CRC를 利用한 情報貯藏과 情報檢査

金　聖　玉
(한남대학　전산과)

〈 차　　례 〉

Ⅰ. 序　　論

現代의 대부분의 데이터들은 컴퓨터의 記憶裝置 또는 補助記憶裝置에 저장되어 있으며, 필요한 경우 우리는 이들 데이터들을 情報로써 利用하게 된다. 그러나 우리가 주로 사용하고 있는 계수형 컴퓨터(digital computer)에서는 모든 情報가 2진 형태(binary form)로 저장되기 때문에 자칫 심각한 問題가 발생할 수도 있다. 예를 들어 0000 0000 0000 0001로 저장된 데이터가 0100 0000 0000 0001로 단 한 비트(bit) 변했을 경우, 원래의 데이터는 1인데 반해 변한 데이터는 16384라는 값으로 解釋되게 된다. 더우기 저장되어 있는데이터의 각 비트가 어떤 意味를 가지고 있을 경우에는 正反對의 結果를 超來하는 아주 심각한 地境에 이를 수도 있다. 따라서 저장되어 있는 情報가 안전하게 維持되어야 하는 것은 말할 것도 없거니와 만일 데이터의 어떤 비트가 변했을 경우에는 그

事實을 미리 알 수 있어야 하겠다.

여기서는 저장된 情報의 異狀 有無를 미리 檢査하는 한가지 方法으로 데이터 交信(data communication)에 사용되는 CRC(Cyclic Redundancy Check)를 利用해 보고자 하는 것이다.

Ⅱ. CRC(Cyclic Redundancy Check)

비트의 수가 k 개인 2진 정보(binary information)는 x에 대한 $(k-1)$차의 多項式으로 표시된다. 즉 2진 정보가 $a_0, a_1, a_2, \cdots\cdots a_k$ 일 때

$$M(x) = a_0 + a_1 x^1 + a_2 x^2 + \cdots\cdots + a_{k-1} x^{k-1}$$

로 표시된다. 예를 들어 110101이라는 2진 정보는 多項式 $M(x) = 1 + x + x^3 + x^5$ 으로 표시된다. 이들 데이터 블럭(block) $a_0, a_1, \cdots\cdots, a_k$가 送信할 데이터 일 때 M(x)를 메시지(Message)多項式이라고 하며 a_i는 1또는 0이다. 이들 다항식들의 演算은 일반 代數法則에 따르지만 덧셈은 modulo - 2를 이용한다.

즉,

$$\begin{array}{rl} & 1 + x + x^3 \\ \times) & 1 + x \\ \hline & 1 + x + x^3 \\ & \quad x + x^2 + x^4 \\ \hline & 1 + x^2 + x^3 + x^4 \end{array}$$

실제 데이터 交信에서는 원래 전송하고자 하는 데이터에 검사용 비트들을 붙여서 電送하게 되는데 이렇게 만들어진 전체 비트의 수가 n이고, 원래 보내고자 하는 비트가 k개 일 때 $n-k$차인 다항식 G(x)를 생각해 보자. 이 G(x)는 x^{n-k}와 x^0를 포함하는 것으로 한다. 메시지 다항식 M(x)에 G(x)의 최고차 항 x^{n-k}를 곱하여 G(x)로 나눈 후 몫과 나머지를 각각 Q(x), R(x)라고 하면,

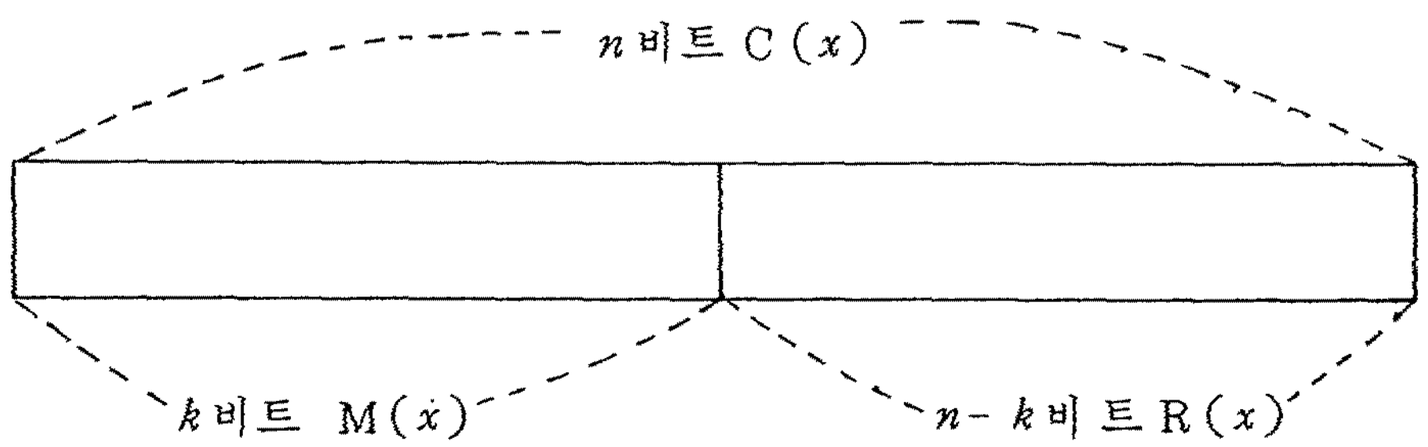
$$x^{n-k} \cdot M(x) = Q(x) \cdot G(x) + R(x) \cdots\cdots\cdots\cdots (1)$$

이 된다. (1)식의 兩邊에 Modulo-2 계산으로 $R(x)$를 더하면

$$C(x) = x^{n-k} \cdot M(x) + R(x) = Q(x) \cdot G(x) \cdots\cdots\cdots\cdots (2)$$

이다. 이 식에서 $C(x)$를, 메시지 k 디지트(digit)를 제너레이터(generator) 多項式 $G(x)$로 엔코우드(encode)한 코우드 다항식이라고 한다. 또한 $R(x)$에 해당되는 비트를 CRC 비트라고 한다. 여기서 식(2)의 $C(x) = x^{n-k}M(x) + R(x)$를 살펴보면 〈圖 1〉과 같이 $C(x)$로 표시되어 電送되는 n개의 비트들은 $M(x)$로 표시되는 데이터 블럭에 $R(x)$를 餘分의 비트(CRC 비트)로 붙여서 만들어진 것임을 알 수 있다. 또한 식 (2)의 후반 $C(x) = Q(x) \cdot G(x)$에서 $C(x)$는 $G(x)$로 나누어 떨어짐도 알 수 있다.

〈図 1〉



예를 들어 $n = 15$, $k = 10$, $n - k = 5$인 경우를 보자.

$M(x) = 1 + x^2 + x^5 + x^9$: 1010010001

$G(x) = 1 + x^2 + x^4 + x^5$: 101011

이라고 하면 식 (1)을 이용하여

$$x^5 + x^7 + x^{10} + x^{14} = (1 + x^2 + x^4 + x^5)(1 + x + x^2 + x^3 + x^7 + x^8 + x^9) = (1 + x)$$

이다 따라서

$$C(x) = (1 + x) + (x^5 + x^7 + x^9 + x^{14})$$

$$\underbrace{11000}_{\text{검사비트}} \quad \underbrace{1010010001}_{\text{메시지 비트}}$$

즉 실제 電送되는 비트들은 1100010100100001이며 이 중 뒤의 10비트가 원래

보내고자 하는 비트이고 나머지 5비트가 檢査에 필요한 CRC 비트 이다.

Ⅲ. 에러(error)의 検出

데이터 블럭이 電送되는 동안 電線에서는 雜音에 의하여 에러 비트가 생길 수 가 있고 이들 에러 비트를 나타내는 다항식을 에러 다항식 $E(x)$로 표시하기로 하자. 受信機 쪽에서 받는 데이터에 對應하는 다항식 $T(x)$는 다음과 같이 표시된다.

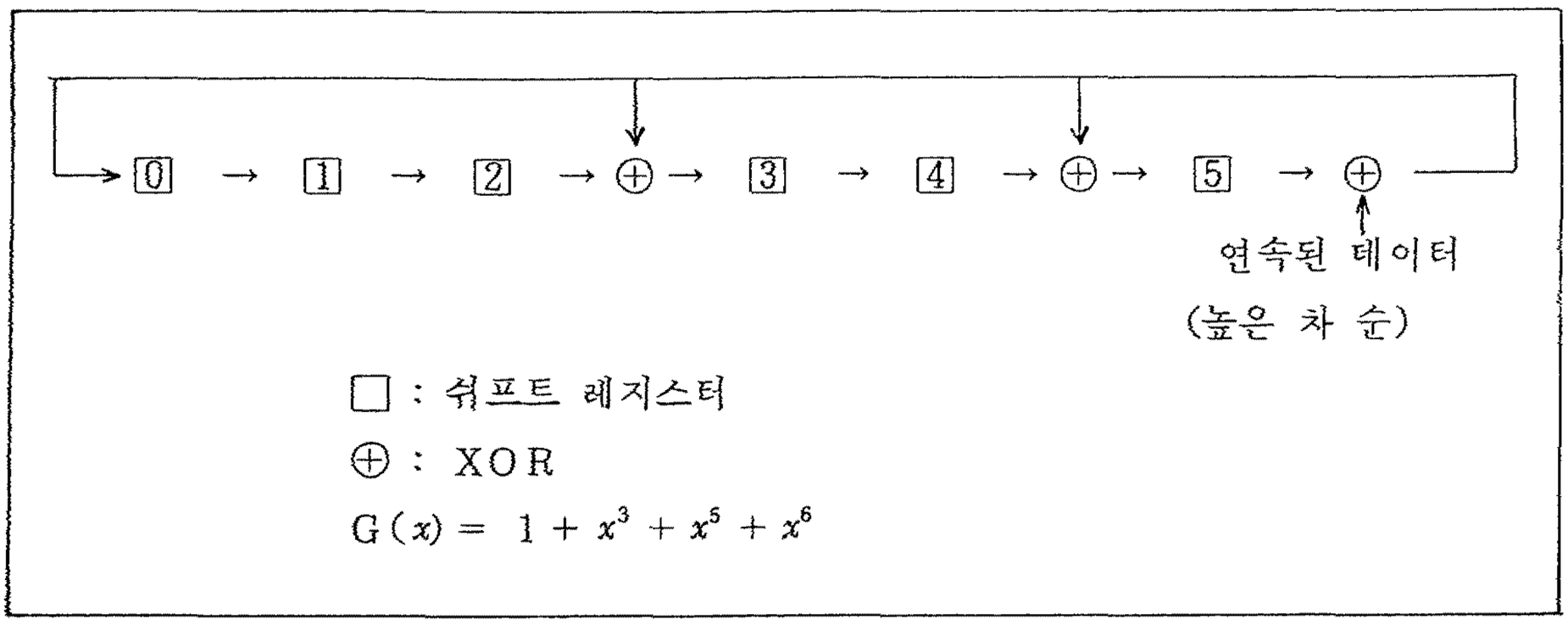
$$T(x) = C(x) + E(x) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (3)$$

이 때 $T(x)$가 $G(x)$로 나누어 떨어지지 않는 경우 즉 $E(x) \neq 0$인 경우에는 에러가 발생했음을 알 수 있다. 그러나 $T(x)$가 $G(x)$로 나누어 떨어지는 경우에는 에러가 없다고 생각할 수도 있고 또 $E(x)$조차도 $G(x)$로 나누어 떨어져서 檢出되지 않는 에러가 발생한 것이라고 생각할 수도 있다. 따라서 어떤 에러의 形態인 $E(x)$도 $G(x)$로 나누어지지 않도록 $G(x)$를 決定할 必要가 있다. 그러나 完全한 $G(x)$는 찾을 수 없고 에러 檢査率을 極大化하는 $G(x)$를 사용할 수 밖에 없다(이런 $G(x)$에 대해서는 研究된 바가 많으므로 後尾에 列擧한 參考文獻을 參考하기 바란다).

이제 실제로 에러를 찾는 方法을 살펴보기로 한다. 실제 $x^{n-k} \cdot M(x) / G(x)$를 計算하기 위하여 다항식들의 곱셈과 나눗셈을 그대로 遂行하는 것은 극히 복잡한 일이다. 그러나 실제는 쉬프트 레지스터(shift register)를 사용하면 간단히 解決할 수 있다. 예를 들어 $M(x) = 1 + x^5 + x^8 + x^{10}$(10000100101)이고 $G(x) = 1 + x^3 + x^5 + x^6$ 일 때 〈圖 2〉와 같은 쉬프트 레지스터를 사용하면 간단하다.

이 때 쉬프트 레지스터의 내용과 電送되는 비트는 다음과 같다.

〈 図 2 〉



	쉬프트 레지스터	전송된 비트
초 기 상 태 ⟶	0 0 0 0 0 0	
	1 0 0 1 0 1	1
	1 1 0 1 1 1	0
	0 1 1 0 1 1	1
	1 0 1 0 0 0	0
	0 1 0 1 0 0	0
	1 0 1 1 1 1	1
	1 1 0 0 1 0	0
	0 1 1 0 0 1	0
	1 0 1 0 0 1	0
	1 1 0 0 0 1	0
송신기에서 보내는 ⟶	0 1 1 0 0 0	1
나머지	0 0 1 1 0 0	0
	0 0 0 1 1 0	0
	0 0 0 0 1 1	0
	0 0 0 0 0 1	1
	0 0 0 0 0 0	1
수신기쪽의 최종 ⟶	0 0 0 0 0 0	0
상태		

M(x)

C(x)

CRC

Ⅳ. 情報의 貯藏과 檢査

記憶裝置에 貯藏되어 있는 데이터를 사용하기 전에 원래대로 잘 維持되어 있는지 調査하기 위해서는 데이터를 저장할 때 부터 檢査할 수 있는 餘分의 데이터를 함께 저장해야 된다. 이 때는 저장하고자 하는 데이터에 대한 CRC 비트들을 計算하여 주어진 데이터와 CRC 데이터를 함께 저장하면 된다. 이들 데이터를 檢査할 때는 주어진 데이터와 CRC 비트들을 합해서 $T(x)$ 데이터로 看做하고 CRC 검사를 하면 될 것이다. 이 때 最終的으로 남는 쉬프트 레지스터의 값이 0이면 저장된 情報는 安全하게 유지된 것이고 그 값이 0이 아닐 경우에는 에러가 생긴 것이다. 이를 위해서는 미리, 檢査하고자 하는 데이터의 構成, CRC 비트의 수, 에러 檢出率, 그리고 제너레이터 다항식을 決定하는 것이 중요하다. 이 때 CRC 비트의 수는 $G(x)$의 차수와 같고, 또 대부분의 컴퓨터가 8비트 바이트(byte) 또는 16비트 워드(word) 單位로 데이터를 處理하기 때문에 $G(x)$의 차수는 16의 倍數로 택하는 것이 바람직하다.

이 경우에는 거의 완전한 에러 檢出率을 保障하기 위하여 $G(x)=(1+x^{23})(1+x^2+x^9)$을 택하기로 한다. 이 다항식의 차수는 32이므로 CRC 비트는 32비트가 될 것이다. 이 경우에 n의 최대값은 11753이므로 k의 최대값은 11721이 된다. 즉 732워드의 데이터에 대해서 두 개의 CRC 워드가 필요하게 된다. 한 데이터 블럭의 일부분에 에러가 생겼을 경우, 첫번째 에러 비트와 마지막 에러 비트를 包含하여 b 개의 비트를 가진 에러 블락이 생겼을 때 b 와, 에러 檢出에 失敗할 確率 p 와의 관계는;

$$b < n+k-1 \qquad p = 0$$
$$b = n+k-1 \qquad p = 2^{-(n-k-1)}$$
$$b > n+k-1 \qquad p = 2^{-(n-k)}$$

이므로 위와 같은 $G(x)$를 사용하면 에러를 檢出할 수 있는 確率은 $1-(\frac{1}{2})^{31} = 9.9999999$ 이상이므로 거의 완전한 에러 檢出能力을 갖는다고 말 할 수 있다.

만일 데이터 交信(data communication)에 관한 問題라면 위와 같은 $G(x)$를 사용해서 CRC 비트들을 만들어 내고 또 檢査하는데 〈圖 3〉과 같은 피이드 백 쉬프트 레지스터로 된 간단한 하드웨어 裝置를 附着시키면 되겠지만 우

리는 汎用 컴퓨터의 메모리에 情報를 貯藏하고 또 檢査하는데 위의 方法을 사용해야 하므로 순수 소프트웨어 方法으로 處理해야 된다. 여기서는 方法만을 提示하기 위하여 732 워드 이내의 情報가 連續된 컴퓨터의 메모리에 저장되는 경우만을 생각하기로 하면, 情報를 貯藏시킬 때, 그리고 情報를 檢査할 때의 알고리즘(Algorithm)은 아래와 같다.

A. 데이터 貯藏 알고리즘

A - 1. 초기 CRC 워드 두 개를 0으로 둔다.

A - 2. 더 받아들여서 저장시킬 데이터가 없을 경우에는 CRC 워드를 指定된 場所, 예를 들어 데이터에 配定된 記憶場所 다음에 저장시키고 A과정을 끝낸다. 그리고 받아 들일 데이터가 있을 경우 A - 3으로 간다.

A - 3. 16비트 데이터를 받아 들여 저장한다.

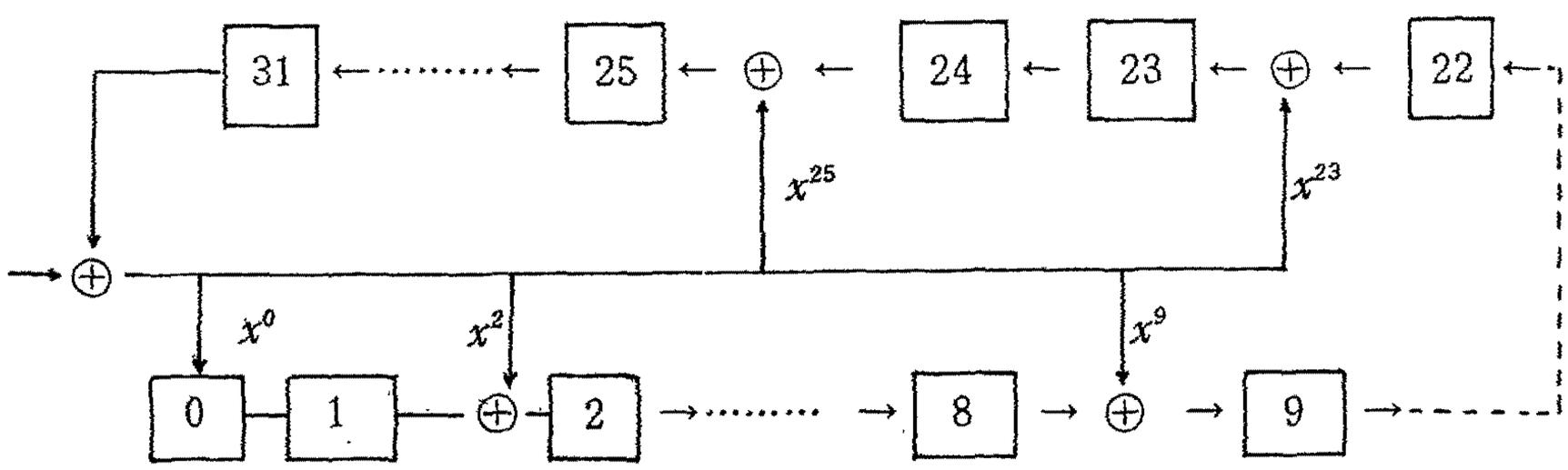
A - 4. 카운터를 16으로 정한다.

A - 5. 두 워드로 된 CRC 비트들을 왼쪽으로 한자리 환상 쉬프트 시킨다.

A - 6. 데이터 비트 15(가장 높은 차의 비트)와 CRC 비트 0(가장 낮은 차의 비트)와 XOR을 시킨다.

A - 7. 결과가 1인 경우 CRC 비트들과 $G(x)$에 해당하는 두 워드의 비트들과 XOR을 시켜서 나온 結果를 새로운 CRC 비트들로 한다. 그리고 A - 6의 結果가 0인 경우에는 바로 다음 過程으로 넘어간다.

〈図 3〉



$$G(x) = 1 + x^2 + x^9 + x^{23} + x^{25} + x^{32}$$

A - 8. 데이터 비트들을 한자리 左側으로 쉬프트 시킨다.

A - 9. 카운터의 內容을 1 減少시킨다.

A - 10. 카운터의 內容이 0일 경우에는 A - 2 過程으로 돌아가고 아닌 경우 A - 5 과정으로 돌아간다.

B. 데이터 檢査 알고리즘

B - 1. 초기 CRC 워드 두 개를 0으로 둔다.

B - 2. 저장된 데이터가 끝났으면 B - 5 과정으로 간다. 아닌 경우에는 貯藏된 한 워드의 데이터를 로우드(load)한다.

B - 3. 過程 A - 4부터 A - 9 過程까지를 遂行한다.

B - 4. 카운터의 內容이 0인 경우 B - 2 과정으로 돌아가고 아닌 경우 B - 3 과정으로 돌아간다.

B - 5. 이미 저장된 CRC 첫 워드를 로우드 한다.

B - 6. 과정 A - 4부터 A - 9 과정까지를 수행한다.

B - 7. 카운터의 내용이 0인 경우 다음 과정으로 가고 아닌 경우 B - 6과정으로 돌아간다.

B - 8. 저장된 CRC의 둘째 워드를 로우드 한다.

B - 9. 과정 A - 4부터 A - 9 과정까지를 수행한다.

B - 10. 카운터의 내용이 0인 경우 다음 과정으로 가고, 아닌 경우에는 B - 9 과정으로 돌아간다.

B - 11. 최종 CRC가 0인 경우 에러가 없는 것으로 간주하고, 0이 아닌 경우에는 에러가 있는 것으로 判斷하고 B 과정을 끝낸다.

V. 結　　論

이 方法은 위의 알고리즘에서 보듯이 특수한 하드웨어의 補助없이 어느 컴퓨터에서나 간단한 프로그램으로 손쉽게 適用시킬 수 있다. 또한 저장시켜야 할 데이터가 많을 경우에는 위의 알고리즘을 약간 修正하여 732 워드 씩으로 나누어 각각의 데이터 블럭에 대해서 두 워드 씩의 CRC 비트를 계산하면 될 것이다. 또한 위의 알고리즘은 補助記憶裝置에 情報를 저장하는 경우에도 이용할 수

가 있겠다.

이런 알고리즘을 사용할 경우 732 워드의 데이터 마다 두개씩의 추가 메모리가 요구되지만 저장된 중요 정보를 미리 點檢하는 데 이 정도의 출현은 무시할 수 있겠다. 또한 所要時間面에서 살펴 볼 때 732 워드의 情報를 點檢하는데 소형 컴퓨터(Eclipse S- 200 기준)에서의 테스트 結果 약 0.2秒의 時間이 所要되므로 實行時間 역시 무시해도 된다.

또한 데이터가 저장된 장비 내에서 바뀌는 일은 흔하지 않다고 해도 情報가正反對의 內容으로 作用하여 엄청난 結果를 超來할 수 있는 可能性을 완전히 排除하지 못한다는 點을 勘案한다면, 그리고 또 補助記憶裝置나 ROM에 長期間 저장되어 있는 경우에는 이런 방법으로 저장된 정보를 사용하기 전에 미리 檢査하는 일은 중요하고도 필요한 일이 될 것이다.

〈參 考 文 獻〉

1. Rucky, R. W. J. SAIZ ;
   Principles of Data Communication, 1968.
2. Liccardo, Michael A.,"Polynomial error detecting codes and their implementation", *Computer Design*(Sept. 1971), pp.53- 59.
3. Swanson, Robert, "Understanding cyclic redundancy codes", *Computer Design*, (Nov. 1975), pp.93 - 99.