목차

문제 1. n 의 보수에 의한 감산에서 올림수를 어떻게 처리하는지 설명하고, 10 진수로 피감수가 13 감수가 10 인 감산을 10 의 보수에 의한 방법으로 처리하시오.(교재 200~203 참조)

먼저 피감수에 감수의 n 의 보수를 더한 다음

올림수가 있는 경우는 올림수를 버린다. (올림수가 n 개일 때 n 개 모두)

올림수가 없는 경우는 결과값의 n 의 보수를 구하고 –를 붙인다.

10 의 10 의 보수를 구하면 89+1=90

13+90=103 인데 올림수가 있으므로 올림수를 버리면

103 에서 1 을 버려서 3(03)이 계산한 값이 된다.

문제 2. 2 진수 1011 을 그레이 코드로 표시하시오.

그레이 코드는 2 진수의 MSB 를 그대로 가져온 뒤 그 다음 부터는 2 진수의 해당 자리 바로 왼쪽의 수와 해당 자리의 수 간의 XOR 연산을 하여 구할 수 있으므로

1011(2) → 1???(Gray)

→ 1⊕0=1, 11??(Gray)

→ 0⊕1=1, 111?(Gray)

→ 1⊕1=0, 1110(Gray)

문제 3. 6 비트 BCD 코드를 설명하고 6 비트 BCD 코드에서 알파벳 소문자가 없는 이유를 설명하시오.

6 비트 BCD 코드

총 64 가지 코드에 문자나 기호를 배정한 코드 방식을 6 단위 코드 체제인 표준 2 진화 10 진 코드(Standard Binary Coded Decimal Interchange Code)라고 한다.

6 비트 BCD 코드는 2 비트의 Zone 비트(그룹 분류)와 4 비트의 Digit 비트로 구성된다.

알파벳 소문자가 없는 이유

1. Zone 비트 측면

6 비트 BCD 코드의 알파벳 배열 방식을 보면 Zone 비트(2 비트, 4 가지 경우)만을 영문자 3 가지, 숫자 1 가지로 할당하여

8421 BCD 코드의 1~9 까지에 알파벳 9 가지를 할당하는데 이러한 알파벳 배열 방식을 따를 때 2 비트의 Zone 비트로는 부족하다. (이후 EBCDIC 코드에서는 Zone 비트에 소문자와 대문자/숫자 구역이 각각 할당되었음)

2. 할당 활용 효율성 측면

알파벳은 소문자 26 자 대문자 26 자 총 52 자로 6 비트로 모든 알파벳을 할당하자면 이론상으로는 가능하나 알파벳을 6 비트에 대소문자를 모두 할당시키면 0~9 의 아라비안 숫자 10 개를 추가할당한 상태에서 기타 특수 문자를 할당할 공간이 (64-52-10) 2 개 밖에 남지 않고 다른 중요한 특수 문자를 할당하는 것에 비해 활용성이 떨어진다.

문제 4. 해밍코드를 설명하고, 수신 비트가 111011001111 일 때 몇 번째 비트에서 에러가 발생했는지와 교정된 2 진 정보 데이터(D8~ DI)를 적으시오.(교재 218~220 참조)

해밍 코드(Hamming code)

해밍 코드는 에러 정정 코드 중 가장 간단한 형태로 에러를 검출하고 자동으로 정정까지 해주는 코드이다. 해밍 코드는 데이터 비트 사이에 패러티 비트를 추가하여 전체 데이터를 검증하고 정정한다.

해밍 코드를 적용된 데이터 정보에 필요한 패러티 비트의 개수를 구하려면 아래의 식에서 P 의 값을 구하면 된다.

$$2^P \geq D + P + 1$$

예를 들어 8 비트의 데이터 정보를 갖는 짝수 해밍 코드를 보면 2^P >= D+P+1 을 만족해야 하므로 D=0 비트이므로 P 는 최소한 4 비트가 되어야 한다.

2 진 데이터 10010010 의 사례를 보면 이 데이터의 패러티 비트는 계산식에 의해 4 가 산출되고, 따라서 패러티 비트가 추가되는 자리는 1, 2, 4, 8 이 된다.

패러티 비트를 포함하여 구성되는 데이터 비트는 총 12 자리가 될 것이다. 새롭게 구성된 데이터 비트는 아래 표와 같다.

12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	P4	0	0	1	P3	0	P2	P1

패러티 비트의 결정 과정은 다음과 같다.

1. 추가된 패러티 비트는 바로 다음 자리의 비트를 시작으로 전체 비트의 끝에 도달할 때까지 검사를 수행한다.

2. 이 때, 다음 2^k 개의 비트를 검사한 후, 2^k 자리를 건너뛰어 검사를 계속한다.

3. 검사 도중에 다른 추가된 패러티 비트를 만난다면 그 2^k 비트를 포함한 자리를 건너뛴 후 다시 ② 를 수행한다.

4. 비트의 끝에 도달했다면, 1 의 값을 가진 비트의 개수에 따라 그 패러티 비트 값을 결정한다.

짝수 해밍 코드의 경우를 계산해보면 에러 체크 비트는 다음과 같다.

P1 = 3, 5, 7, 9, 11 에서 데이터 비트 1 의 개수가 짝수이므로 0 을 적용한다.

P2 = 3, 6, 7, 10, 11 에서 데이터 비트 1 의 개수가 홀수이므로 1 을 적용한다.

P3 = 5, 6, 7, 12 에서 데이터 비트 1 의 개수가 홀수이므로 1 을 적용한다.

P4 = 9, 10, 11, 12 에서 데이터 비트 1 의 개수가 짝수이므로 0 을 적용한다.

P4	P3	P2	P1
0	1	1	0

에러 체크 비트 0110 을 10 진수로 바꾸면 6 이 된다. 6 번째 데이터 비트가 에러임을 알 수 있다 따라서 6 번째 데이터 비트 0 을 1 로 바꾸면 해밍 코드를 이용한 오류 정정이 이루어진다.

문제 5. 한글코드를 종류별로 설명하고 한글코드를 8 비트로 처리할 수 있는 방안이 있는지 의견을 개진하시오.

(1) 완성형 (KSC 5601) 코드

음절 한 문자마다 한 개의 코드를 부여하는 방법이 완성형 한글이다. 즉, '가'라는 글자는 "10110000 10100001"로, '각'이라는 글자는 "10110000 10100010"으로 정의하는 방식으로 2 바이트에 글자별로 코드를 부여하는 방법이다. 이 코드는 한글을 초성, 중성, 종성으로 나누지 않고 글자 단위로 처리하기 때문에 조합형에 비해서 입력과 출력 처리가 매우 단순하게 된다. 이 코드는 한글을 나타내는 코드의 첫 번째와 두 번째 바이트의 상위 비트 (MSB) 가 모두 "1"로 되어 있다. 8 비트 ASCII 영문자 코드는 상위 비트가 "0"으로 세트되기 때문에, 한글과 영문 ASCII 코드가 중복되지 않으므로 처리가 용이한 장점이 있다.

(2) 2 바이트 조합형 코드

2 바이트의 연속된 16 비트에 초성, 중성, 종성을 각 5 비트씩 할당하여 이를 자모의 조합에 의하여 한글 문자를 표시할 수 있게 규정한 한글 코드이다. 처음 한 바이트의 상위 비트(MSB)에 1 이 세트되면 한글 코드로 인식하게 되어 다음 연속으로 오는 한 바이트를 처음과 연결된 하나의 코드로 구성한다. 그러나 MSB 에 0 이 세트되면 영문 코드로 인식하여 다음에 오는 한 바이트는 처음의 한 바이트와 무관하게 된다.

(3) 유니코드(unicode)

유니코드는 기존의 ASCII 8 비트 체계를 16 비트(2 바이트) 코드체계로 확장 전환시킴으로써 전 세계의 문자 체계를 수용할 수 있게 하는 컴퓨터 표준을 마련한 셈이다. 현재 이 코드는 UNIX, Linux, NT 이상의 윈도우 시스템에서 사용하고 있으며, Java 에서도 사용되고 있다. 1995 년에 발표된 유니코드 2.0 에서는 조합할 수 있는 모든 완성형 한글 글자 11,1172 자가 "가나다"순으로 정렬(sort)된 상태로 배열되어 있고, 조합형 한글도 포함되어 있다. 따라서 이 체계에서 한글 코드는 완성형 코드 형태이지만 조합형의 장점도 모두 가지고 있는 것이다. 유니코드는 보통 4 자리의 16 진수로 ₩uhhhh 형식으로 나타낸다. 여기에서 ₩u 는 유니코드를 표시하고 으는 16 진수 한 자리를 표시한다. 예를 들어, 알파벳 "A"를 유니코드로 표현하면 ₩u0041 이 된다.

한글코드를 8 비트로 처리할 수 있는 방안에 대한 의견

초성

ㄱ,ㄲ,ㄴ,ㄷ,ㄸ,ㄹ,ㅁ,ㅂ,ㅃ,ㅅ,ㅆ,ㅇ,ㅈ,ㅉ,ㅊ,ㅋ,ㅌ,ㅍ,ㅎ 19 개

중성

ㅏ,ㅐ,ㅑ,ㅒ,ㅓ,ㅔ,ㅕ,ㅖ,ㅗ,ㅘ,ㅙ,ㅚ,ㅛ,ㅜ,ㅝ,ㅞ,ㅟ,ㅠ,ㅡ,ㅢ,ㅣ 21 개

종성

ㄱ,ㄲ,ㄳ,ㄴ,ㄵ,ㄶ,ㄷ,ㄹ,ㄺ,ㄻ,ㄼ,ㄽ,ㄾ,ㄿ,ㅀ,ㅁ,ㅂ,ㅄ,ㅅ,ㅆ,ㅇ,ㅈ,ㅊ,ㅋ,ㅌ,ㅍ,ㅎ,[X] 27+1 개

한글의 초성 중성 종성에 들어가는 자모는 위와 같이 각각 19, 21, 28 의 가짓 수로 5 비트로 표현 할 수 있으므로

저는 2 바이트 조합형 코드의 변형으로 한글 자모를 8 비트(1 바이트)에 담는 조합형 코드를 생각했습니다. ([] = 비트 한 자리)

[](한글 MSB 1 비트) [][](초성, 중성, 종성 표현 2 비트) [][][][][](자모 표현 5 비트)

기존의 조합형처럼 MSB 를 1 로 설정하여 128 이전까지의 ASCII 코드와의 호환성을 고려하였고

초성, 중성, 종성 표현 2 비트는

00(0) - 초성

01(1) - 중성

10(2) - 종성

11(3) - 예약(옛한글, 한글 전용 특수 기호)

으로 사용하여 최종 사용자단의 IME(Input Method Editor)나 프로그램이 각 자모가 어느 자리의 자모인지 구분하여 처리할 수 있도록 설계하였고

예약 구역을 지정하여 32 개의 추가적인 한글과 관련된 문자를 표현 할 수 있게 할당했습니다.