LAB EXERCISE - 2

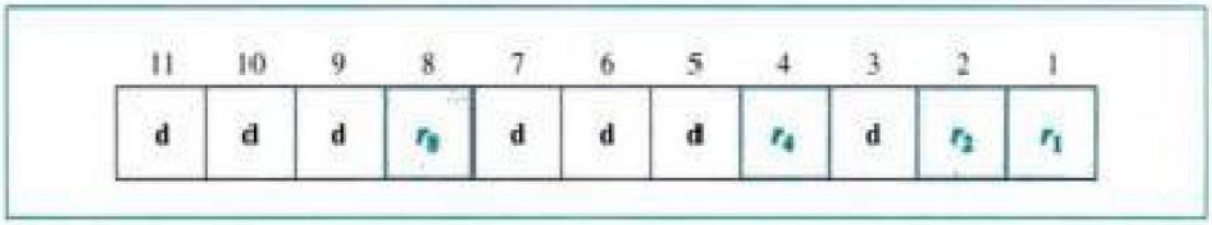
Write a Program in C/ C++ for hamming code generation for error detection/correction

---

Hamming Code

Hamming provides a practical solution. The **Hamming code** can be applied to data units of any length and uses the relationship between data and redundancy bits discussed above. For example, a 7-bit ASCII code requires 4 redundancy bits that can be added to the end of the data unit or interspersed with the original data bits. In Figure 10.14, these bits are placed in positions 1, 2, 4, and 8 (the positions in an 11-bit sequence that are powers of 2). For clarity in the examples below, we refer to these bits as r_1, r_2, r_4, and r_8.

Figure 10.14 *Positions of redundancy bits in Hamming code*

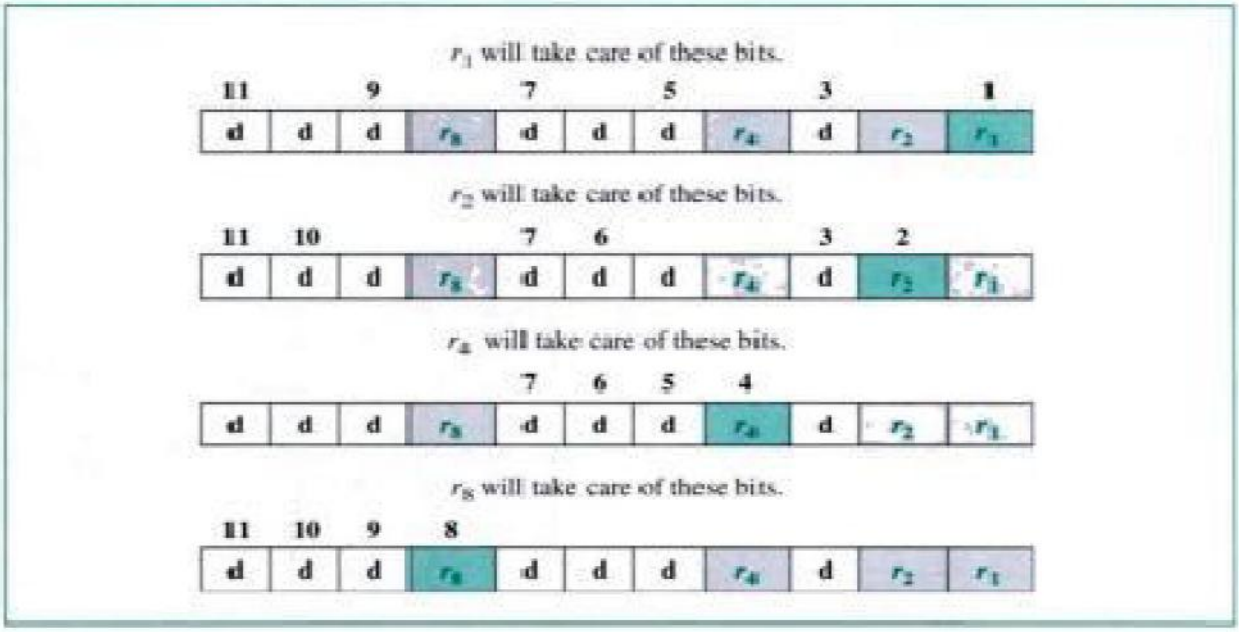


In the Hamming code, each r bit is the parity bit for one combination of data bits, as shown below:

r_1: bits 1, 3, 5, 7, 9, 11
r_2: bits 2, 3, 6, 7, 10, 11
r_4: bits 4, 5, 6, 7
r_8: bits 8, 9, 10, 11

Each data bit may be included in more than one calculation. In the sequences above, for example, each of the original data bits is included in at least two sets, while the r bits are included in only one (see Fig. 10.15).

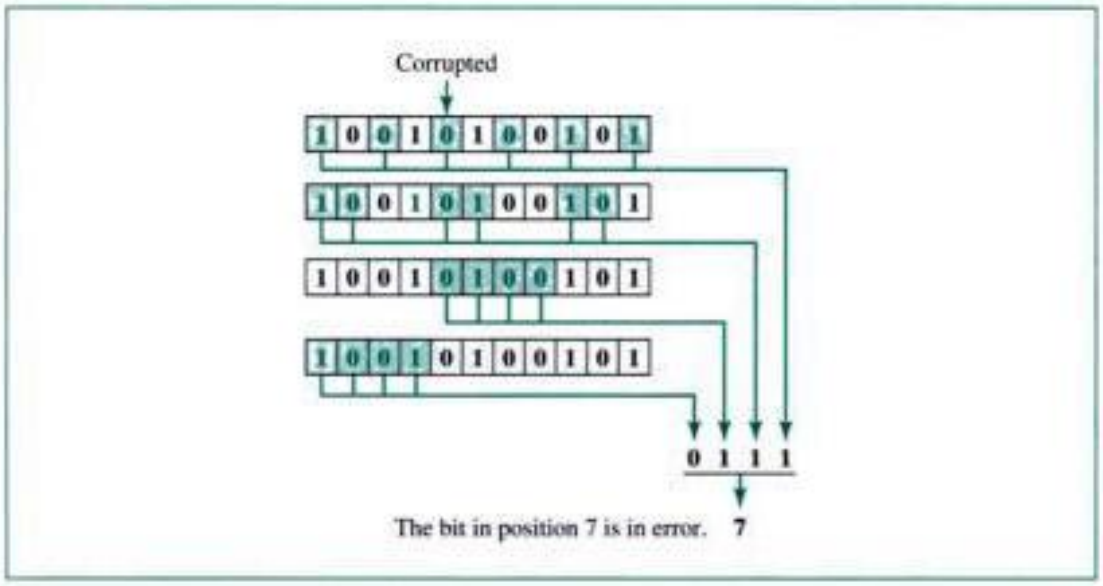
Figure 10.15 *Redundancy bits calculation*



Calculating the *r* Values Figure 10.16 shows a Hamming code implementation for an ASCII character. In the first step, we place each bit of the original character in its appropriate position in the 11-bit unit. In the subsequent steps, we calculate the even parities for the various bit combinations. The parity value for each combination is the value of the corresponding *r* bit.

Error Detection and Correction Now imagine that by the time the above transmission is received, the number 7 bit has been changed from 1 to 0. The receiver takes the

Figure 10.17 *Error detection using Hamming code*



Once the bit is identified, the receiver can reverse its value and correct the error. The beauty of the technique is that it can easily be implemented in hardware and the code is corrected before the receiver knows about it.

Expected Output:

```
sr@msritcse:~$ ./a.out
Enter 4 bits of data one by one
1
0
1
0

Encoded data is
1010010

Enter received data bits one by one
1
0
1
0
0
1
0

No error while transmission of data
sr@msritcse:~$ ./a.out
Enter 4 bits of data one by one
1
0
1
0

Encoded data is
1010010

Enter received data bits one by one
1
0
1
0
1
1
0

Error on position 3
Data sent : 1010010
Data received : 1010110
Correct message is
1010010sr@msritcse:~$
```