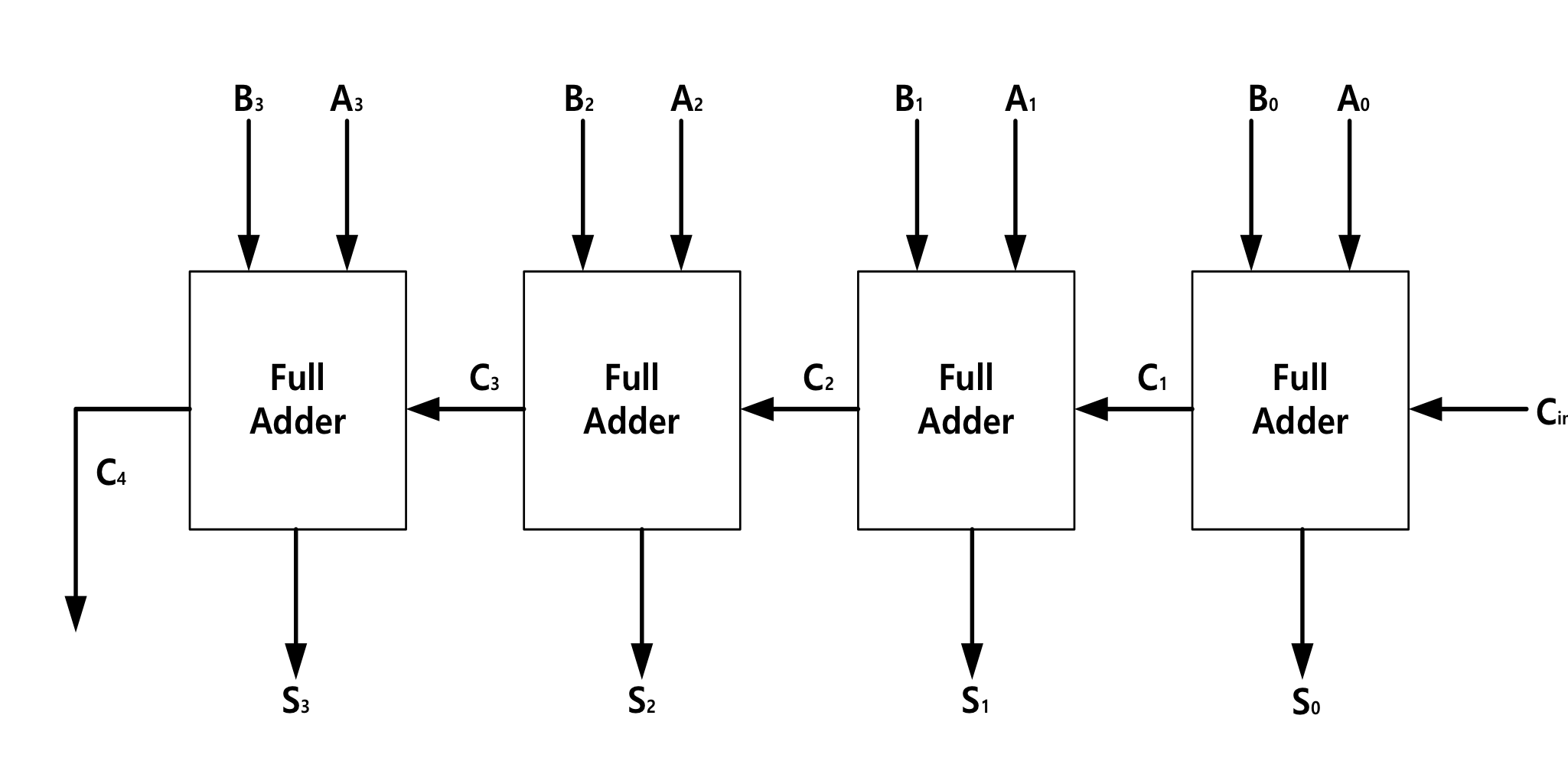
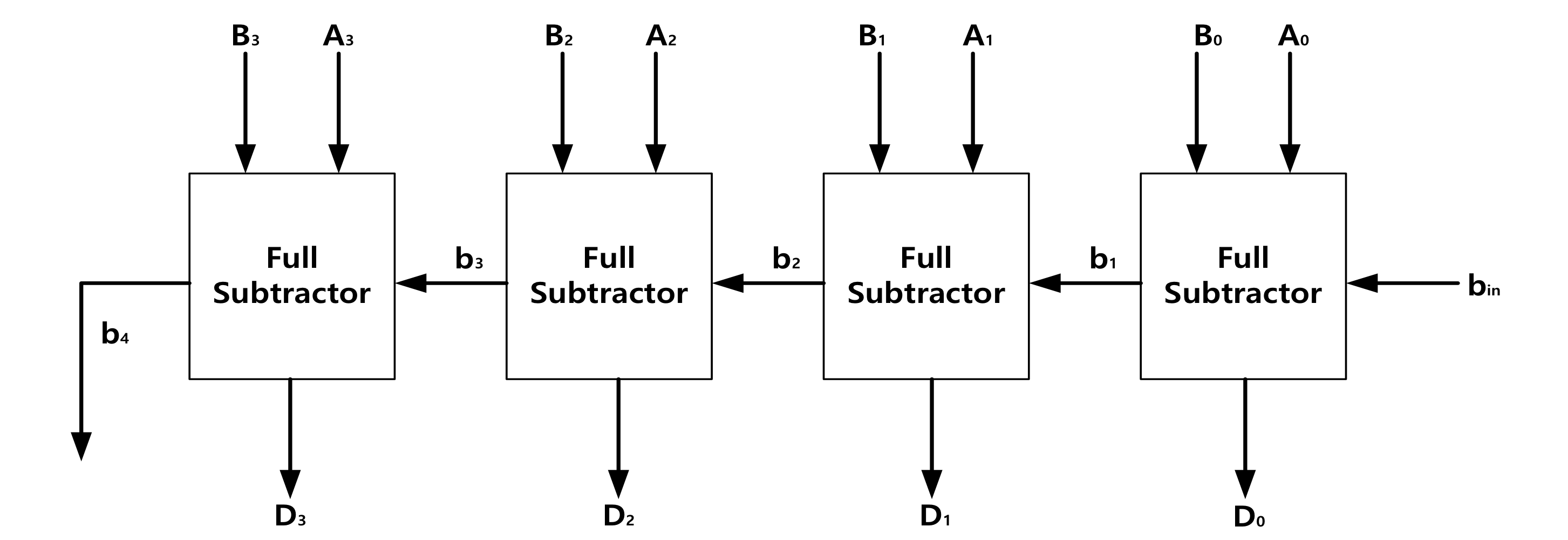
10주차 예비보고서

전공 : 아트앤테크놀로지 학년 : 3학년 학번 : 20191098 이름 : 백승주

**1 .** 이전 주차 때 Full adder에 대해 다룬 적이 있다. Full adder의 경우 1비트의 2진수 두 개와 carry in 을 더하는 회로였다. 이러한 full adder는 더하는 숫자의 크기가 1비트밖에 되지 않기 때문에 4비트 크기의 이진수를 덧셈하기 위해 4개의 full adder를 병렬 연결한 회로가 바로 4-bit adder이다.



4-bit adder의 기본 형태는 위와 같으며 위 그림에서 c값은 input으로 쓰일 때는 이전 가산기에서 발생한 자리올림값을 더하는 역할, output으로 쓰일 때는 현재 가산기에서 발생한 자리올림 값을 다음 값으로 옮겨주는 역할을 한다.\

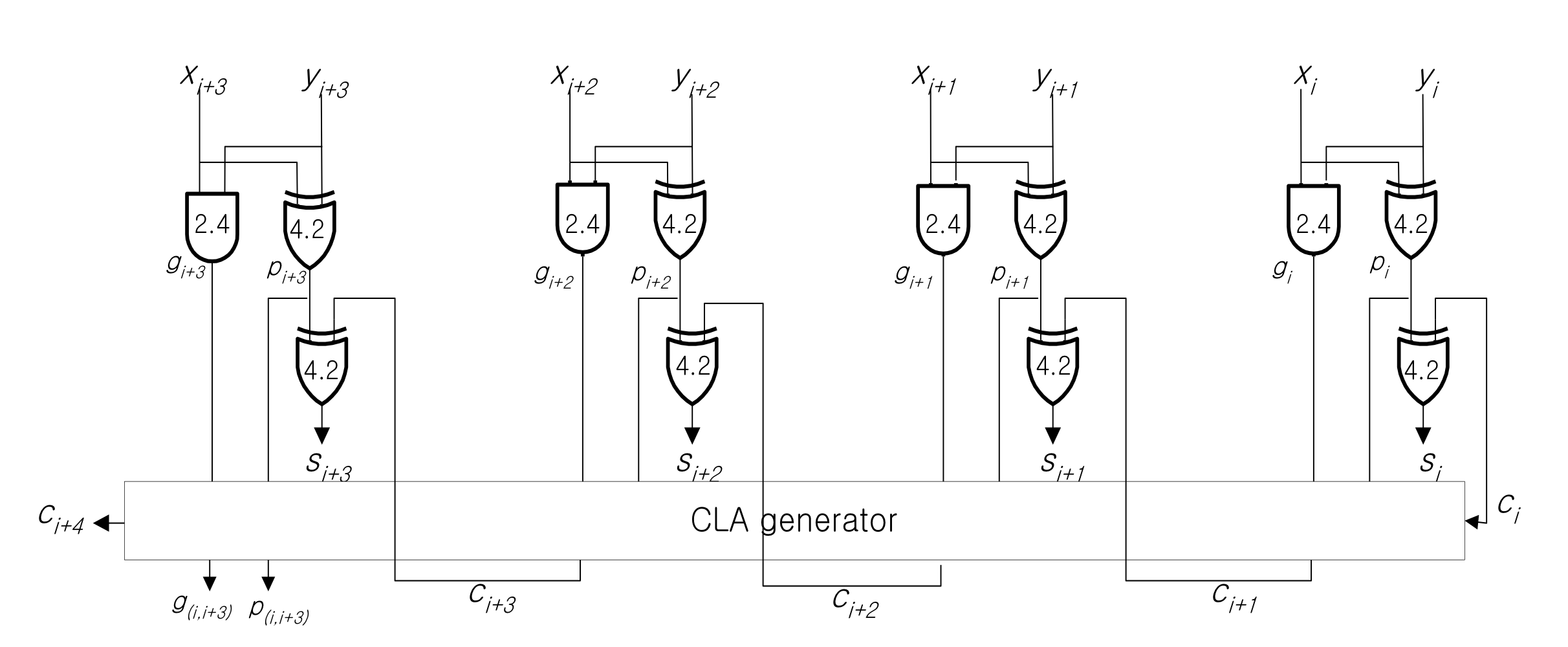


앞서 관찰한 4-bit full adder 처럼 full subtractor를 4개 병렬 연결한 형태가 바로 4-bit sutractor이다. 위 그림에서 b값이 input으로 쓰일 때는 이전 감산기에서 발생한 빌림수를 빼주는 역할을 하고 output으로 쓰일 때는 다음 감산기로 현재 감산기에서 발생한 빌림수를 전달하는 역할을 한다.

또한 이러한 방식을 사용해 n 비트 크기의 이진수를 계산할 때도 full adder나 full sutractor를 n개 이어 붙여 사용하는 것이 가능하다.

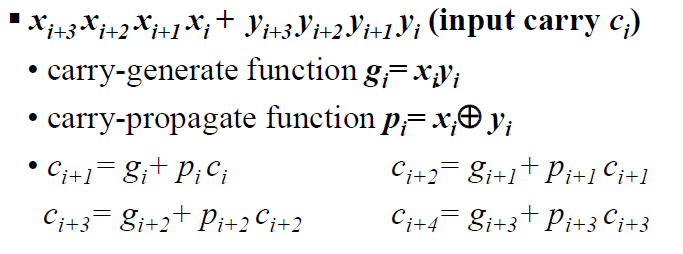
**2.** 앞선 4-bit adder와 같이 adder 병렬 연결된 형태를 ripple carry adder라 부르는데 이러한 ripple carry adder 는 연결된 adder 개수가 늘어날수록 그에 따른 딜레이가 점차 증가한다. 왜나하면 ripple carry adder는 모든 adder들의 연산이 순차적으로 시행되어야 하기 때문이다. 각각의 adder들은 이전 자리에서 발생한 carry값을 받아야 하기 때문에 이전 adder가 끝나기 전까지는 다음 adder를 실행시킬 수 없다.

이러한 문제점을 개선해 각각의 adder가 필요로 하는 carry 값을 따로 계산해 전달해 이전 가산기가 계산이 끝났는지 여부와는 상관없이 모든 adder들이 따로 계산을 수행하게 해 딜레이를 줄인 형태의 회로가 바로 look ahead carry adder 이다.

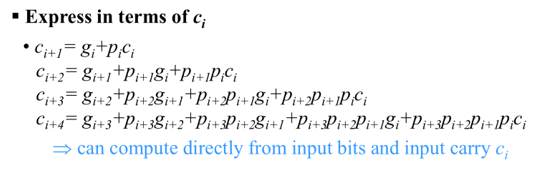


look ahead carry adder는 위와 같이 cla generator를 통해 각각의 adder가 필요로 하는 carry 값을 이전 자리의 가산기를 거치지 않고 바로 전달한다.

다음 식들은 4bit의 look ahead carry adder의 carry 값들을 구하기 위해 식을 풀어 전개한 것이다.



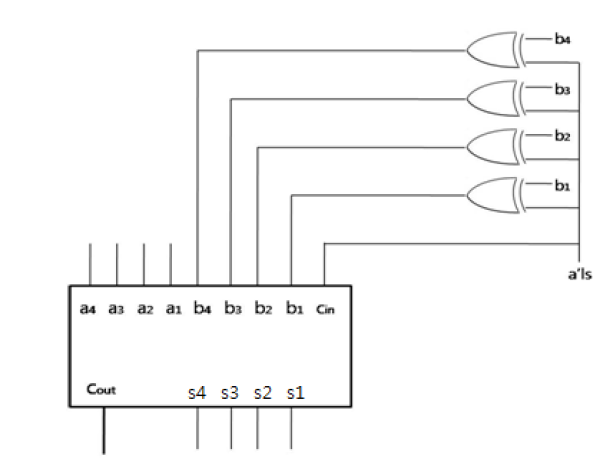
위와 같이 각각의 carry 값들은 이전의 carry이 x와 y값으로 풀어 쓰는 것이 가능하기 때문에 이전의 carry 값들과 상관없이 x와 y만의 식으로 표현하는 것이 가능하다.



즉 위와 같이 x와 y를 활용해 직접적으로 carry값들을 계산하는 것이 가능해진다. 이렇게 계산한 carry 값들은 cla generator가 각각의 adder에 전달하게 된다.

**3.** 앞선 회로들은 감산기 또는 가산기로 덧셈 또는 뺄셈의 역할만을 수행하는 회로였다2’s complement를 이용해 뺄셈을 하는 경우 양수였던 수를 음수로 바꿔 덧셈을 수행하는 방식으로 뺄셈을 수행하게 된다.

즉 2의 보수법을 이용한 뺄셈 방식은 사실상 덧셈 연산이기 때문에 뺄셈과 덧셈을 모두 수행하는 회로를 xor 게이트를 활용해 만들 수 있다. 0과 xor 연산을 하면 원래 식이 그대로 나오고 1과 xor 연산을 하면 식에 not을 취한 값이 나오는 성질을 이용한 것이다.



위 그림은 두 수 a4a3a2a1과 b4b3b2b1에 대해 덧셈 또는 뺄셈을 하는 회로로 s가 덧셈 또는 뺄셈을 할지 결정하게 된다. 만일 s값들이 0이되어 덧셈을 하게 된다면 b값들을 0과 xor 연산을 한다. 이 경우 그대로 b값들이 다시 출력되고 이러한 b값들을 a값과 더해 덧셈 연산을 수행한다.

만약 s값들이 1이 되어 뺄셈을 하게 될 경우 b값들이 1과 xor 연산 한다. 그 결과 b값들에 not을 취한 값이 출력되어 이 값들을 a값들과 더한다면 2의 보수법을 활용한 뺄셈을 수행하게 된다.

**4.** 먼저 bcd란 binary coded decimal로 기존의 2진수를 사람이 인식하기에는 불편하기 때문에 좀 더 편한 형태로 10진수를 2진수로 표현한 것을 의미한다. Bcd code는 보통 8421 code 방식을 의미한다. 이 방식은 0부터 9까지의 수를 4비트의 이진수로 표현한 방식으로 msb부터 8, 4, 2, 1로 가중한 형태로 이 형태는 기존에 일반적으로 쓰던 10진수를 2진수로 변환한 형태와 비슷하다.   
 기존의 10진수를 2진수로 변환하는 법과의 차이점은 10진수의 전체 값을 2진수 형태로 바꾸는 것과 달리 0~9까지의 각 자릿수를 2진수 형태로 변환한다는 점에 있다.

예를 들어 숫자 12일 경우 0001 0010 와 같은 형태를 가지게 된다.

이러한 점 bcd의 특징 때문에 bcd연산의 경우 9를 넘을 경우 기존에 사용하던 덧셈 방식으로는 그 값을 구할 수가 없다.

예를 들어 6+5를 하는 경우 0110 +0101 = 1011 이다. 이 경우 1011은 11로 bcd code에서 정의하지 않은 수다. 이 경우 0110을 더하면 0001 0001 로 11이 원하는 bcd code 형태로 나타나는 것을 확인할 수 있다.

두 자릿수의 덧셈에 대해서도 각각의 자릿수를 그대로 더한 후 0110을 더하면 원하는결과를 얻을 수 있다. 만일 첫 번째 자릿수 즉 오른쪽 4개의 비트에서 자리올림 값이 발생했다면 다음 자리의 값, 즉 그 왼쪽 4개 비트 값으로 자리올림값인 1을 옮겨주면 된다.

즉 4bit 이진수의 결과가 1011, 즉 10이상의 결과가 나온다면 결과에 0110, 즉 6을 더하면 원하는 덧셈의 결과를 bcd 형태로 얻을 수 있다.

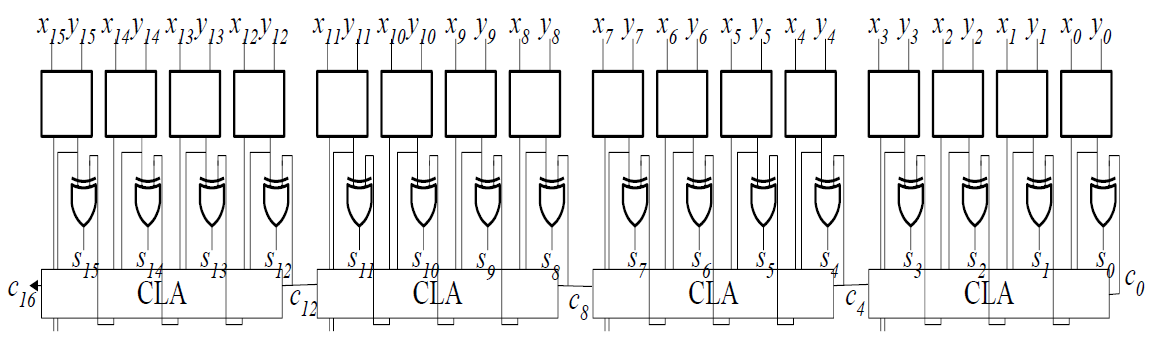
**5.** ALU는 Arithmetic Logic Unit)로 컴퓨터의 덧셈이나 뺄셈, 비교연산과 같은 산술 연산이나 AND, OR과 같은 논리 연산을 수행해주는 장치다. 이러한 ALU는 CPU의 구성요소 중 하나로 쓰이며 CPU 내에서 연산을 담당하게 된다.

ALU는 가산기, 레지스터, 보수기, 시프터, 오버플로우 검출기 등으로 구성되어 있으며

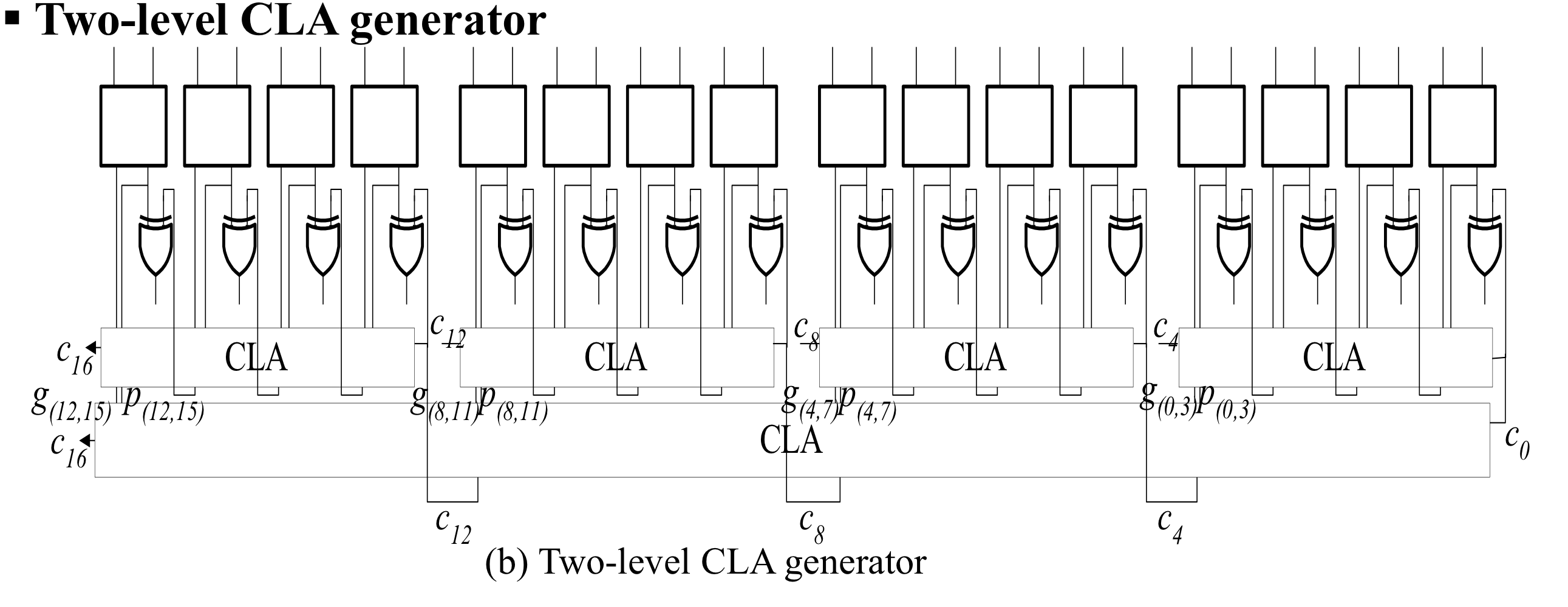
이 중 가산기는 덧셈을 행하고 보수기는 2의 보수를 형성해주고 시프터는 2진수 수에 대해 shift 연산을 수행해 곱셈을 수행하게 해준다. 그리고 오버플로우 검출기는 결과값이 레지스터의 용량을 초과하면 이를 알려주는 회로다.

또한 alu는 메모리 장치인 레지스터와 연결되어 있는데 레지스터는 alu가 사용하는 입력 값과 alu의 연산 결과를 임시로 저장하기 위해 사용한다.

**6.**



위 회로는 16 비트 크기의 carry look ahead adder 이다. 위 회로는 앞서 살펴본 4-bit carry look ahead adder를 다시 4개씩 이어 만든 회로로 8비트 크기의 수들의 가산을 ripple carry adder보다 효율적으로 계산이 가능하다.



위 회로는 한 단계 더 딜레이를 줄이기 위해 개선한 2-level carry look ahead adder의 회그림이다. 위 회로는 cla generator를 이중으로 집어넣어 c12, c8,c4 값들은 2 level cla generator가 계산하게 만들어 그 딜레이를 줄게 만들었다.