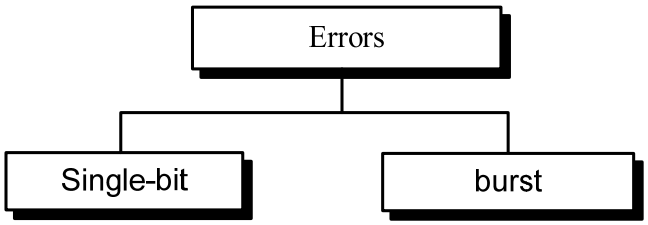
CHƯƠNG 9:
PHÁT HIỆN VÀ SỬA LỖI

Việc phát hiện và sửa lỗi được thiết lập ở **lớp kết nối dữ liệu** hoặc **lớp vận chuyển** trong mô hình OSI.

9.1 CÁC DẠNG LỖI

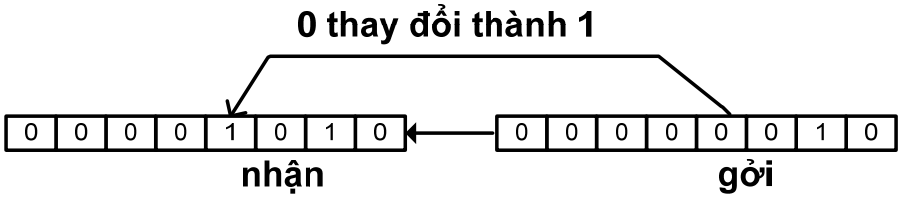
Có 2 dạng lỗi: Lỗi một bit và lỗi nhiều bit (burst)



+ Lỗi một bit: **Chỉ có một bit bị sai trong một đơn vị dữ liệu (byte, ký tự, đơn vị dữ liệu, hay gói)**

Ví dụ: thay đổi từ 1 → 0 hoặc từ 0 → 1.

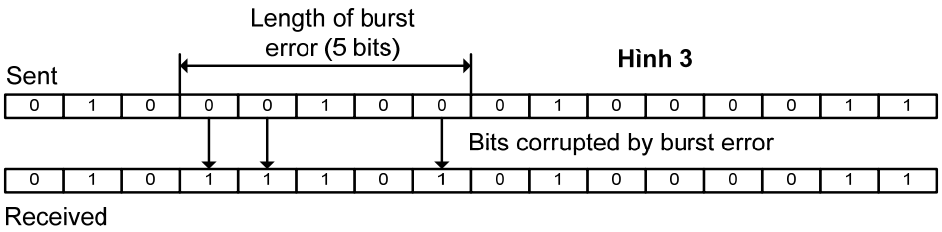
00000010 (STX: start of text) khi bị sai 1 bit dữ liệu nhận được 00001010 (LF: line feed)



Lỗi một bit ít xuất hiện trong phương thức truyền nối tiếp. Thường xuất hiện trong truyền song song.

+ Lỗi bệt: **có hai hoặc nhiều bit sai trong đơn vị dữ liệu.**

Nhiễu bệt không có nghĩa là các bit bị lỗi liên tục, chiều dài của bệt tính từ bit sai đầu tiên cho đến bit sai cuối. Một số bit bên trong bệt có thể không bị sai.

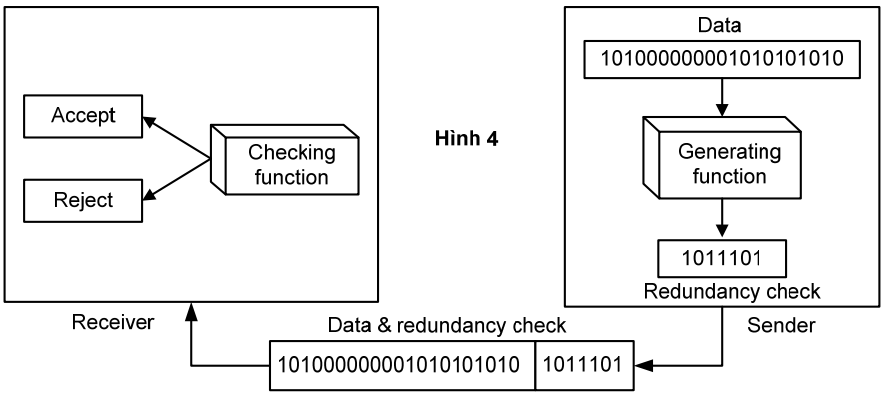


Hình 9.1

Nhiễu bệt thường xuất hiện trong truyền nối tiếp.

9.2 PHÁT HIỆN LỖI

+ Mã thừa (Redundancy)

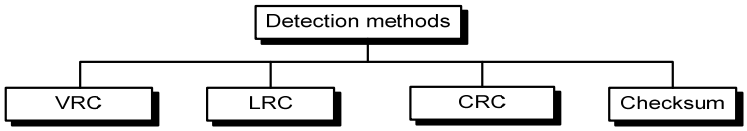


- ***Ý tưởng thêm các thông tin phụ vào trong bản tin chỉ nhằm mục đích giúp kiểm tra lỗi.***
- ***Mã thừa sẽ được loại bỏ sau khi đã xác định xong độ chính xác của quá trình truyền.***

Có bốn dạng kiểm tra lỗi cơ bản dùng mã thừa trong truyền dữ liệu:

- **VRC** (vertical redundancy check): kiểm tra tính chẵn lẻ của tổng bit '1' trong một đơn vị dữ liệu.
- **LRC** (longitudinal redundancy check): kiểm tra tính chẵn lẻ của tổng các bit '1' trong một khối.
- **CRC** (cyclic redundancy check) : kiểm tra chu kỳ dư.
- **Checksum**: kiểm tra tổng.

Ba dạng đầu, VRC, LRC, và CRC thường được thiết lập trong lớp vật lý để dùng trong lớp kết nối dữ liệu. Dạng checksum thường được dùng trong các lớp trên.



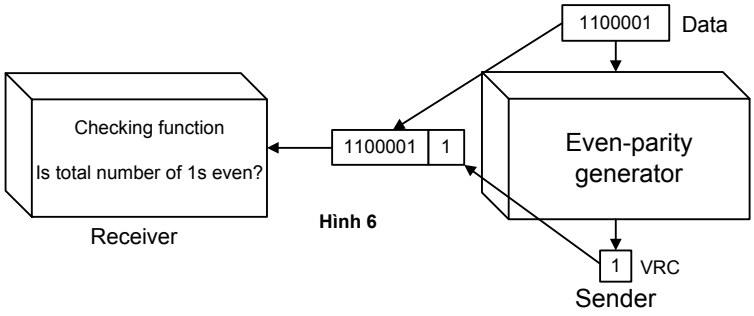
9.3 VRC (*kiểm tra parity (chẵn/lẻ)*)

Thêm một bit (0 hoặc 1) vào đơn vị dữ liệu sao cho tổng số bit '1' là một số chẵn.

Đặc điểm: Một bit thừa (bit parity) được gắn thêm vào các đơn vị dữ liệu sao cho tổng số bit '1' trong đơn vị dữ liệu (bao gồm bit parity) là một số chẵn (even).

- Giả sử ta muốn truyền đơn vị dữ liệu nhị phân **1100001** [ASCII là a (97)]; **1100011** [ASCII là c (99)];
- Ta thấy tổng số bit 1 là 3 (a), tức là một số lẻ; tổng số bit 1 là 4 (c), tức là một số chẵn.
- Trước khi truyền, ta cho đơn vị dữ liệu qua bộ tạo bit parity, để gắn thêm vào đơn vị dữ liệu một bit, làm tổng số bit 1 là số chẵn.

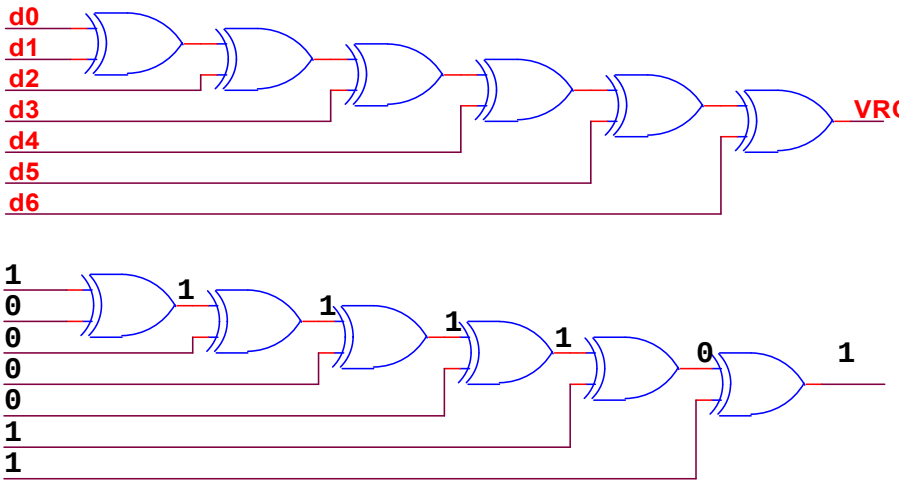
- Hệ thống truyền dữ liệu với parity bit này vào đường truyền: 1100001*1*, **1100011*0***
- Thiết bị thu, sau khi nhận sẽ đưa đơn vị dữ liệu sang hàm kiểm tra parity chẵn.
- Nếu dữ liệu nhận được có tổng số bit 1 là số chẵn thì chấp nhận.
- Nếu dữ liệu nhận được có tổng số bit 1 là số lẻ thì loại toàn đơn vị dữ liệu.



Hình 9.2

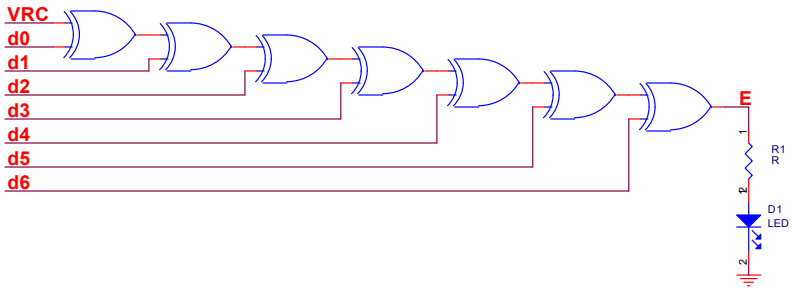
+ Mạch tạo bit Parity chẵn (VRC):

Ví dụ: Mạch tạo bit VRC của một dữ liệu 7 bit: 1100001

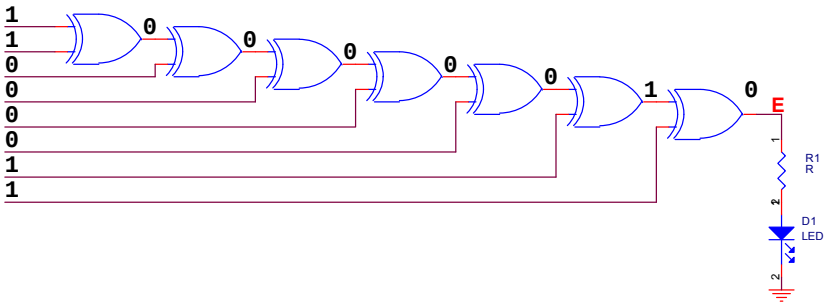


+ Mạch kiểm tra bit Parity chẵn (VRC):

Ví dụ: Mạch kiểm tra VRC của một dữ liệu 8 bit: 1100001*1*.



Nếu E=1 dữ liệu sai, E=0 dữ liệu đúng.



Ví dụ 1:

Giả sử ta muốn truyền từ "world" trong mã ASCII, năm ký tự này được mã hóa như sau:

Bốn ký tự đầu có số bit một là chẵn, nên có bit parity là 0, còn ký tự cuối có số bit 1 là lẻ nên có bit parity là 1 (các bit parity được gạch dưới)

Ví dụ 2:

Giả sử ký tự tạo được từ Ví dụ 1 được máy thu nhận được như sau:

Máy thu đếm số bit 1 và nhận ra có số bit một là chẵn và lẻ, phát hiện có lỗi, nên loại bản tin và yêu cầu gởi lại.

+ Hiệu năng:

- ***VRC có thể phát hiện lỗi 1 bit.***
- *Đồng thời cũng có thể phát hiện các lỗi bệt mà tổng số bit sai là số lẻ (1, 3, 5, v,v....)*

Ví dụ:

1000111011,

- Nếu có ba bit thay đổi thì kết quả sẽ là lẻ và máy thu phát hiện ra được: 1*111*111011: 9 *0110* 0111011:7
- Trường hợp hai bit bị lỗi: 1*11*0111011:8 1*1*00*0*11011:6 1000*0*1101*0*:4

Máy thu không phát hiện được ra lỗi và chấp nhận.

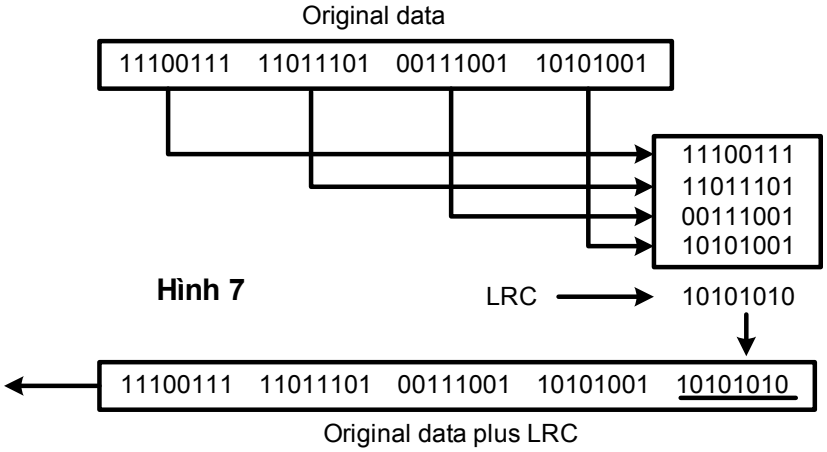
9.4 LRC

LRCKiểm tra một khối bit. Khối bit được sắp xếp thành bảng (hàng và cột).

+Tạo LRC:

Ví dụ: Gởi một khối có 32 bit

- Sắp xếp dữ liệu thành 4 hàng và 8 cột.
- Tìm bit VRC cho mỗi cột
- Tạo một hàng mới gồm 8 bit, đó là LRC
- Gởi kèm LRC vào cuối dữ liệu.

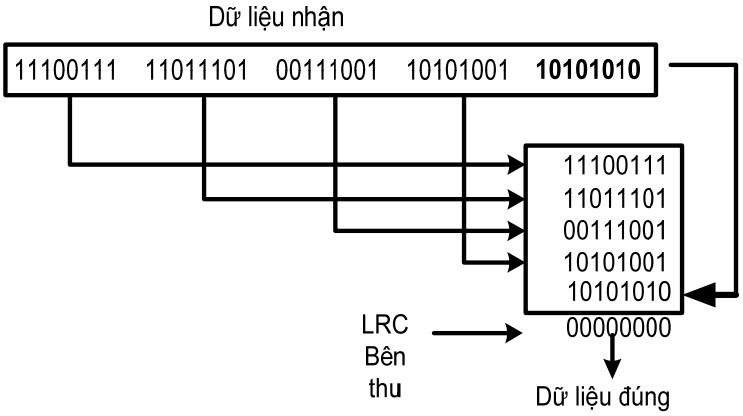


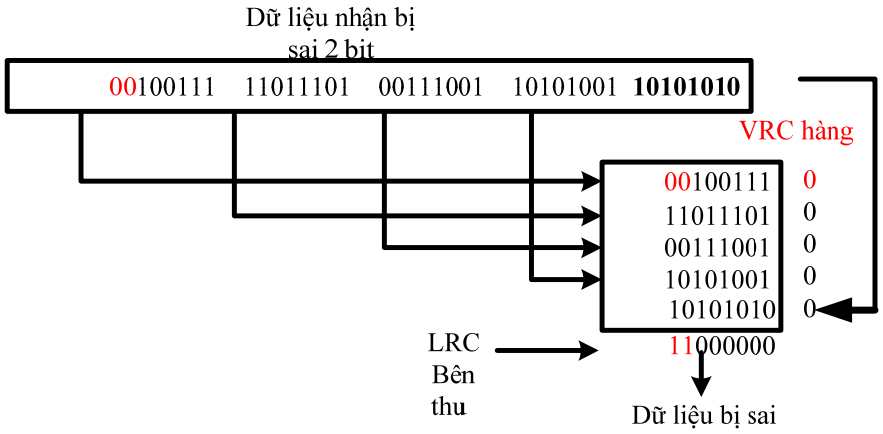
Hình 7

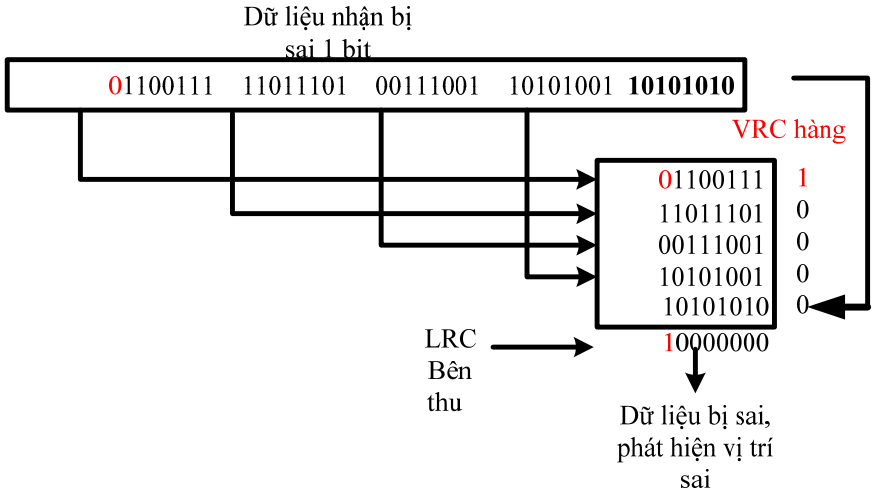
+Kiểm tra LRC

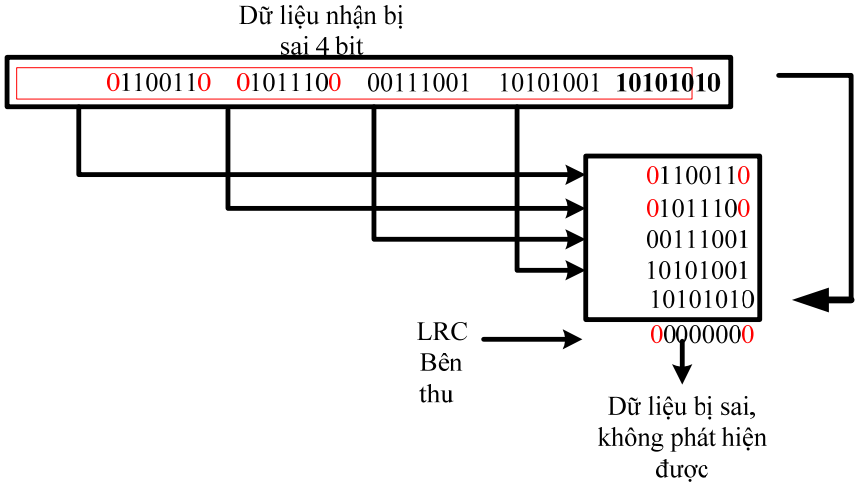
Ví dụ: Thu một khối có 40 bit

- Sắp xếp dữ liệu nhận được thành 5 hàng và 8 cột (giống bên phát).
- Tìm bit VRC cho mỗi cột, nếu VRC bằng 1 thì dữ liệu bị sai.
- Nếu VRC của mỗi cột bằng 0 thì dữ liệu đúng.
- Nếu LRC bên thu là zêrô thì dữ liệu đúng. Ngược lại dữ liệu bị sai.



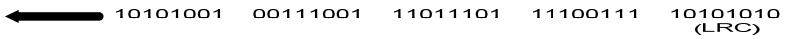




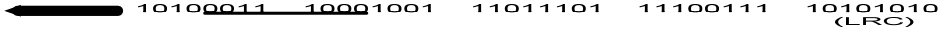


Ví dụ 3:

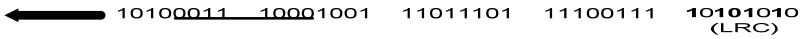
Giả sử khối bit truyền đi là:



Tuy nhiên, có nhiễu bệt độ dài tám bit xuất hiện, làm một số bit bị lỗi:



Khi máy thu kiểm tra LRC, một số bit không theo đúng parity chẵn và toàn khối bị loại (các giá trị sai được in đậm)



+ Hiệu năng:

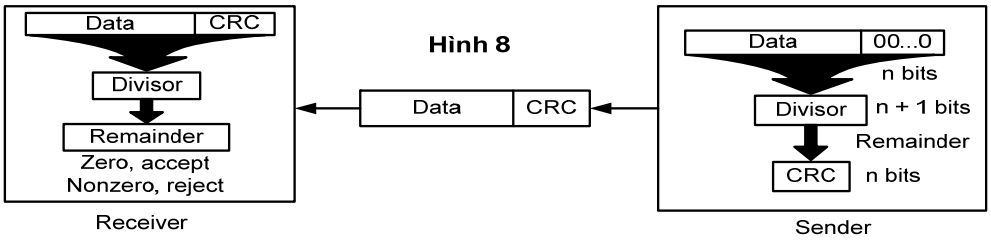
- **LCR cho phép phát hiện lỗi bệt**.
- **Khi hai (số chẵn) bit cùng sai ở các vị trí giống nhau trong một đơn vị dữ liệu thì LRC không phát hiện được**.

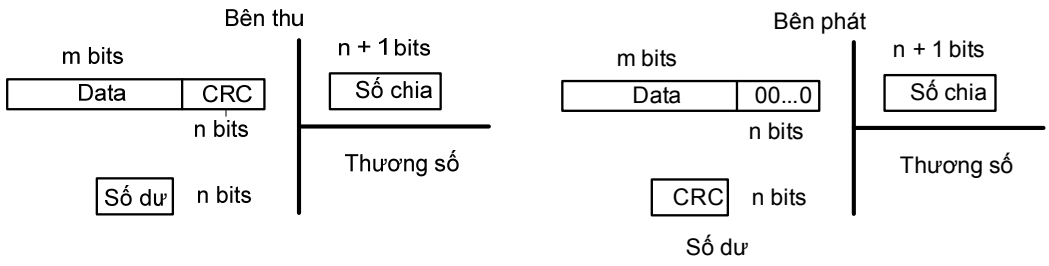
Thí dụ, hai đơn vị dữ liệu: 11110000 và 11000011. Nếu bit đầu và bit cuối của hai đơn vị đều bit lỗi, tức là dữ liệu nhận được là *0*1110001 và *0*1000010 thì LCR không thể phát hiện được lỗi.

9.5 CRC (CYCLIC REDUNDANCY CHECK):

+ Sơ đồ khối của Bên phát và Bên thu của phương pháp CRC:

- **Divisor:** số chia (đa thức sinh), có số bit là **n+1**; Dữ kiện cho trước, giống nhau ở bên phát và bên thu.
- **CRC:** số dư của phép chia bên phát, có số bit là **n**.
- **Remainder:** số dư phép chia bên thu. Nếu số dư này zêrô →dữ liệu thu không bị sai, ngược lại dữ liệu thu bị sai.
- **Data:** Dữ liệu cần mã hoá lỗi CRC.





Số dư bằng zêrô thì dữ liệu thu đúng, ngược lại dữ liệu bị sai

Các bit thừa trong dạng mã hoá CRC có được bằng cách chia đơn vị dữ liệu với một số chia (divisor) cho trước và dư số là CRC. Yêu cầu đối với CRC gồm hai yếu tố:

- Có số bit nhỏ hơn số bit bộ chia một bit.
- Được gắn vào cuối chuỗi dữ liệu

+Các bước tìm CRC:

- Thêm n bit '0' vào đơn vị dữ liệu, số n này nhỏ hơn một so với (n+1) bit của bộ chia (divisor).
- Dữ liệu mới này được chia cho số chia dùng phép chia nhị phân. Kết quả có được chính là CRC.
- CRC với n bit của bước hai thay thế các bit 0 gắn ở cuối đơn vị dữ liệu (CRC có thể chứa toàn bit '0').

+Tại máy thu:

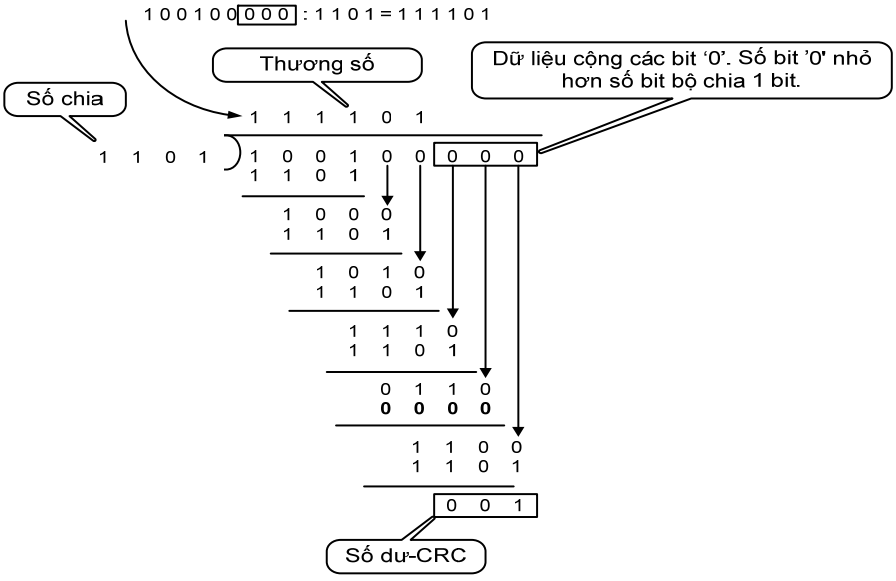
- Đơn vị dữ liệu đến máy thu với phần đầu là dữ liệu, tiếp đến là CRC. Máy thu xem toàn chuỗi này là một đơn vị và đem chia chuỗi cho cùng số chia đã được dùng tạo CRC.
- Khi chuỗi dữ liệu đến máy thu không lỗi, thì bộ kiểm tra CRC có số dư là 0 và chấp nhận đơn vị dữ liệu này.
- Khi chuỗi bị thay đổi trong quá trình truyền, thì số dư sẽ khác không và bộ thu không chấp nhận đơn vị này.

9.5. 1 Bộ tạo CRC

Bộ CRC dùng phép chia modulo–2. Trong bước đầu, bộ chia bốn bit được trừ đi. Mỗi bit trong bộ chia được trừ với các bit tương ứng mà không ảnh hưởng đến bit kế tiếp. Trong Ví dụ này, bộ chia 1101, được trừ từ bốn bit của số bị chia 100, có được 100 (bit 0 đầu bị bỏ qua).

Bước kế tiếp, lấy 1000 – 1101, thực hiện tương tự nhu phép chia.

Trong quá trình này, bộ chia luôn bắt đầu với bit 1; và hệ thống thực hiện phép chia theo cách trừ nhị phân không có số nhớ (tức là 0 – 0 = 0; 1 – 1 = 0; 0 – 1 = 1; 1 – 0 =1).



Hình 9.3

Ví dụ: Cho một dữ liệu X: **100100**, được mã hóa lỗi theo dạng CRC với số chia (đa thức sinh) có dạng **1101**.

a. Tìm CRC.

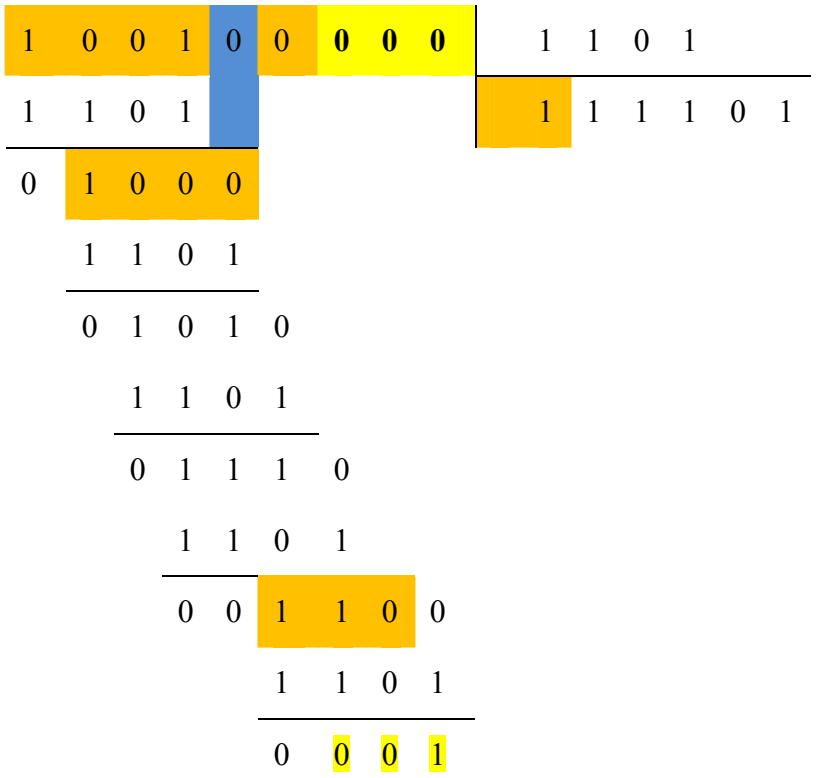
b. Tìm chuỗi dữ liệu phát.

c. Giả sử máy thu nhận 2 chuỗi dữ liệu **Y**: **100100001 và Z: 111100001;** Hãy cho biết chuỗi dữ liệu nào đúng và chuỗi dữ liệu nào sai? Giải thích.

Giải

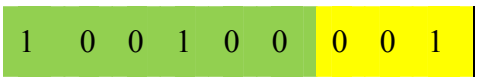
a. Tìm CRC;

Số bit của số chia là 4, suy ra n = 4-1=3, thêm vào dữ liệu 3 bit '0'



Vậy CRC là **001**

b. Tìm chuỗi dữ liệu phát theo dạng CRC



c. Giả sử máy thu nhận 2 chuỗi dữ liệu Y: 100100001; Z: 111100001; Hãy cho biết chuỗi dữ liệu nào đúng và chuỗi dữ liệu nào sai.

+ Dữ liệu Y: **100100001**

```
1 0 0 1 0 0 0 0 1 | 1 1 0 1
1 1 0 1           | 1 1 1 1 0 1
-----------------
0 1 0 0 0
  1 1 0 1
  ---------
  0 1 0 1 0
    1 1 0 1
    -------
    0 1 1 1 0
      1 1 0 1
      -------
      0 0 1 1 0 1
        1 1 0 1
        -------
        0 0 0 0
```

Số dư bên thu là Zêrô → Dữ liệu Y đúng.

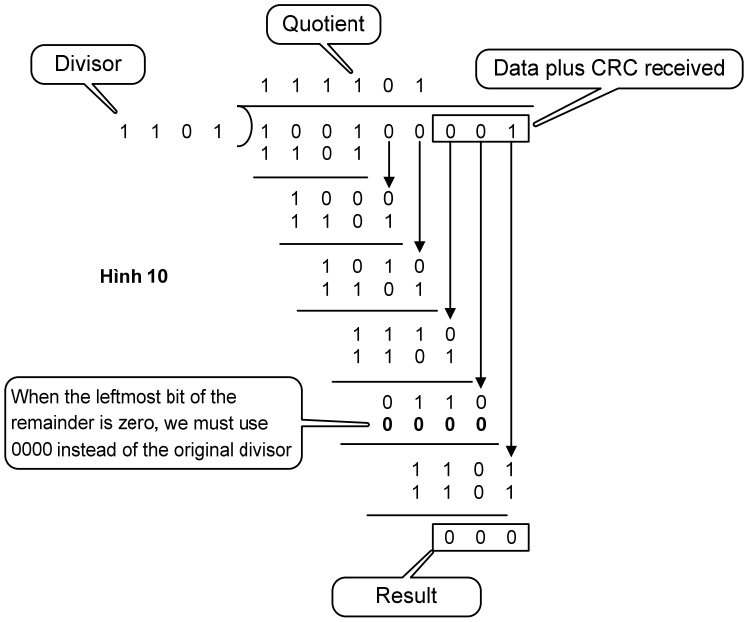
+ Dữ liệu Z: **111100001;**

```
1 1 1 1 0 0 0 0 1 | 1 1 0 1
1 1 0 1           | 1 0 1 1 1
-----------------
0 0 1 0 0
  0 0 0 0
  ---------
  0 1 0 0 0
    1 1 0 1
    -------
    0 1 0 1 0
      1 1 0 1
      -------
      0 1 1 1 0
        1 1 0 1
        -------
        0 0 1 1 1
```

Số dư bên thu là 111≠zêrô → dữ liệu Z sai.

9.5. 2 Bộ kiểm tra CRC

Bộ này hoạt động giống hệt như bộ phát. Sau khi nhận được giữa liệu có gắn thêm phần CRC, mạch thực hiện lại phép chia modulo – 2. Nếu kết quả là 0, cắt bỏ phần CRC và nhận dữ liệu; ngược lại thì loại bỏ dữ liệu và yêu cầu gởi lại. Giả sử là không có lỗi, dư số là 0 và dữ liệu được chấp nhận.



Hình 9.4

9.5. 3 Các đa thức:

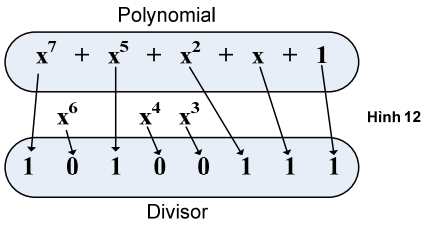
Bộ tạo CRC (bộ chia) thường không chỉ là chuỗi các bit 1 và 0, nhưng tạo ra từ đa thức đại số. Các đa thức này tiện lợi vì hai lý do: Chúng thường ngắn và thường được dùng để chứng minh các ý niệm toán học trong quá trình CRC.

Đa thức của bộ chia:

$\sum$ (ký số. x^i); với i là vị trí của ký số, i= 0÷ n; bộ chia có n+1 bit.

$$x^7 + x^5 + x^2 + x + 1$$ Hình 11

Quan hệ giữa chuỗi đa thức với biểu diễn nhị phân được minh họa ở hình sau:



Một đa thức sinh của bộ chia cần được chọn theo các đặc tính sau:

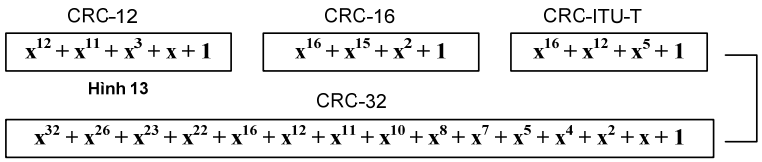
- ***Không được chia hết cho thức x***
- Chia đúng cho đa thức (x + 1)

Điều kiện đầu nhằm bảo đảm là tất cả các nhiễu bệt có độ dài bằng bậc của đa thức sinh đều được phát hiện. Điều kiện thứ hai bảo đảm là tất cả các nhiễu bệt ảnh hưởng lên thứ tự bit lẻ được phát hiện.

Ví dụ 4:

Rõ ràng là ta không thể chọn x (số nhị phân 10) hay $x^2 + x$ (số nhị phân 110) làm đa thức được vì chúng chia hết cho x. Tuy nhiên, ta có thể chọn x+1 (tương ứng 11) do không chia hết cho x, mà chia hết cho (x+1), cũng như ta có thể chọn $x^2 + 1$ (số nhị phân 101) do chia hết cho (x+1).

Các đa thức chuẩn dùng trong bộ chia CRC được minh họa trong hình 13. Các số 12, 16, và 32 có liên quan đến kích thước của dư số CRC. Bộ chia CRC tương ứng là 13, 17 và 33 bit.



Hình 9.5

Hiệu năng:

CRC là phương pháp phát hiện lỗi rất hiệu quả nếu bộ chia được chọn theo các luật vừa nêu do:

a. CRC có thể phát hiện tất cả các nhiễu bệt ảnh hưởng lên các bit có thứ tự lẻ.
b. CRC có thể phát hiện các nhiễu bệt có độ dài bé hơn hay bằng bậc của đa thức.
c. CRC có thể phát hiện với xác suất cao các nhiễu bệt có độ dài lớn hơn bậc của đa thức.

Ví dụ 5:

CRC – 12 ($x^{12}+x^{11}+x^3+x+1$) có bậc 12, có thể phát hiện tất cả các nhiễu bệt ảnh hưởng lên các bit lẻ, và cũng có thể phát hiện tất cả các nhiễu bệt có độ dài lớn hơn hay bằng 12, và phát hiện đến 99,97% các nhiễu bệt có độ dài lớn hơn 12 hay dài hơn nữa.

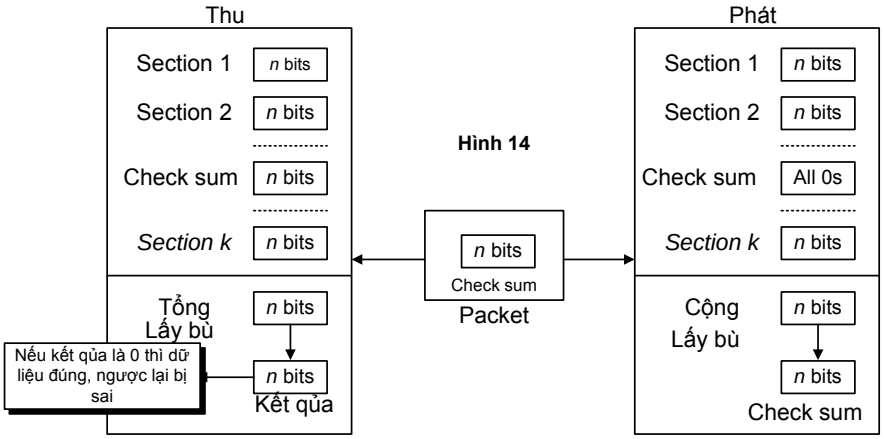
9.6 CHECKSUM

Phương pháp phát hiện lỗi ở lớp cao hơn và giống như các phương pháp VRC, LRC, và CRC thì phương pháp này cũng dựa trên yếu tố thừa (redundancy).

9.6.1 Bộ tạo Checksum:

Bên phát thực hiện các bước như sau:

- Bộ tạo checksum sẽ chia các đơn vị dữ liệu thành k phần, mỗi phần n bit (thường là 8, 16).
- Các phân đoạn này được cộng lại.
- Lấy bù 1 của kết quả cộng. Giá trị này được gắn vào đuôi của dữ liệu gốc và được gọi là trường checksum.(Phép bù 1: 0→1; 1→0)
- Chhecksum được truyền cùng với dữ liệu.



Ví dụ 6: Cho một khối dữ liệu có 16 bit: **10101001 00111001**. Mã hoá lỗi chuỗi dữ liệu trên dùng phương pháp checksum 8 bit. Tìm checksum và chuỗi dữ liệu phát.

Giải: Chia dữ liệu thành 2 phần, mỗi phần 8 bit

```
                    10101001
                +   00111001
                ------------
Tổng            =   11100010
Lấy bù 1        =   00011101

Chuỗi dữ liệu phát
<-- 10101001  00111001  00011101
                        Checksum
```

9.6.2 Bộ kiểm tra Checksum:

Máy thu thực hiện các bước như sau:

- Bộ kiểm tra checksum sẽ chia các đơn vị dữ liệu thành k phần mỗi phần n bit (giống như bên phát).
- Cộng các phần trên, được tổng (Sum).
- Lấy bù 1 của tổng.
- ả ếu kết qủa lấy bù là zêrô thì dữ liệu thu không bị sai, ngược lại dữ liệu bị sai.

Ví dụ 7: Giả sử máy thu nhận được chuỗi bit được mã hoá lỗi dạng checksum. Dữ liệu này đúng hay sai?

```
<-- 10101001  00111001  00011101
                        Checksum
```

Giải: Chia dữ liệu thành 3 phần, mỗi phần 8 bit

```
                    10101001
                +   00111001
                    00011101
                ------------
Tổng            =   11111111
Bù 1            =   00000000        Dữ liệu thu không bị sai
```

Ví dụ 8: Giả sử máy thu nhận được chuỗi bit được mã hoá lỗi dạng checksum. Dữ liệu này đúng hay sai?

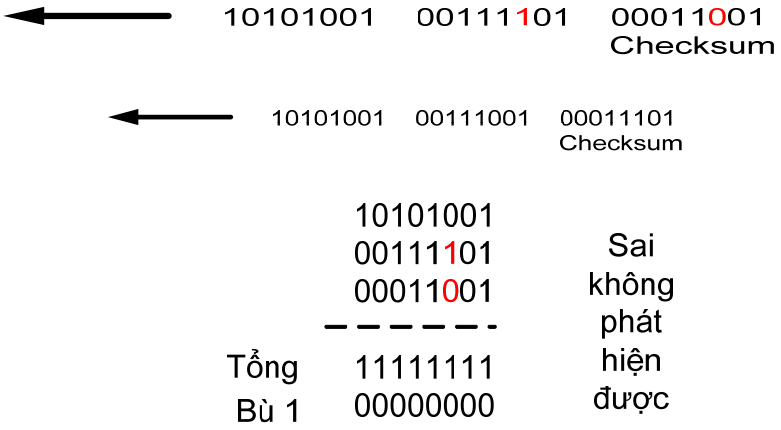
10101111 11111001 00011101

Giải: Chia dữ liệu thành 3 phần, mỗi phần 8 bit

```
                10101111
            +   11111001
                00011101
            ---------
Kết qủa     1   11000101
nhớ             ------1
tổng            11000110
Bù 1            00111001        Dữ liệu thu bị sai
```

Bù 1 của tổng khác zêrô nên dữ liệu thu bị sai

Ví dụ 9: sai 2 bit 0, 1 của 2 phân đoạn có vị trí giống nhau.



Hiệu năng:

Checksum phát hiện được tất cả các lỗi bit lẻ cùng như hầu hết các bit chẵn. Tuy nhiên, nếu một hay nhiều bit trong phân đoạn bị hỏng và bit tương ứng hay bit có giá trị đảo trong phân đoạn thứ hai cũng bị lỗi, thì khi lấy tổng, không nhận ra thay đổi và máy thu không phát hiện lỗi được. ả ếu bit cuối trong một phân đoạn là 0 và bi đổi thành 1 khi truyền, thì ta không thể phát hiện ra lỗi nếu bit 1 cuối của phân đoạn thứ hai cũng chuyển thành 0.

9.7 SỬA LỖI

Có hai cách sửa lỗi là:

- Khi phát hiện một lỗi, máy thu phải yêu cầu máy phát truyền lại dữ liệu.
- Máy thu dùng các mã sửa lỗi, để sửa tự động một số lỗi.

Các mã sửa lỗi, thường rất phức tạp hơn so với mã phát hiện lỗi và cần nhiều bit dư. Số bit cần thiết để sửa lỗi nhiều bit thường rất lớn và không phải lúc nào cũng hiệu quả. Thông thường hầu hết các phương pháp sửa lỗi đều giới hạn ở một, hai hoặc ba bit.

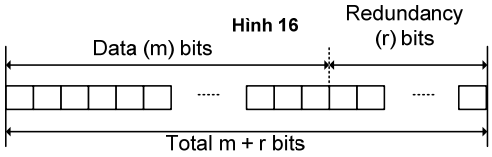
Trong tài liệu này chỉ đề cập đến phương pháp phát hiện sai 1 bit (xác định vị trí sai) và sửa sai. Do vậy để sử sai một bit, ta phải biết được bit nào bị sai. ả hư thế, ta phải định vị được bit sai này.

Ví dụ khi cần sửa lỗi một bit trong bảng mã ASCII, mã sửa lỗi phải xác định bit nào bị thay đổi trong 7 bit. Trường hợp này, cần phân biệt được giữa 8 trạng thái khác nhau: không lỗi, lỗi ở vị trí 1, lỗi ở vị trí 2, và tiếp tục cho đến vị trí 7. ả hư thế cần thiết phải có đủ số bit dư để biểu diễn được 8 trạng thái này.

Đầu tiên, ta nhận thấy là với 3 bit là đủ do có thể biểu diễn được tám trạng thái (từ 000 đến 111) và như thế thì có thể chỉ ra được tám khả năng khác nhau. Tuy nhiên, việc gì xảy ra nếu lỗi lại rơi vào các bit dư này? Bảy bit trong ký tự ASCII cộng với 3 bit dư sẽ tạo ra 10 bit. Với ba bit là đủ, tuy nhiên cần có thêm các bit phụ cho tất cả các tình huống có thể xảy ra.

9.7.1 Các bit dư

Để tính số bit dư (r) cần có để có thể sửa lỗi một số bit dữ liệu (m), ta cần tìm ra quan hệ giữa m và r. Trong hình sau cho thấy m bit dữ liệu và r bit dư. Độ dài của mã có được là $m+r$.



ả ếu tổng số các bit trong một đơn vị được truyền đi là $m+r$, thì r phải có khả năng chỉ ra ít nhất $m+r+1$ trạng thái khác nhau. Trong đó, một trạng thái là không có lỗi và $m+r$ trạng thái chỉ thị vị trí của lỗi trong mỗi vị trí $m+r$.

Điều đó, tức là $m+r+1$ trạng thái phải được r bit phát hiện ra được; và r bit có chể chỉ được 2^r trạng thái khác nhau. ả hư thế, 2^r phải lớn hơn hay bằng $m+r+1$:

$$2^r \geq m+r+1.$$

Giá trị của r có thể được xác định từ cách gắn vào trong giá trị của m (chiều dài ban đầu của đơn vị dữ liệu cần gởi đi).

Thí dụ, nếu giá trị của m là 7 (trường hợp 7 bit của mã ASCII), thì giá trị bé nhất của r cần thỏa mãn phương trình là 4:

$$2^r \geq 7+r+1 \text{ ; chọn } r=4$$

$$2^4 \geq 7+4+1.$$

Bảng B.1 cho thấy một số khả năng của các giá trị m và r tương ứng.

Số lượng bit dữ liệu (m)	Số lượng bit dư (r)	Tổng số bit ($m+r$)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

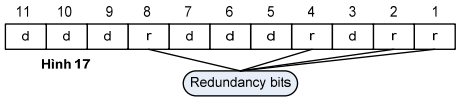
Mã Hamming

Ta đã xem xét số lượng bit cần thiết để phủ toàn bộ trạng thái bit lỗi có thể có khi truyền. ả hưng điều còn lại là phải xử lý như thế nào để biết được trạn thái đang xuất hiện? R.W.Hamming cung cấp một giải pháp thực tiễn.

Định vị của các bit dư

Mã Hamming có thể được áp dụng vào đơn vị dữ liệu có chiều dài bất kỳ dùng quan hệ giữa dữ liệu và các bit dư đã được khảo sát trước đây.

Thí dụ, mã 7 bit ASCII cần có 4 bit dư được thêm vào phần cuối đơn vị dữ liệu hay phân bố vào bên trong các bit gốc. Các bit này được đặt ở các vị trí 1, 2, 4 ,8,…. (2^n). Ta gọi các bit này lần lượt là r_1, r_2, r_4 và r_8.



Hình 17

Trong mã Hamming, mỗi bit ***r* là bit VRC của một tổ hợp các bit dữ liệu**; r_1 là bit VRC của một tổ hợp bit; r_2 là một bit trong một tổ hợp bit khác và cứ thế tiếp tục. Tổ hợp được dùng để tính toán mỗi giá trị trong bốn bit *r* này trong chuỗi bảy bit được tính toán như sau:

r_1 (bit 1), 3, 5, 7, 9, 11 ; tổng số bit 1 là một số chẵn

r2 (bit 2), 3, 6, 7, 10, 11 ; tổng số bit 1 là một số chẵn

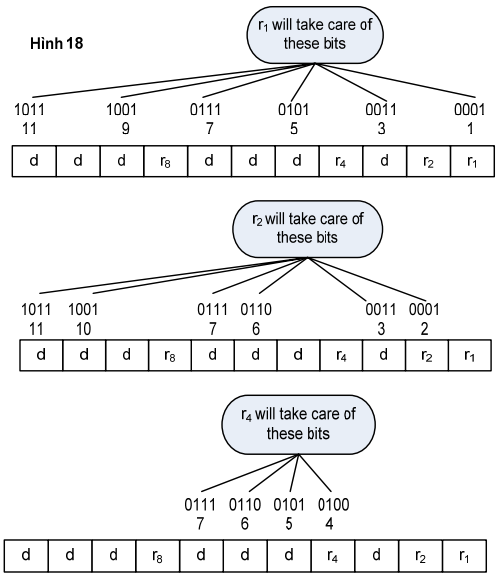
r4 (bit 4), 5, 6, 7 ; tổng số bit 1 là một số chẵn

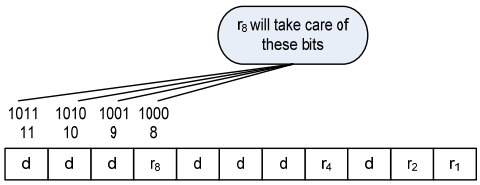
r8 (bit 8), 9, 10, 11 ; tổng số bit 1 là một số chẵn

Mỗi bit dữ liệu có thể tính đến trong nhiều hơn một lần tính VRC. Thí dụ, trong chuỗi trên, mội bit dữ liệu gốc được tính đến trong ít nhất hai tập, trong khi *r* chỉ được tính một lần.

Để tìm các mẫu trong chiến lược tính toán này, hảy xem cách biểu diễn của mỗi vit trí bit. Bit r_1 được tính dùng tất cả các vị trí bit có cách biểu diễn nhị phân có 1 trong vị trí tận cùng bên phải. Bit r_2 được tính dùng tất cả các vị trí bit có cách biểu diễn nhị phân có 1 trong vị trí thứ hai bên phải và tiếp tục như vẽ trong hình 18.

Hình 18





Hình 9.6

9.7.2 Các bit dư

Ví dụ: Cho một dữ liệu 1001101, tìm chuỗi dữ liệu được mã hoá dạng Hamming.

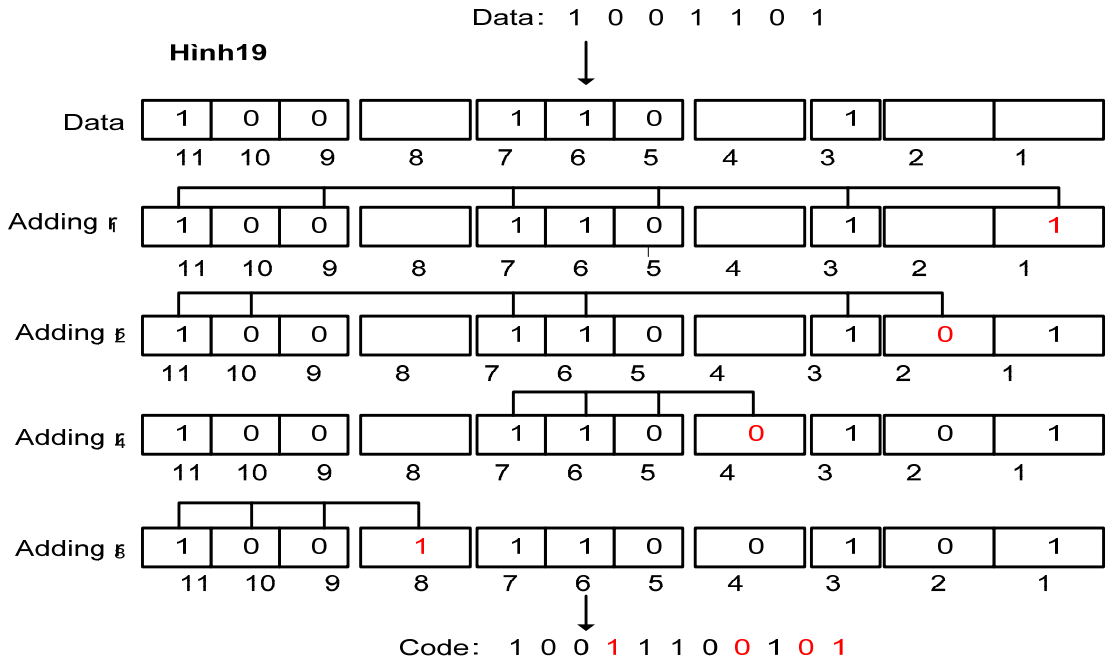
Giải:

- Xác định số bit dư: số bit của dữ liệu là m=7;

  Suy ra số bit dư r theo bất đẳng thức: $2^r \geq m+r+1$

  $m= 7 \rightarrow 2^r \geq 7+r+1$; chọn $r=4$

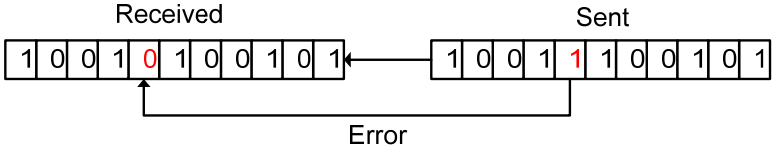
- **Tính toán các giá trị r:**



Hình 9.7

Bước đầu tiên, ta đặt mỗi bit của ký tự gốc vào vị trí thích hợp trong đơn vị 11 bit. Trong bước kế tiếp, ta tính các parity chẵn với nhiều tổ hợp bit khác nhau. Giá trị parity của mỗi tổ hợp là giá trị bit *r* tương ứng.Thí dụ, giá trị của r_1 được tính để cung cấp parity chẵn cho tổ hợp các bit 3, 5, 7, 9 và 11. Giá trị của r_2 được tính để cung cấp parity bit cho các bit 3, 6, 7, 10 và 11, và cứ thế tiếp tục. Mã 11 bit sau cùng được gởi đi qua đường truyền.

9.7.3 Phát hiện và sửa lỗi



Giả sử trong lúc truyền tín hiệu đi, bit thứ 7 đã thay đổi từ 1 → 0.

Máy thu nhận và tính lại bốn số dư r ở bên thu (VRC):

r_1 bên thu, 1, 3, 5, 7, 9, 11 ; tổng số bit 1 là một số chẵn

$r2$ bên thu, 2, 3, 6, 7, 10, 11 ; tổng số bit 1 là một số chẵn

$r4$ bên thu, 4, 5, 6, 7 ; tổng số bit 1 là một số chẵn

$r8$ bên thu, 8, 9, 10, 11 ; tổng số bit 1 là một số chẵn

Vị trí bit sai của dữ liệu thu là giá trị thập phân của số nhị phân $r_8\ r_4\ r_2\ r_1$.

Ví dụ: Giả sử máy thu nhận được một dữ liệu 10011100101 đã được mã hoá dưới dạng Hamming. Hãy cho biết chuỗi dữ liệu nhận được đúng hay sai.

11	10	9	8	7	6	5	4	3	2	1	Vị trí
1	0	0	1	1	1	0	0	1	0	1	

r_1 bên thu, 1, 1, 0, 1, 0, 1 ; tổng số bit 1 là một số chẵn → r_1 =0

$r2$ bên thu, 0, 1, 1, 1, 0, 1 ; tổng số bit 1 là một số chẵn → $r2$ =0

$r4$ bên thu, 0, 0, 1, 1 ; tổng số bit 1 là một số chẵn → r_4 =0

$r8$ bên thu, 1, 0, 0, 1 ; tổng số bit 1 là một số chẵn → r_8 =0

$r_8\ r_4\ r_2\ r_1$ =0000_2= 0_{10}, *Không có bit sai*

Ví dụ: Giả sử máy thu nhận được một dữ liệu 10010100101 đã được mã hoá dưới dạng Hamming. Hãy cho biết chuỗi dữ liệu nhận được đúng hay sai.

11	10	9	8	7	6	5	4	3	2	1	Vị trí
1	0	0	1	0	1	0	0	1	0	1	

r_1 bên thu, 1, 1, 0, 0, 0, 1 ; tổng số bit 1 là một số chẵn → r_1 =1

$r2$ bên thu, 0, 1, 1, 0, 0, 1 ; tổng số bit 1 là một số chẵn → $r2$ =1

$r4$ bên thu, 0, 0, 1, 0 ; tổng số bit 1 là một số chẵn → r_4 =1

$r8$ bên thu, 1, 0, 0, 1 ; tổng số bit 1 là một số chẵn → r_8 =0

Vậy vị trí sai là giá trị thập phân của số nhị phân $r_8\ r_4\ r_2\ r_1$ bên thu, $r_8\ r_4\ r_2\ r_1$ =0111_2= 7_{10}, *Vậy vị trí sai là 7, sửa bit ở vị trí 7: '0' → '1'*

TÓM TẮT

- Lỗi truyền dẫn thường được phát hiện tại lớp vật lý trong mô hình OSI
- Lỗi truyền dẫn thường được sửa trong lớp kết nối dữ liệu trong mô hình OSI
- Lỗi có thể được chia ra thành:
    a. Lỗi một bit: chỉ sai một bit trong đơn vị dữ liệu
    b. Bệt: sai hai hay nhiều bit trong đơn vị dữ liệu
- Redundancy là ý niệm nhằm gởi thêm các bit dư dùng trong phát hiện lỗi
- Có bốn phương pháp kiểm tra lỗi thông thường là:
    a. VRC (vertical redundancy check)
    b. LRC (longitudinal redundancy check)
    c. CRC (cylic redundancy check)
    d. Checksum
- Trong VRC, một parity bit được thêm vào đơn vị dữ liệu
- VRC chỉ có thể phát hiện một bit và các bit lẻ bị lỗi; không thể phát hiện số bit chẵn.
- Trong LRC, có một dữ liệu thừa theo sau một đơn vị dữ liệu n bit
- CRC, phương pháp mạnh nhất trong phương pháp kiểm tra lỗi dùng bit dư, có cơ sở là phép chia nhị phân
- Checksum được dùng trong giao thức cấp cao hơn (TCP/IP) để phát hiện lỗi

Để tính checksum, thì cần:

a. Chia dữ liệu thành nhiều phần nhỏ
b. Cộng các phần này lại dùng phương pháp bù một
c. Lấy bù của tổng cuối cùng, đây chính là checksum

Tại máy thu, khi dùng phương pháp checksum, dữ liệu và checksum phải được cộng lại thành giá trị 0 khi không có lỗi

- Mã Hamming là phương pháp sửa lỗi một bit dùng các bit thừa. Số bit là hàm của độ dài đơn vị dữ liệu
- Trong mã Hamming, một đơn vị dữ liệu m bit thì dùng công thức $2^r \geq m + r + 1$ để xác định r, số bit dư cần có.