In [1]:
```python
def prepare_word(input_word: str) -> str:
    """
    The function fills every '2^n' bytes with zeros and prepares the word for enc
oding.
    """

    n = 0
    while 2 ** n < len(input_word):
        i = (2 ** n) - 1
        n += 1
        input_word = input_word[:i] + '0' + input_word[i:]

    return input_word

def split_word_on_chucks(prepared_word: str, hamming_byte_index: int) -> list:
    """
    Every Hamming byte is responsible for some bytes.
    The function splits prepared word on chunks for which Hamming byte is respons
ible for.
    """

    length = 2 ** hamming_byte_index
    start = length - 1
    end = start + length if (start + length) <= len(prepared_word) else len(prepa
red_word)
    word_chunk = [prepared_word[i:(i + length)] for i in range(start, len(prepare
d_word), 2*length)]

    return word_chunk

def calculate_bytes_in_word(word_chunk: list) -> int:
    """
    The function calculates bytes in word chunks.
    """

    counter = 0
    for word in word_chunk:
        for byte in word:
            counter += int(byte)

    return counter

def hamming_encode(input_word: str) -> str:
    """
    The function adds additional bytes at '2^n' indexes.
    These bytes allows to check if sent message is correct or not by checking res
pondible bytes.
    Every Hamming byte is responsible for 'n' bytes starting from 'n'th index and
 repeating after n bytes.

    E.g. 1st byte is reponsible for 1, 3, 5, 7 and etc bytes.
    And for 11th byte are responsible 1, 2 and 8 bytes.
    11 = 1 + 2 + 8
    """
    prepared_word = prepare_word(input_word)

    n = 0
    while 2 ** n < len(prepared_word):
        length = 2 ** n
        start = length - 1
        word_chunk = split_word_on_chucks(prepared_word, n)
        counter = calculate_bytes_in_word(word_chunk)

        if counter % 2:
            prepared_word = prepared_word[:start] + '1' + prepared_word[length:]
        n += 1

    return prepared_word
```

```python
print(hamming_encode('1001000'))
print(hamming_encode('1100001'))
print(hamming_encode('1101101'))
print(hamming_encode('1101001'))
print(hamming_encode('1100111'))
```

```
00110010000
10111001001
11101010101
01101011001
01111001111
```

In [2]:

```python
def xor(a: str, b: str) -> str:
    """
    The function makes XOR operation for input bytes.
    """
    result = ''

    for i in range(len(b)):
        result += str(int((a[i] != b[i])))

    return result


def calculate_remainder(word: str, divisor: str):
    """
    The function calculates remainder by dividing input bytes with a key.
    In boolean algebra division is achieved with XOR operation.
    """
    pick = len(divisor)
    divident = word + '0'*(pick-1)
    start = 0
    word = divident[:pick]

    while start + pick < len(divident):
        if word[0] == '1':
            word = (xor(word, divisor) + divident[start+pick])[1:]
        else:
            word = word[1:] + divident[start+pick]

        start += 1

    if word[0] == '1':
        word = xor(word, divisor)

    return word[1:]

def crc_encode(word: str, key: str) -> str:
    """
    The function encodes string using CRC algorithm.
    """
    remainder = calculate_remainder(word, key)

    return word + remainder

print(crc_encode('1101011011', '10011'))
print(crc_encode('1100110011', '10011'))
print(crc_encode('1101111011', '10011'))
print(crc_encode('1101101111', '10011'))
print(crc_encode('1001111011', '10011'))
```

```
11010110111110
11001100111000
11011110110100
11011011111101
10011110111011
```