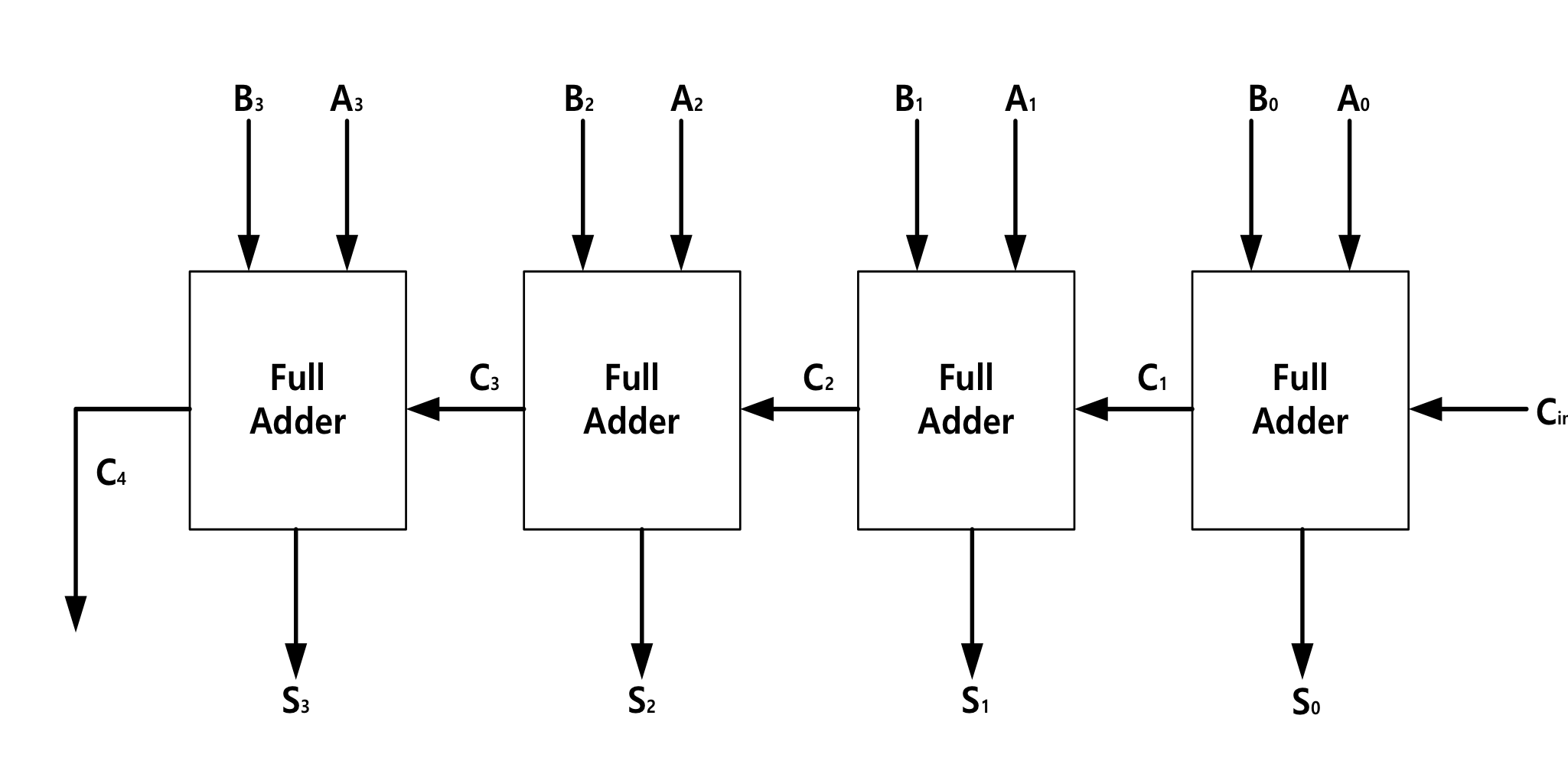
10주차 예비보고서

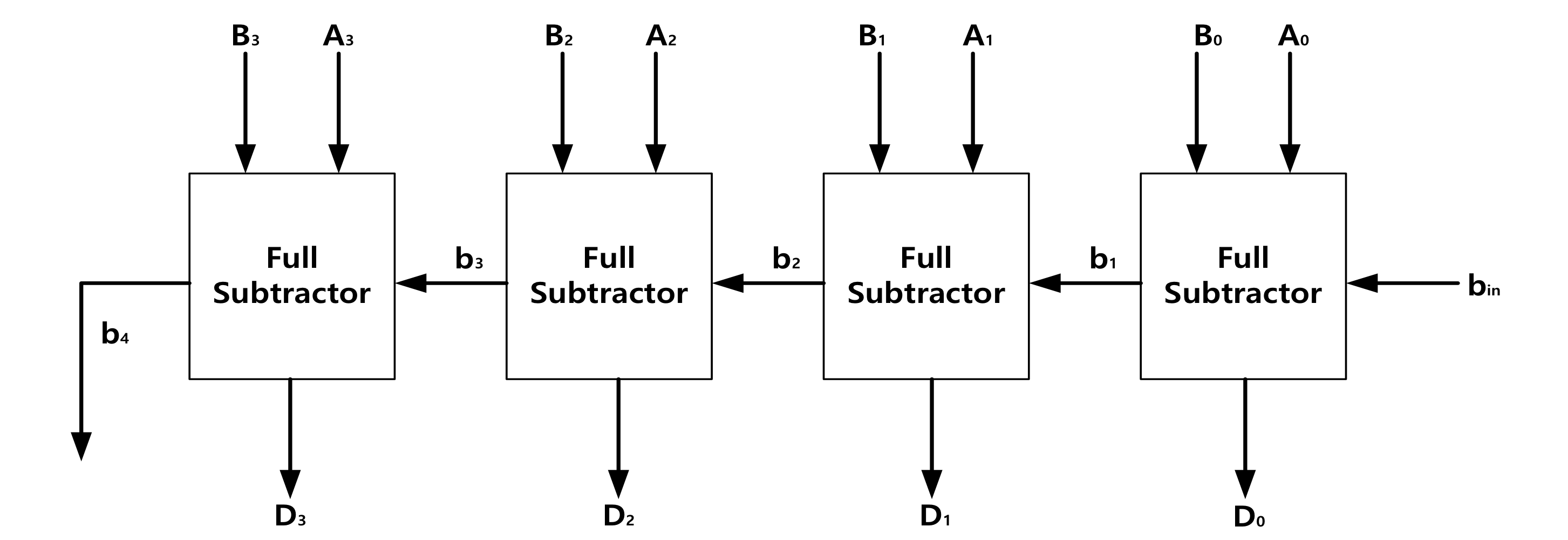
전공: 컴퓨터공학과/생명과학과 학년: 9학기 학번: 20181435 이름: 박다희

**1.**

4-bit adder와 4-bit subtractor는 ­1-bit full adder와 1-bit full subtractor을 4개 병렬 연결하여 2개의 4-bit 이진수를 더하고 빼는 연산을 수행하는 논리 회로이다.



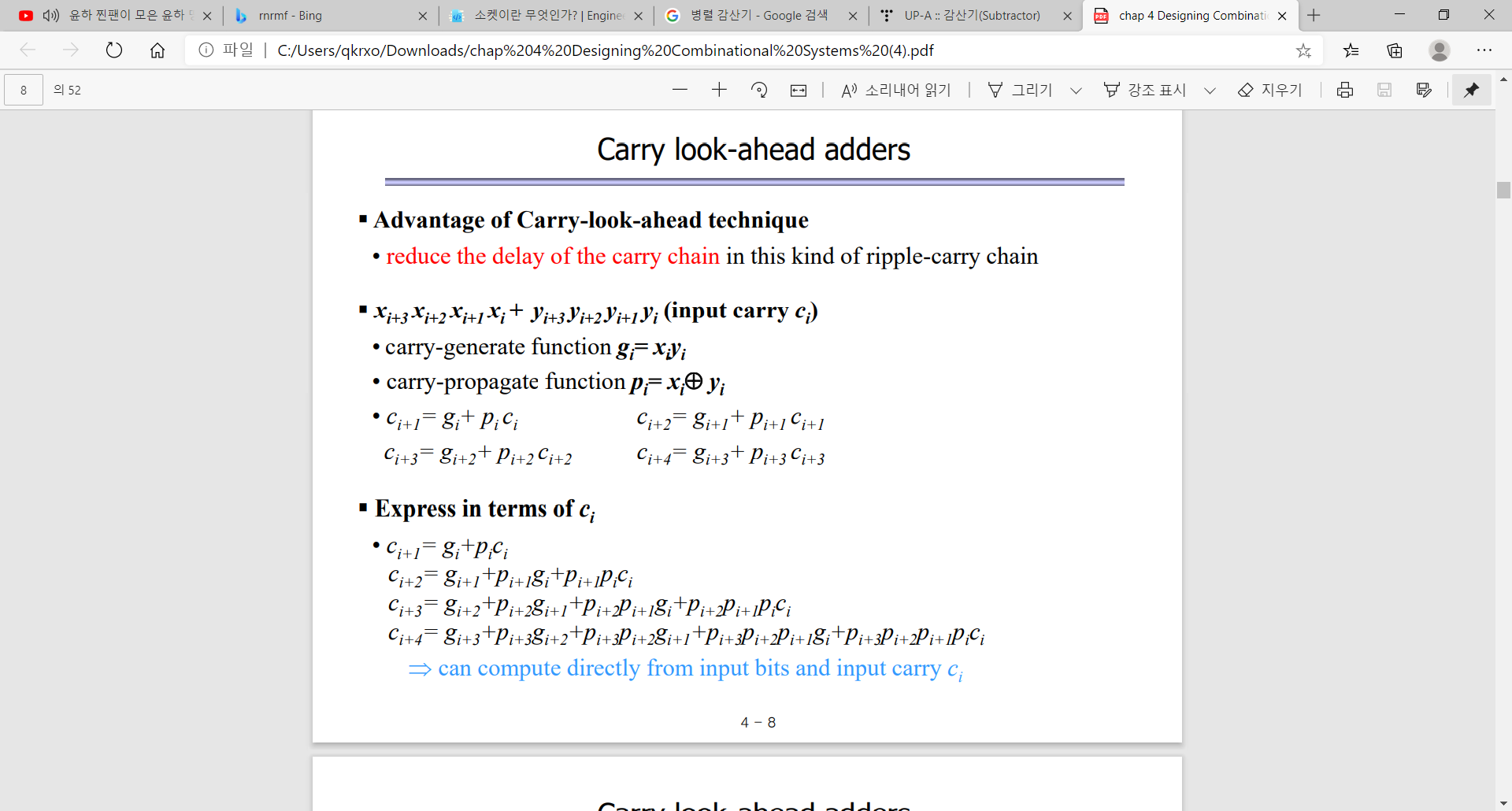
위의 그림은 4-bit adder로, 두 개의 4-bit 이진수 A(A3A2A1A0), B(B3B2B1B0)을 더하여 출력 값으로 S3S2S1S0과 C4를 가지는 논리 회로이다. 그림에서 보면 알 수 있다시피 4-bit adder는 두 이진수의 하위 자릿수부터 덧셈 연산하여 S0, S1, S2, S3 순으로 출력하는데 이 출력 값은 An, Bn, Cn 값들 중에서 1의 값을 가지는 입력 값이 홀수 이면 1, 짝수 이면 0을 갖는다. Cn은 carry을 의미하는 값으로 1인 입력 값의 개수가 2개 이상일 때 1을 가지고 그 외에는 0의 값을 가진다. 이렇게 한 자릿수에서 bit 계산한 후 carry가 생성된다면 다음 full adder에 넘겨준다.



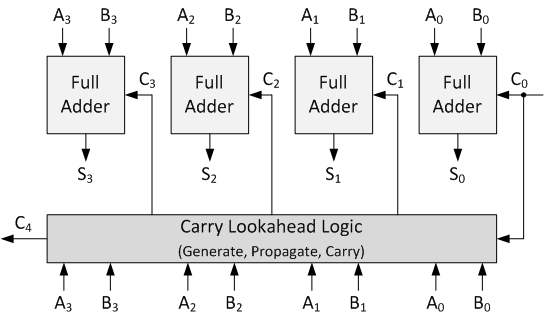
위의 그림은 4-bit subtractor로, 두 개의 4-bit 이진수 A(A3A2A1A0), B(B3B2B1B0)을 뺄셈하여 출력 값으로 D3D2D1D0과 b4를 가지는 논리 회로이다. 그림에서 보면 알 수 있다시피 4-bit subtractor는 두 이진수의 하위 자릿수부터 뺄셈 연산하여 D0, D1, D2, D3 순으로 출력하는데 이 출력 값은 An, Bn, Cn 값들 중에서 1의 값을 가지는 입력 값이 홀수 이면 1, 짝수 이면 0을 갖는다. bn은 borrow을 의미하는 값이고 이 값은 이전 자릿수에서의 빌림 수로 한 자릿수에서 bit 계산한 후 borrow가 생성된다면 다음 full subtractor에 넘겨준다.

**2.**

1번에서 설명했던 4-bit subtractor와 4-bit adder는 하위 자릿수에서 상위 자릿수 순으로 차례대로 연산하여 ripple carry adder또는 ripple carry subtractor고 일컫는데, 이는 최종 출력 값을 나타내기까지 시간 지연이 생긴다. 이전 자릿수에서 생성된 carry를 받아야지만 다음 연산을 진행할 수 있으므로 bit가 커질수록 시간 지연도 점점 커진다. 이러한 문제를 해결하기위해 look ahead carry를 사용한다. 이 방법은 carry를 하위 자릿수에서 하나하나 계산하는 것이 아니라 carry-generate signal과 carry-propagate signal 값을 이용하여 이전 자릿수에서 carry가 넘어오길 기다리지 않고 바로바로 full adder의 carry값을 계산하는 것이다. 이를 계산하는 함수는 다음 그림과 같다.

****

만약 입력 값으로 C0의 carry가 들어온다면 C1는 g0 + p0c0이고, C2는 g1 + p1c1인데 이는 다시 g1 + p1(g0 + p0c0)로 나타낼 수 있다. 이렇게 마지막 자릿수까지 적용하고 나면 각각의 자리에서 생성된 carry 값을 gn, pn, C0로 나타낼 수 있다. 따라서 이전 자릿수에서 carry가 계산되길 기다리지 않아도 모든 carry값을 구할 수 있다.



**3.**

도표, 스케치, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명 먼저 XOR는 1⊕x=x’이며 0⊕x=x이고, 이를 활용한 2’s complement는 부호 선택 신호(M의 기호로 표현하였다)와 함께 수행되는데 만약 4-bit의 두 개의 입력 값 A(A3A2A1A0), B(B3B2B1B0)가 존재하고 이를 덧셈, 뺄셈 연산을 수행한다고 했을 때 Bn이 0이고 M도 0이면 Bn⊕M값은 0, S값이 1이면 Bn⊕M값은 1, Bn이 1이고 M도 1이면 Bn⊕M값은 0, M값이 0이면 Bn⊕M값은 1이 된다. 덧셈 연산을 수행할 때는 M가 0이고, 뺄셈 연산을 진행할 때는 M는 1이다. 그렇기 때문에 XOR gate을 사용하여 Bn을 inversion시키고 계산한다.

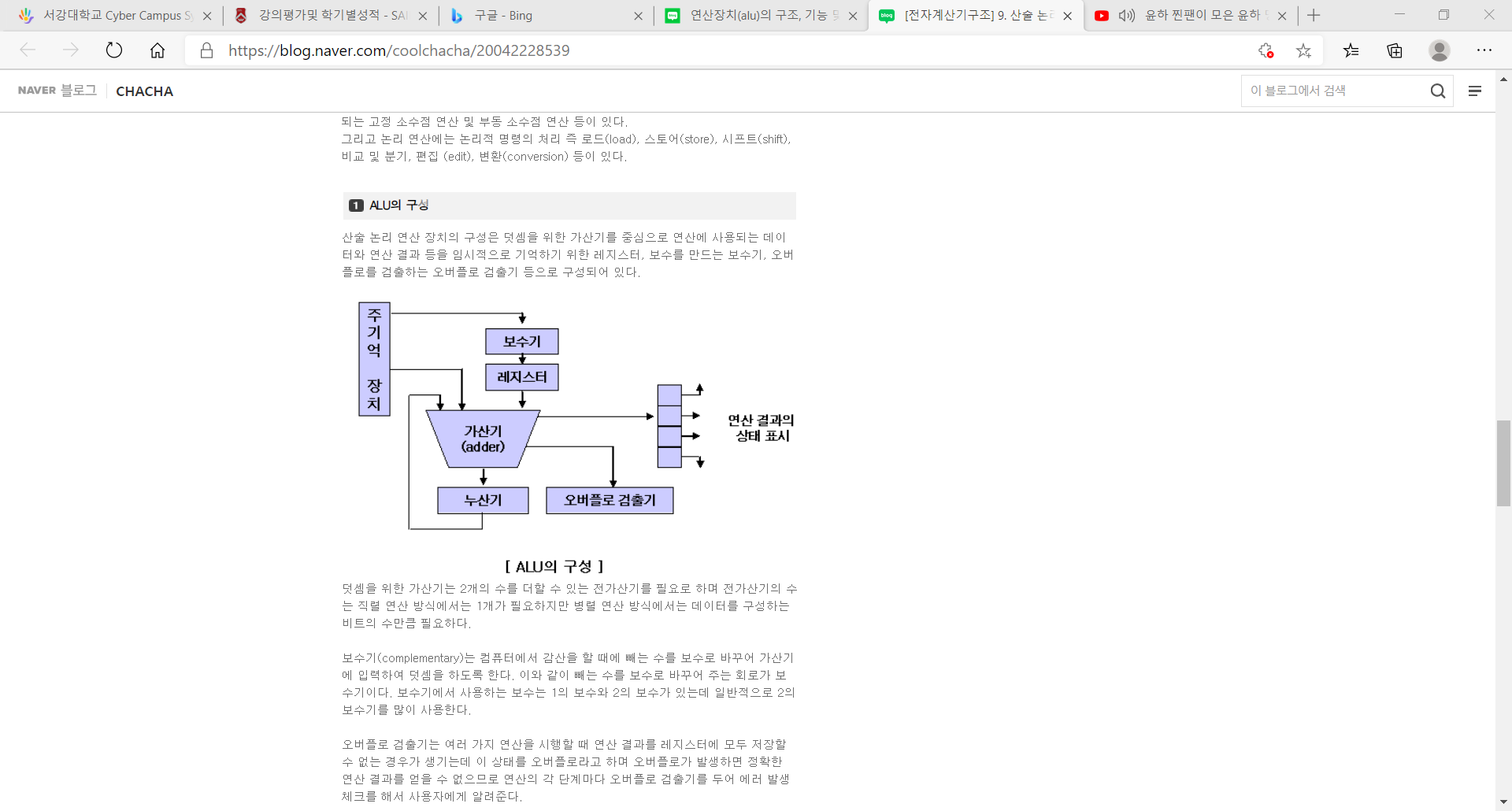
위의 그림을 살펴보면 Bn 값은 inversion되므로 입력 값으로 Bn’이 들어가지만 M의 값이 0이라면 full adder에 들어가는 값이 원래의 Bn이 들어가고 Cin값은 0이므로 마치 4-bit adder처럼 실행되고 M이 1이라면 full adder에 들어가는 Bn의 값이 inversion된 그 상태인 B의 1의 보수로 들어가고 Cin 값은 1이어서 An + Bn’ + 1의 연산을 수행하게 된다. 이는 마치 4-bit subtractor처럼 실행된다.

**4.**

BCD는 Binary Coded Decimal의 약어로 십진수로 표현된 숫자의 각 자릿수를 4-bit의 이진수로 나타낸 것을 일컫는다. 각각의 자릿수는 0에서 9까지로만 표현되므로 4-bit의 이진수도 0000에서 1001까지의 수만 사용하고 그 외의 이진수는 don’t care로 처리한다. BCD 연산을 할 때는 각각의 자릿수 별로 계산해주면 되는데 그 결과가 0에서 9사이에 있는 값이라면 문제가 없지만 10이상의 수가 나온다면 따로 처리를 해줘야 된다. 예를 들어 31과 62를 더하고자 할 때 0011 0001 + 0110 0010이고, 이는 1001 0011로 93이다. 그러나 18과 29를 더할 때 이는 0001 1000 + 0010 1001이고 일의 자리에서의 합이 10을 넘는다. 이진수 식 그대로 덧셈을 진행한다면 0011 10001인데 이럴 때는 6인 0110을 더해줌으로 처리하면 된다. 따라서 최종 결과 값은 10001 + 0110 = 10111, 자리 올림 수를 다음 자릿수로 옮기면 47인 0100 0111이다. 같은 방법으로 세 자리 수도 BCD 연산이 가능한데 184와 576을 더할 때 이는 0001 1000 0100 + 0101 0111 1000로 표현되고, 일의 자리부터 덧셈 연산을 하여 그 값이 10 이상이라면 0110을 더해주고 이 때 생긴 자리 올림 수는 다음 자릿수로 넘겨주어 최종 결과 값은 0111 0110 0010이 된다.

**5.**

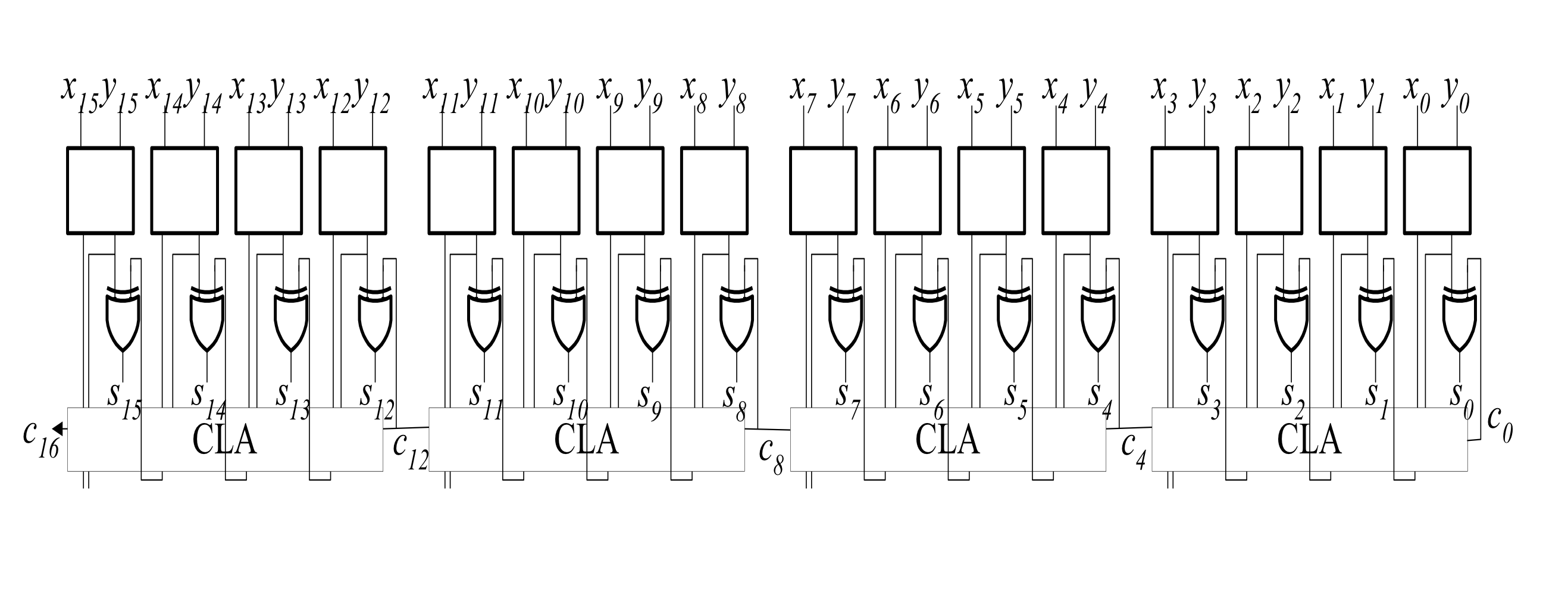
ALU는 Arithmetic Logic Unit의 줄임 말로 덧셈 연산과 뺄셈 연산과 같은 산술 연산이나 대소 비교, 논리합, 논리곱 등 논리 연산을 수행하는 디지털 회로이다. ALU의 구조는 크게 두 부분으로 나뉘는데 하나는 내부 장치 부분이고 다른 하나는 레지스터 부분이다. 특히 내부 장치 부분을 세부적으로 나누면 보수기, 오버플로우 검출기, 시프터, 가산기로 이루어져 있는데 보수기는 complementer로 보수를 만들어 주고 오버플로우 검출기는 해당 계산 결과 값이 레지스터의 용량을 넘는지 확인하는 회로이다. 시프터는 2진수의 각각의 자리를 오른쪽이나 왼쪽으로 옮겨주는 회로이고 마지막으로 가산기는 산술 연산을 실행하는 회로이다.



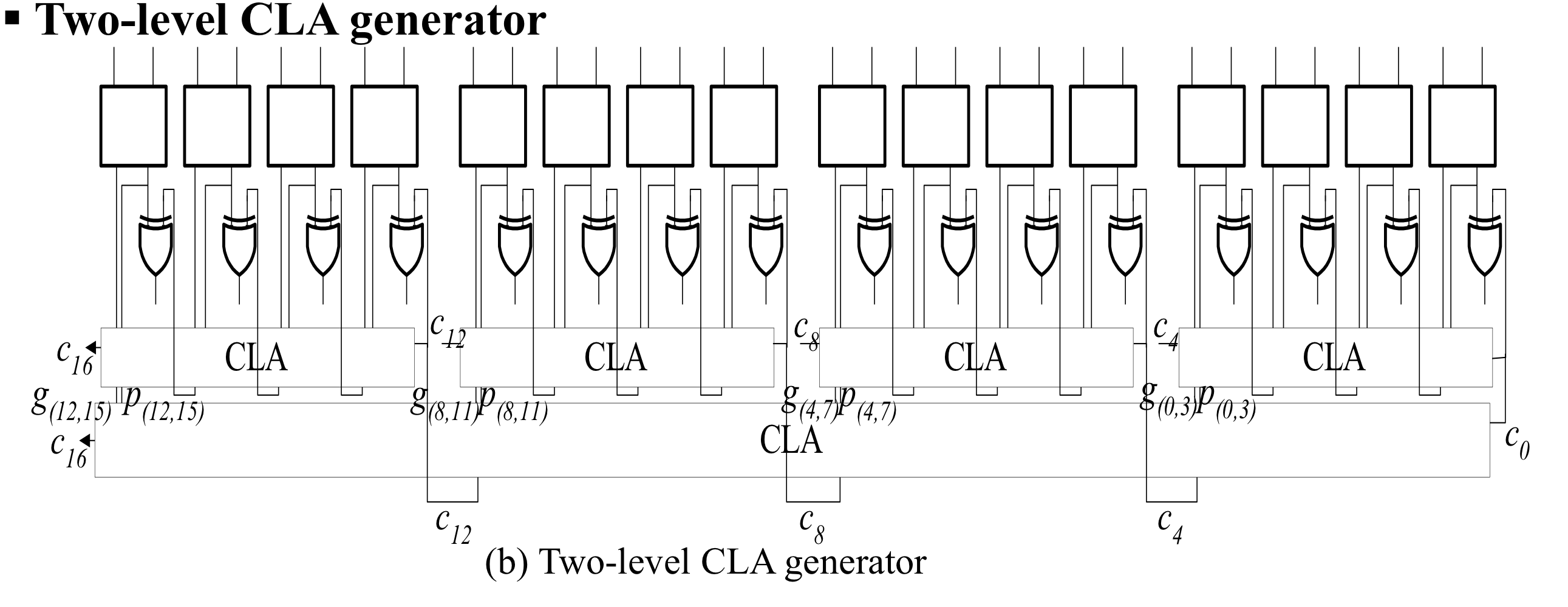
ALU는 여러 논리 마이크로 동작과 산술 마이크로 동작을 실행하는 디지털 회로이기에 하나의 동작을 선택하기 위해서 여러 선택선들을 가진다. 예를 들어 선택 변수가 n개 있다면 서로 다른 개의 동작을 수행할 수 있게 된다. 논리 연산을 수행할 때에는 논리적 명령이 여러 가지 있는데 로드, 편집, 비교 및 분기, 스토어 그리고 변환 등을 예로 들 수 있다. 로드는 기억 장치에서 내부 연산장치로 가져오는 것이고 편집은 pack 10진수를 zone 10진수로 변환하는 것이다 그 역, 그리고 소수점, 콤마 삽입 등을 실행하는 것이고 비교는 데이터 두 개를 대소 비교하는 것이다. 분기는 명령이 실행되는 순서를 바꿔주는 것이고 스토어는 정보를 내부 연산 장치에서 기억 장치로 옮기는 작업이고 변환은 코드를 변환하거나 10진수를 2진수로, 2진수를 10진수로 바꿔주는 역할을 한다. 이 중에서 비교 및 분기, 스토어 그리고 로드는 논리 조작에 속한다.

**6.**

Single-level 16-bit carry look ahead adder는 4개의 4-bit carry look-ahead adder을 연결하여 만든 것으로 parallel 구조 4개를 연속적으로 연결한 형태이고 serial구조와 parallel 구조가 섞여 있다.



이 구조는 여전히 delay가 존재하므로 이 delay을 더 줄이기 위하여 CLA generator을 이중으로 사용하는 2-level 16-bit Carry Look-ahead Adder을 사용하기도 한다.



2-level 16-bit Carry Look-ahead Adder가 delay을 줄이는 방법은 다음과 같다.

