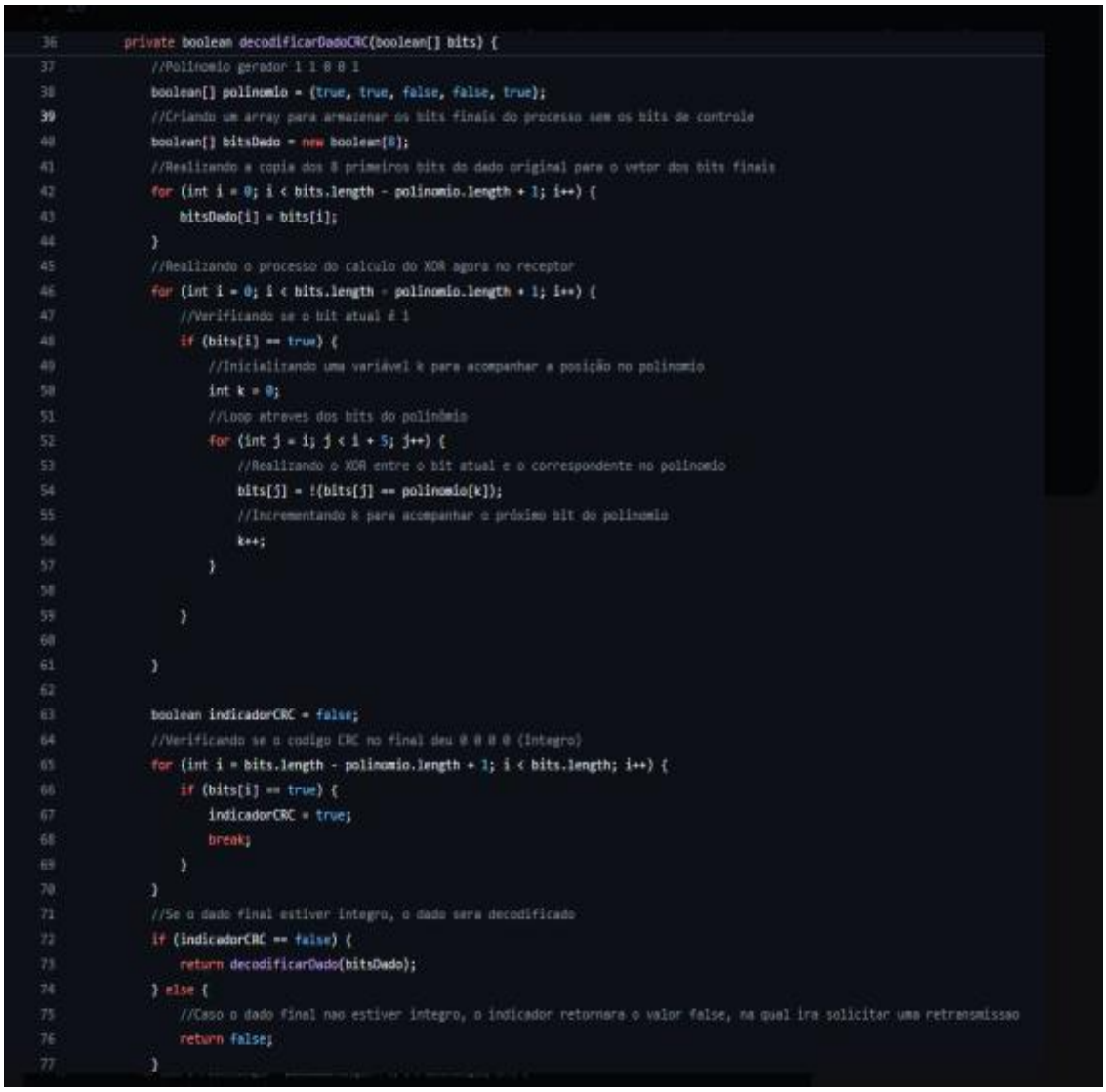
```java
public void enviaDado(Receptor receptor) {
    for (int i = 0; i < this.mensagem.length(); i++) {
        boolean bits[] = streamCaracter(this.mensagem.charAt(i));
        boolean bitsCRC[] = dadoBitsCRC(bits);

        //add ruidos na mensagem a ser enviada para o receptor
        geradorRuido(bitsCRC);

        //enviando a mensagem "pela rede" para o receptor (uma forma de testarmos esse método)
        //Uma retransmissao do dado e solicitada enquanto o receptor encontrar erros e o dado nao estiver integro
        while (!receptor.receberDadoBits(bitsCRC)) {
            bitsCRC = dadoBitsCRC(bits);
            geradorRuido(bitsCRC);
        }
    }
}
}
```

2.1. Receptor:

Após passar os dados com o CRC incluso, entramos no receptor, que ira decodificar os dados recebidos, dessa forma na função “decodificarDado”, a função decodifica um conjunto de bits em um valor ASCII. Inicialmente, codigoAscii é definido como zero e expoente como o comprimento do array de bits menos um. Em seguida, itera-se sobre cada bit no array. Se um bit for verdadeiro (1), o valor correspondente à sua posição é calculado usando a fórmula 2 elevado à potência do expoente e adicionado a codigoAscii. Após isso, o expoente é decrementado para acompanhar a próxima posição do bit.

```
17      private boolean decodificarDado(boolean bits[]) {
18          int codigoAscii = 0;
19          int expoente = bits.length - 1;
20
21          //converntendo os "bits" para valor inteiro para então encontrar o valor tabela ASCII
22          for (int i = 0; i < bits.length; i++) {
23              if (bits[i]) {
24                  codigoAscii += Math.pow(2, expoente);
25              }
26              expoente--;
27          }
28
29          //concatenando cada simbolo na mensagem original
30          this.mensagem += (char) codigoAscii;
31
32          //esse retorno precisa ser pensado... será que o dado sempre chega sem ruído???
33          return true;
34      }
```

Na função “decodificarDadoCRC”, nessa parte do programa Este código implementa um algoritmo de verificação de redundância cíclica (CRC) para decodificar um dado recebido. Primeiro, o código copia os 8 primeiros bits do dado original para um novo array chamado bitsDado. Em seguida, realiza-se o cálculo do XOR entre os bits do dado e um polinômio gerador específico. Este processo simula a divisão polinomial. Após o cálculo do XOR, verifica-se se os últimos 5 bits do resultado são todos zeros, o que indicaria uma transmissão íntegra. Se os bits do CRC forem todos zeros, o dado é decodificado com sucesso usando a função decodificarDado(). Caso contrário, se algum dos bits do CRC for diferente de zero, o indicador indicadorCRC é definido como true, indicando que houve um erro na transmissão. Nesse caso, a função retorna false, indicando que uma retransmissão é necessária para garantir a integridade do dado.



```
36      private boolean decodificarDadoCRC(boolean[] bits) {
37          //Polinomio gerador 1 1 0 0 1
38          boolean[] polinomio = {true, true, false, false, true};
39          //Criando um array para armazenar os bits finais do processo sem os bits de controle
40          boolean[] bitsDado = new boolean[8];
41          //Realizando a copia dos 8 primeiros bits do dado original para o vetor dos bits finais
42          for (int i = 0; i < bits.length - polinomio.length + 1; i++) {
43              bitsDado[i] = bits[i];
44          }
45          //Realizando o processo do calculo do XOR agora no receptor
46          for (int i = 0; i < bits.length - polinomio.length + 1; i++) {
47              //Verificando se o bit atual é 1
48              if (bits[i] == true) {
49                  //Inicializando uma variável k para acompanhar a posição no polinomio
50                  int k = 0;
51                  //Loop atreves dos bits do polinômio
52                  for (int j = i; j < i + 5; j++) {
53                      //Realizando o XOR entre o bit atual e o correspondente no polinomio
54                      bits[j] = !(bits[j] == polinomio[k]);
55                      //Incrementando k para acompanhar o próximo bit do polinomio
56                      k++;
57                  }
58
59              }
60
61          }
62
63          boolean indicadorCRC = false;
64          //Verificando se o codigo CRC no final deu 0 0 0 0 (Integro)
65          for (int i = bits.length - polinomio.length + 1; i < bits.length; i++) {
66              if (bits[i] == true) {
67                  indicadorCRC = true;
68                  break;
69              }
70          }
71          //Se o dado final estiver integro, o dado sera decodificado
72          if (indicadorCRC == false) {
73              return decodificarDado(bitsDado);
74          } else {
75              //Caso o dado final nao estiver integro, o indicador retornara o valor false, na qual ira solicitar uma retransmissao
76              return false;
77          }
```

Nesse trecho, o “receberDadoBits,” inicializa a sequência de decodificação da mensagem, e retorna se a mensagem está integra ou não.

```
81      //recebe os dados do transmissor
82      public boolean receberDadoBits(boolean bits[]) {
83
84          //aqui você deve trocar o médodo decofificarDado para decoficarDadoCRC (implemente!!)
85          return decodificarDadoCRC(bits);
86      }
87  }
```

3. Conclusão:

Durante o trabalho, uma das coisas mais difíceis de entender logica utilizada para converter mensagens em bits usando a tabela ASCII, como vimos cada letra da mensagem, e transformada em 8 bits, sendo assim cada letra e enviada de forma separada. Além disso, tive dificuldade em desenvolver uma maneira simples e clara de fazer cálculos usando a operação XOR. Mesmo que o conceito seja simples. Basicamente comparar cada bit em dois conjuntos de bits, encontrar uma maneira de fazer isso de forma eficiente e fácil de entender foi um desafio. Para superar essas dificuldades, precisei dedicar bastante tempo para estudar e praticar, além de pedir ajuda aos colegas quando necessário.

Outro ponto a ser citado, e que durante o trabalho, escolhemos um polinômio "grande" para um problema de apenas 8 bits, isso acabou dificultado no momento de criação do código, no entanto esse desafio foi uma forma de melhorar nosso nível de entendimento da matéria, muitas vezes, ao utilizar o "debug" e saber se os cálculos e o programa estavam corretos, tivemos que fazer a operação no lápis e papel

Por fim final, isso nos ajudou a ficar mais confortável com esses aspectos técnicos do trabalho, além de agregar em nossos conhecimentos sobre a matéria.

4. Referência:

As imagens utilizadas no trabalho, foram retiradas do link abaixo:

https://github.com/Joao-Dutra/AOC-TrabalhoPratico1