**Program:**

def lrc():  
 bits\_store = []  
 temp\_list = []  
 final\_output = []  
 print("\n\*\*\*LRC\*\*\*")  
 no\_bits = int(input("Enter number of bits(multiples of 8) : "))  
 temp1 = input("Enter the data bits : ")  
 bits\_store = list(temp1)  
 a = int(len(bits\_store) / 8)  
 k = 0  
 count = 0  
 parity = input("Do you want ODD(O) or EVEN(E) parity ? : ")  
 for i in range(0, 8):  
 for j in range(0, a):  
 temp\_list.append(int(bits\_store[k]))  
 k = k + 8  
 if j == a - 1:  
 temp = parity\_check(temp\_list, parity.upper())  
 final\_output.append(temp)  
 temp\_list.clear()  
 count = count + 1  
 k = count  
 break  
 print("LRC is : " + ''.join([str(elem) for elem in final\_output]) + "\n")  
  
  
def vrc():  
 bits\_store = []  
 final\_output = []  
 start = 0  
 stop = 8  
 print("\n\*\*\*VRC\*\*\*")  
 no\_bits = int(input("Enter number of bits(multiples of 8) : "))  
 temp1 = input("Enter the data bits : ")  
 bits\_store = [int(i) for i in temp1]  
 parity = input("Do you want ODD(O) or EVEN(E) parity ? : ")  
 a = int(no\_bits / 8)  
 for i in range(0, a):  
 temp = parity\_check(bits\_store[start:stop], parity.upper())  
 final\_output.append(temp)  
 start = start + 8  
 stop = stop + 8  
 print("VRC is : " + ''.join([str(elem) for elem in final\_output]) + "\n")  
  
  
def parity\_check(temp\_list, parity):  
 count = 0  
 for i in range(0, len(temp\_list)):  
 if temp\_list[i] == 1:  
 count = count + 1  
 a = count % 2  
 if parity == 'E':  
 return a  
 if parity == 'O':  
 if a == 1:  
 return 0  
 else:  
 return 1  
  
  
def xor(a, b):  
 result = []  
 for i in range(1, len(b)):  
 if a[i] == b[i]:  
 result.append('0')  
 else:  
 result.append('1')  
  
 return ''.join(result)  
  
  
def crc():  
 print("\n\*\*\*CRC\*\*\*")  
 frame\_size = int(input("Enter frame size : "))  
 data = input("Enter frame : ")  
 if len(data) != frame\_size:  
 crc()  
 generator\_size = int(input("Enter generator size : "))  
 key = input("Enter generator : ")  
 if len(key) != generator\_size:  
 crc()  
 zeroes\_appended = generator\_size - 1  
 print("SENDER SIDE:\nFrame = " + str(data) + "\nGenerator = " + str(key) + "\nNo of zeroes to be appended = " + str(  
 zeroes\_appended))  
 appended\_data = data + '0' \* (generator\_size - 1)  
 print("Message after appending zeroes = " + str(appended\_data))  
 remainder = mod2div(appended\_data, key)  
 codeword = data + remainder  
 print("CRC bits : ", remainder)  
 print("Transmitted frame is : ",codeword)

print(**"\n"**)

def mod2div(divident, divisor):  
 pick = len(divisor)  
 tmp = divident[0: pick]  
 while pick < len(divident):  
  
 if tmp[0] == '1':  
 tmp = xor(divisor, tmp) + divident[pick]  
 else:  
 tmp = xor('0' \* pick, tmp) + divident[pick]  
 pick += 1  
 if tmp[0] == '1':  
 tmp = xor(divisor, tmp)  
 else:  
 tmp = xor('0' \* pick, tmp)  
 checkword = tmp  
 return checkword  
  
  
def checksum():  
 bits\_store = []  
 temp\_list = []  
 final\_output = []  
 start = 0  
 binary1 = "0b00000000"  
 print("\n\*\*\*CHECKSUM\*\*\*")  
 segment\_size = int(input("Enter the segment size : "))  
 no\_bits = int(input("Enter number of bits(multiples of 8) : "))  
 temp1 = input("Enter the data bits : ")  
 bits\_store = [int(i) for i in temp1]  
 a = int(no\_bits / 8)  
 for i in range(0, a):  
 temp\_list = bits\_store[start:segment\_size]  
 start = start + 8  
 segment\_size = segment\_size + 8  
 binary2 = list\_to\_str(temp\_list)  
 integer\_sum = int(binary1, 2) + int(binary2, 2)  
 binary1 = bin(integer\_sum)  
 temp\_list.clear()  
 temp\_list = list(binary1)  
 temp\_list = temp\_list[2:]  
 if len(temp\_list) == 9:  
 var1 = temp\_list[0]  
 var2 = temp\_list[1:]  
 var1\_str = list\_to\_str(var1)  
 var2\_str = list\_to\_str(var2)  
 integer\_sum = int(var1\_str, 2) + int(var2\_str, 2)  
 temp\_list.clear()  
 temp\_list = list(bin(integer\_sum))  
 temp\_list = temp\_list[2:]  
 for i in range(0, len(temp\_list)):  
 a = int(temp\_list[i])  
 if a == 1:  
 final\_output.append(0)  
 if a == 0:  
 final\_output.append(1)  
 print("CHECKSUM generated is : " + ''.join([str(elem) for elem in final\_output]) + "\n")  
  
  
def list\_to\_str(list\_str):  
 list\_to\_str = ''.join([str(elem) for elem in list\_str])  
 binary = "0b" + list\_to\_str  
 return binary  
  
  
def error():  
 while (True):  
 print("Press 1 for LRC")  
 print("Press 2 for VRC")  
 print("Press 3 for CRC")  
 print("Press 4 for Checksum")  
 choice = int(input("Enter your choice : "))  
 if choice == 1:  
 lrc()  
 elif choice == 2:  
 vrc()  
 elif choice == 3:  
 crc()  
 elif choice == 4:  
 checksum()  
 else:  
 print("\nWrong choice, re-enter\n")  
 error()  
  
  
print("ERROR DETECTION TECHNIQUES\n")  
error()

**Output:**

ERROR DETECTION TECHNIQUES

Press 1 for LRC

Press 2 for VRC

Press 3 for CRC

Press 4 for Checksum

Enter your choice : 1

\*\*\*LRC\*\*\*

Enter number of bits(multiples of 8) : 16

Enter the data bits : 1110011111011101

Do you want ODD(O) or EVEN(E) parity ? : E

LRC is : 00111010

Press 1 for LRC

Press 2 for VRC

Press 3 for CRC

Press 4 for Checksum

Enter your choice : 3

\*\*\*CRC\*\*\*

Enter frame size : 8

Enter frame : 10101010

Enter generator size : 4

Enter generator : 1101

SENDER SIDE:

Frame = 10101010

Generator = 1101

No of zeroes to be appended = 3

Message after appending zeroes = 10101010000

CRC bits : 110

Transmitted frame is : 10101010110

Press 1 for LRC

Press 2 for VRC

Press 3 for CRC

Press 4 for Checksum

Enter your choice : 4

\*\*\*CHECKSUM\*\*\*

Enter the segment size : 8

Enter number of bits(multiples of 8) : 16

Enter the data bits : 1010100100111001

CHECKSUM generated is : 00011101