6주차 예비보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 전 가산기 및 반 가산기**

우선, 가산기란 input이 들어왔을 때 input의 합(sum)과 자리올림(Carry)을 출력하는 논리회로이다.

그 중 가산기 회로중에 가장 간단한 반 가산기(Half Adder)는 두 개의 입력(1-bit)을 더하고 합비트와 캐리비트를 출력하는 조합 산술 회로이다. 캐리 비트를 고려하지 않기 때문에, 주로 이진 합에서 최하위 숫자를 더하는데 사용된다. 반 가산기 회로를 k-map을 이용해 표현식을 생각해볼 수 있다. 우선 반 가산기의 진리표 예는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

저번 시간에 배운 k-map을 사용하게 되면,

합 캐리

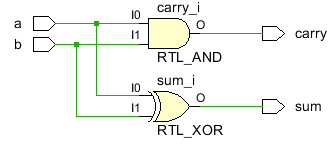
|  |  |  |
| --- | --- | --- |
| B \ A | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| B \ A | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

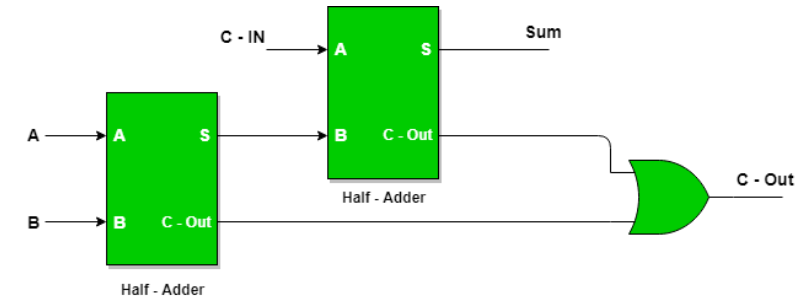
합의 경우 S = (A’B) + (AB’) 이므로, 간단하게 S = A XOR B 혹은 A   
{\displaystyle \oplus }⊕ B 이다.

캐리의 경우 C = AB이다.

따라서, 반 가산기를 스케메틱으로 보면, 다음과 같다. ( NAND와 NOR로 구현가능)



전 가산기(Full Adder)는 반 가산기와 다르게 3개의 input을 가지며 2개의 output을 가지는 가산기이다. 3개의 input은 이전 자리올림(C-in)과 두 개의 입력(A,B)로 이루어지며 출력은 자리올림(C-out)과 합이 된다. 조금 생각해보면, 전 가산기는 두 개의 반 가산기와 하나의 OR게이트로 이루어진다. 이러한 특징 덕분에 반 가산기라는 명칭을 얻었으며, 전 가산기가 실질적 기초 덧셈 회로이다.



전 가산기의 진리표 예는 다음과 같이 작성할 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C-in | Sum | C-Out |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

앞서 반 가산기와 동일하게 k-map을 통해 논리식을 구할 수 있다.

합

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A \ BC-in | 00 | 01 | 11 | 10 |
| 0 |  | 1 |  | 1 |
| 1 | 1 |  | 1 |  |

합의 경우 S = AB’C’ + A’B’C + ABC + A’BC’ 이다. 이는 불 대수 정리를 사용하면

S = A(B’C’ + BC) + A’(B’C + BC’) = A(B⊕C)’ + A’(B⊕C) = A⊕(B⊕C) 가 된다.

캐리

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A \ BC-in | 00 | 01 | 11 | 10 |
| 0 |  |  | 1 |  |
| 1 |  | 1 | 1 | 1 |

카르노 맵에 의해 인접한 1끼리 묶어주게 되면, 3가지로 묶을 수 있다.

따라서 C-out = BC + AC + AB 가 된다. (C는 C-in) 혹은, 강의자료와 동일한 식을 얻고 싶다면, C-out = AB + AC + BC ( A + A’) 에서 정리를 시작하면 된다. 이 식은 = AB (1+C) + AC + A’BC

= AB + AC + A’BC

= AB + AC(B + B’) + A’BC

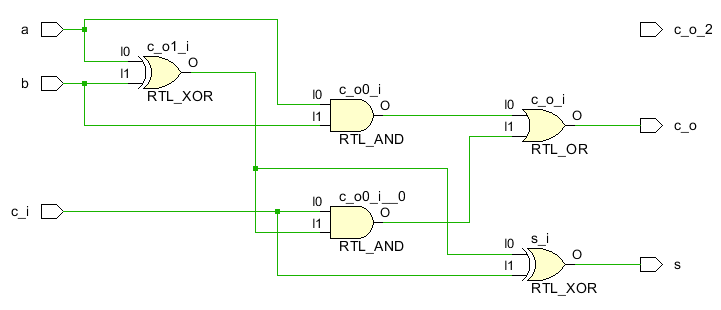
= ABC + AB + AB’C + A’BC

= AB(C+1) + AB’C + A’BC

= AB + AB’C + A’BC = AB + C( AB’ + A’B) = AB + C (A⊕B)

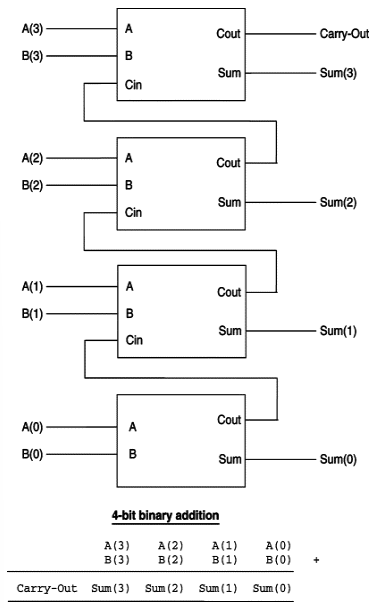
C-out의 두 표현식의 진리표를 작성하게 되면, 동일함을 알 수 있다. (또는 시뮬레이션)

전 가산기를 스케메틱으로 보면, 왜 두 개의 반가산기와 하나의 OR게이트인지 쉽게 이해가 가능하다. (마찬가지로, NAND와 NOR 구현 가능)

  
{\displaystyle \oplus }

전 가산기와 반 가산기 여러 개를 사용하면, n비트 덧셈을 만들 수 있다.

예로, 1001 과 0101을 더한다고 생각해보면 가장 하위 비트를 더 할땐 반가산기를 사용하며 그 이후부턴 전 가산기(캐리비트와 2개의 비트)를 사용한다. 따라서, 1110이 된다.



**2. 전 감산기 및 반 감산기**

감산기란, 두 개의 이상의 이진수를 입력받았을 때, 입력 값의 차(Difference)와 빌림 수(Borrow)을 출력으로 하는 논리 회로이다. 즉, 뺄셈 연산이다.

그 중 반 감산기는 반 가산기와 유사하게 두 개의 이진수를 입력으로 받고 하위 비트에서 요구하는 빌림수는 입력받지 않으며 차와 빌림수를 출력한다. 빌림 수를 고려하지 않기 때문에 가장 하위 숫자를 빼는데 주로 사용한다. 반 감산기 회로 역시 k-map을 통해 표현식을 얻을 수 있다. 입력 값 A를 Minuend 비트(피감수)라고 하며, B를 Subtrahend 비트(감수)라 부른다. (B – A 가 아님!)

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Diff | Br |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

저번 시간에 배운 k-map을 사용하게 되면,

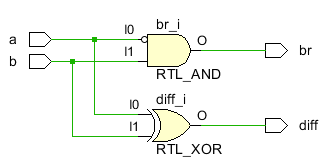
차 빌림

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B \ A | 0 | 1 |  | B \ A | 0 | 1 |
| 0 | 0 | 1 |  | 0 | 0 | 0 |
| 1 | 1 | 0 |  | 1 | 1 | 0 |

차의 경우 가산기와 동일하게 Diff = (A’B) + (AB’) 이므로, 간단하게 Diff = A XOR B 혹은 A   
{\displaystyle \oplus }⊕ B 이다.

캐리의 경우 Br = A’B이다.

따라서, 반 감산기를 스케메틱으로 보면, 다음과 같다. ( NAND와 NOR로 구현가능)



전 감산기의 경우 전 가산기와 유사하게 3개의 입력 A,B와 빌림수(하위 비트의 빌림 수)를 가지며, 차와 빌림수를 출력한다. 마찬가지로, 전 감산기는 2개의 반 감산기와 하나의 OR게이트로 구현 가능하다. 진리표예는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Bin | Diff | Br |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

마찬가지로 k-map을 적용하면,

차

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A \ BBin | 00 | 01 | 11 | 10 |
| 0 |  | 1 |  | 1 |
| 1 | 1 |  | 1 |  |

전 가산기와 동일하게 Diff = A⊕B⊕Bin 이 나오게 된다.

빌림 수

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A \ BBin | 00 | 01 | 11 | 10 |
| 0 |  | 1 | 1 | 1 |
| 1 |  |  | 1 |  |

앞선 전가산기와 동일한 방법으로 3개의 묶음을 만들 수 있다.

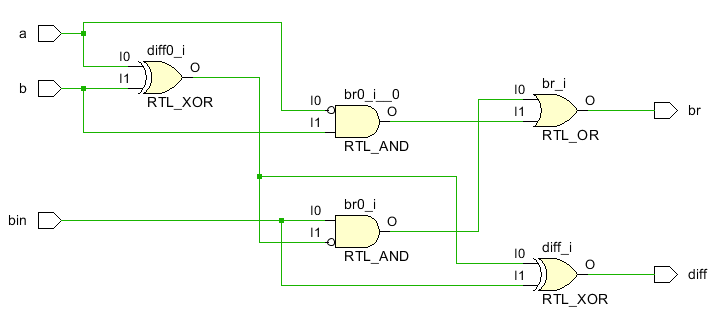
Br = A’Bin + A’B + BBin 또한, 강의자료와 같은 모양의 식을 얻기 위해선

= A’B’Bin + A’BBin’ + A’BBin + ABBin

= Bin(AB + A’B’) + A’B(Bin + Bin’)

= Bin (A XOR B)’ + A’B 가 됨을 알 수 있다.

스케메틱은 다음과 같다. 두 개의 반 감산기가 있는 것을 확인 할 수 있다.



n개의 반 감산기와 전 가산기를 통해 n bit 감산기를 구현 할 수 있다.

**3. BCD 가산기**

BCD는 Binary coded decimal로 이진 코드 십진법을 의미한다. 거창해 보이지만, 단순히 0~9를 4bit를 사용해 표현하는 것이다.

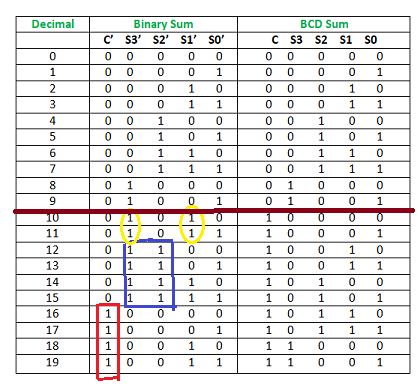
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 십진법 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

이런 표현 방식은 변환이 편하나, 0000과 같이 사용하지 않는 데이터가 많아 데이터 크기가 커진다.

결국 BCD 가산기는 10진수를 BCD로 표현한 것을 입력으로 받고 가산기의 역할을 수행한다. 이때, 출력은 BCD로 출력한다. 당연하게도 반 가산기와 유사하게 캐리를 고려하지 않는다면, 출력은 0~18이 되며 캐리를 고려할 경우 1 + 18 = 19가 된다.

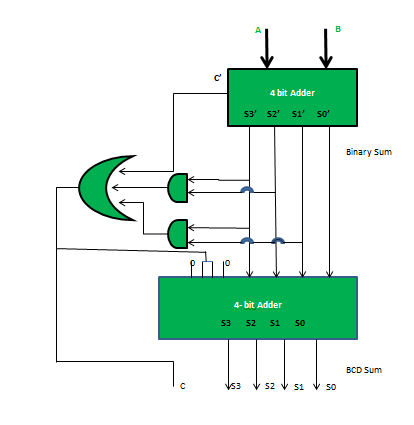
예를 들어, A = 1001 과 B = 0011을 BCD가산기를 이용해 더한다고 생각해보자. 이진 합계는 9 + 3 = 12 이므로, 일반적인 가산기를 통해 합산한다면 본래 1100을 출력해야 하지만 BCD가산기의 경우 1 0010 으로 출력한다. 그렇다면 A = 0011 과 B = 0100 일때는 어떻게 될까? 이는 일반 이진 합과 동일하게 출력하며 결과는 3 + 4 = 7 이므로 0111 이다. 결론적으로 BCD가산기와 일반 이진합의 차이는 2진수로 0110만큼 다름을 알 수 있다.

다음 표를 참고하면 한눈에 볼 수 있다. 이를 통해 , C’(캐리) + S3’S2’ + S3’S1’ = 1임을 알 수 있다. (10 ~ 19가 되는 경우)



이 외에도 Packed BCD(묶음 이진화 십진법)과 더 효율적인 공간을 위한 첸-호 부호화, 집적된 묶음 십진법 등이 있다.

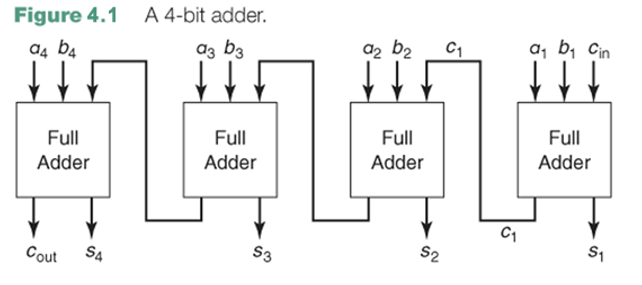
다음은, BCD가산기를 병렬가산기를 이용해 나타낸 것이다.



**4. 병렬 가감산기**

병렬이란, 나란히 늘어선 것을 말한다. 따라서, 병렬 가감산기는 나란히 늘어선 가감산기를 의미한다. 앞서 배운 가산기와 감산기는 모두 단일 비트에 대한 연산이었다. 이를 n개의 비트에 대한 연산을 하기 위해 만든 것이 병렬 가감산기 이다. 앞서 잠깐 언급한 1개의 반 가산기와 3개의 전 가산기를 이용해 만든 4 bit 가산기 역시 병렬 가산기로 부를 수 있다.

다음은 4개의 전 가산기로 만들어진 4-bit ripple carry adder 이다.



하지만, 이는 가산기의 역할만 할 뿐 감산기의 역할은 하지 못한다. Complement를 이용하면 가산기 뿐 아니라 감산기도 가능한 병렬 가감산기를 구현 할 수 있다.

알다시피, 이진수의 뺄셈은 앞서 소개한 방법 뿐 아니라 2의 보수의 덧셈으로 구현 할 수 있다.

예를들어 110110 – 10110을 한다고 가정하자. Subtrahend는 10110으로 비트 수가 5이며, Minuend의 비트 수는 6이다. Subtrahend에 2의 보수를 취한 다음 더하게 되면 뺄셈의 결과를 얻어낼 수있다. 10110은 010110과 같으며 이를 2의 보수를 취하게 되면 101101 + 1 = 101010 이 된다. 이제 110110과 101010을 더하고 캐리되는 값을 제거하면, 100000이 됨을 알 수 있다. 이런 아이디어를 통해, 다음과 같이 회로의 설계가 가능하다.

벽, 실내, 방이(가) 표시된 사진

자동 생성된 설명

Signal 비트 S와 B를 XOR을 하게 되면 2의 보수가 된다. 이를 병렬 가감산기 회로라 부른다.

**5. Carry Look-Ahead Adder / Ripple Carry Adder**

우선, Ripple Carry Adder는 앞서 소개한 바와 같이 4개의 전가산기를 연결한 병렬 가산기를 4-bit Ripple carry adder라고 부른다. 자릿수가 ripple(물결)치듯 움직인다 하여 이런 이름이 붙었다. 또한, 이렇게 연속적으로 순서로 연결되어있는 방식을 cascade로 연결되어있다(carry chain)라고 한다. 이런 방식은 간단하게 구현이 가능하지만, 최종 출력이 나오기 까지 시간이 많이 걸린다. 당연하게도 delay시간이 늘어나며, (이를 Critical path라고도 부른다.) 만큼의 delay가 생긴다. 이런 캐리 전파 지연 속도를 개선하면, 다른 산술 연산 속도 역시 빨라지게 되기 때문에 delay를 줄이는 것은 매우 중요하다. 따라서, 속도를 빠르게 하기 위해 Carry Look Ahead Adder 가 등장했다.

Carry Look-Ahead Adder의 기본 아이디어는, 입력받는 신호를 기반으로 캐리 신호를 미리 게산하여 문제를 해결하는 것이다. 생각해보면, 두 개의 입력값 A\_i와 B\_i의 캐리 여부는 바로 접근 가능하다. 따라서, 이 계산을 위해서 이전 비트의 캐리 값을 기다릴 필요가 없다.

들어가기 앞서 여러 정의와 표현을 이해하고 넘어가야 한다.

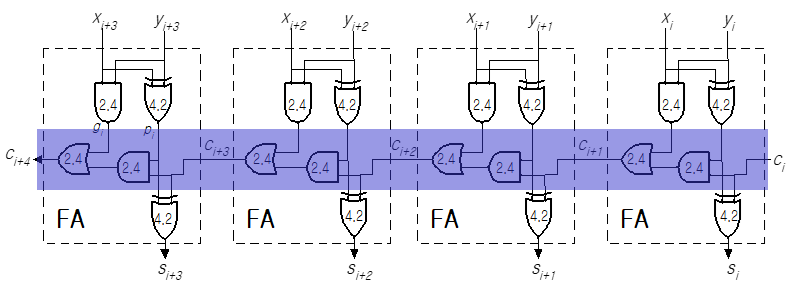
1. Carry-generate function : G\_i = A\_iB\_i -> A\_i와 B\_i 모두 1이면 이전 비트와 관계없이 무조건 캐리비트를 생성한다.
2. Carry Propagation function : p\_i = A\_i⊕B\_i -> 둘 중 하나만 1이면 이전 비트의 캐리(C-in)이 캐리(C-out)로 출력

이를 바탕으로 다음과 같은 표현이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

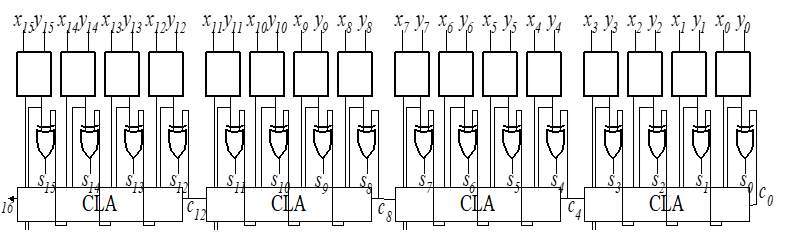
최종적으로 회로는 더욱 더 복잡하게 되지만, 속도는 향상된 다음과 같은 모습을 띈다. (C\_(i+4) 를 Carry-look-ahead-generator는 2 gate delay만 걸린다.)



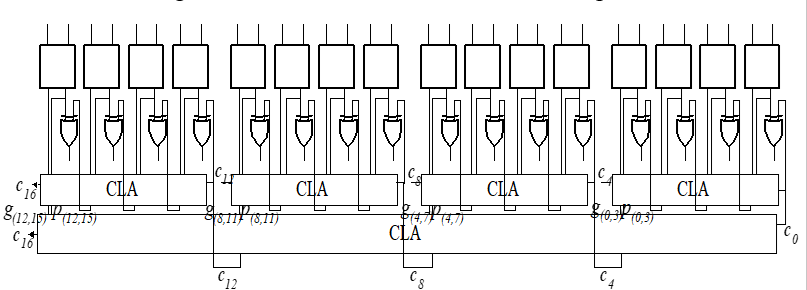
하지만, 계산에 AND/OR게이트가 추가적으로 필요하므로 fan-in문제가 나타나곤 한다. (총 (n(n+1)/2 만큼의 and/or 게이트가 필요)

**6. 기타 이론**

16 bit나 32bit 그 이상의 회로도 구현 가능하다. 앞서 구현했던 4bit 4개를 cascade연결하면 다음 16-bit 회로가 완성된다. (32-bit 는 2개의 16-bit)



그러나, 여기서 더 발전하여 two-level CLA를 사용하는 방법도 존재한다.

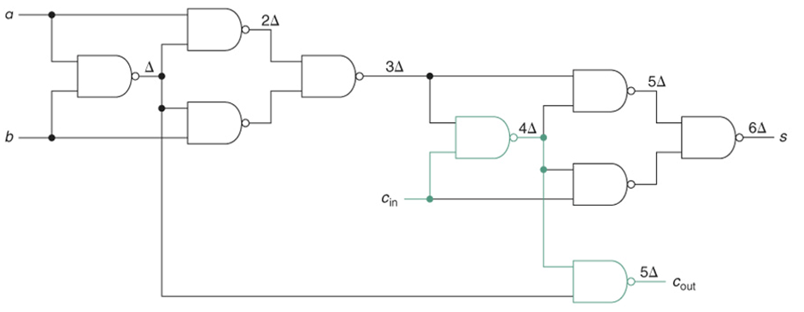


텍스트, 손목시계이(가) 표시된 사진

자동 생성된 설명

BCD 코드는 여러가지가 있다. 5421, 2421 등이 있으며 그 중 8421을 많이 사용한다. 그러나, 요즘은 아스키코드를 데이터 통신에서 가장 많이 사용한다.

게이트 딜레이는 참고로 다음과 같다.



Full\_adder 하나를 추가할 때 마다 게이트 딜레이가 2△증가한다. 비슷하게, 앞서 설명한 Carry look ahead adder에서 g\_i와 p\_i는 1△ 증가한다. 그래서 1개의 Carry look ahead adder의 총 게이트 딜레이는, 1 + c\_i + s\_i 인데 c\_i는 2, s\_i는 3만큼이므로 6만큼의 게이트 딜레이가 생긴다.

**7. 참고문헌**

<https://www.geeksforgeeks.org/full-adder-in-digital-logic/>

<https://www.geeksforgeeks.org/bcd-adder-in-digital-logic/>

디지털 회로개론 chap4 Designing Combinational Systems. 김주호 교수

<https://cms3.koreatech.ac.kr/sites/yjjang/down/dsys10/M05_arith1.pdf>