6주차 예비보고서

전공: 심리학과 학년: 3학년 학번: 20190345 이름: 김동현

1.

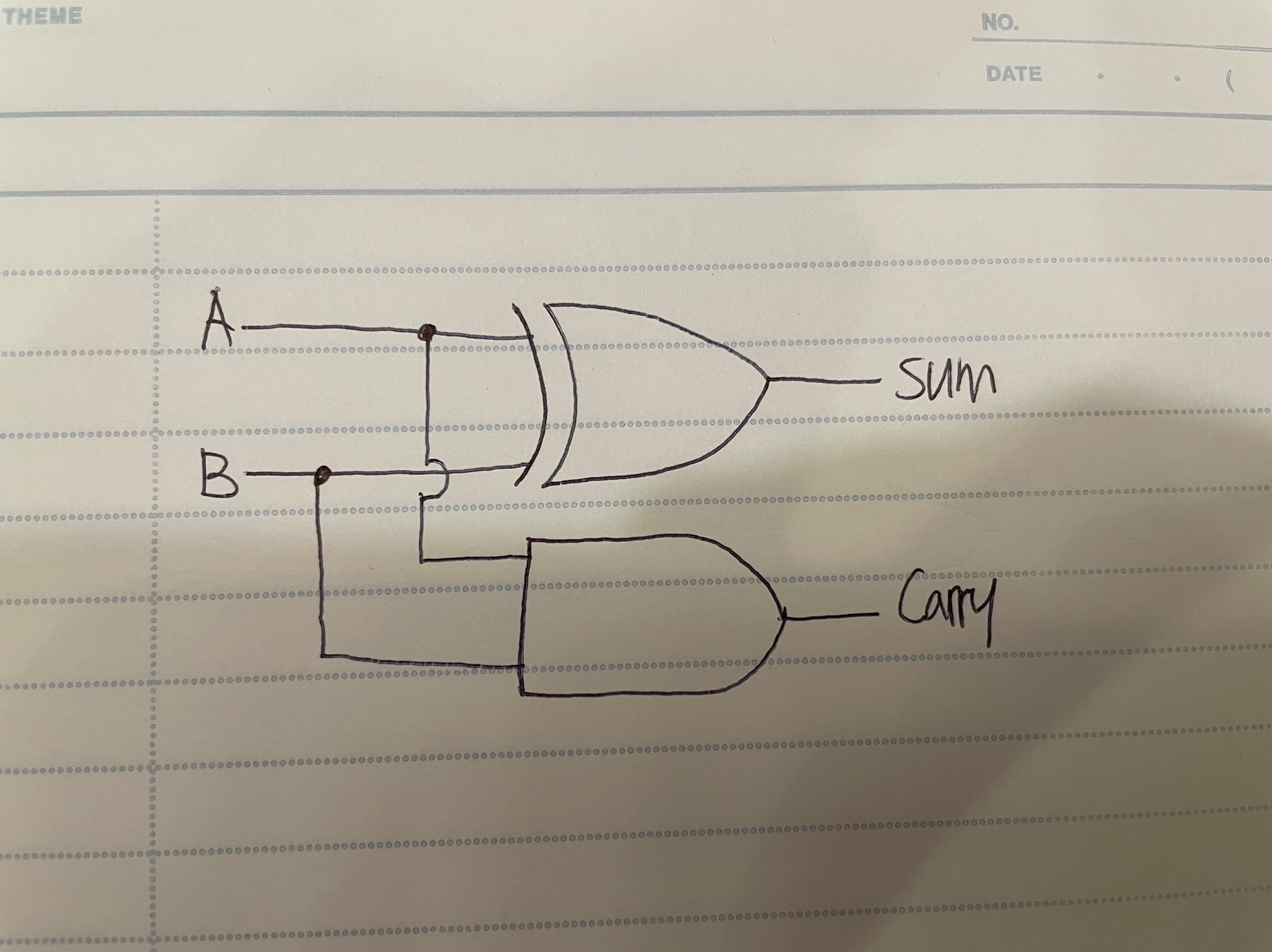
가산기는 1비트 수의 덧셈을 수행하는 논리회로로, 반 가산기와 전 가산기가 있다.

반 가산기는 half-adder로, 2개의 입력과 2개의 출력으로 구성된다. 입력 2개는 1bit 값 2개가 입력으로 들어오며, 2개의 비트는 더하고자 하는 피연산자이다. 출력 2개는 Sum(s)과 Carry(c)로 Sum은 입력의 합을, carry는 1비트로 표현할 수 있는 범위를 넘어가는 경우를 나타낸다.

테이블이(가) 표시된 사진

자동 생성된 설명

위는 반가산기의 진리표이다. 두 입력이 모두 0인 경우에는 sum이 0으로 나오며, 범위를 넘어가는 값이 없으므로 carry값도 0이다. 두 입력 중 하나만 1인 경우는 sum이 1이 되며, 이때도 범위를 넘어가는 값이 없으므로, carry값은 0이다. 두 입력이 모두 1인 경우 값이 이진수로 10이 나오게 되는데, 이때 범위를 넘어가는 값이 있으므로 carry값은 1이 되고, sum은 0이 된다. 이를 논리식으로 나타내면, sum은 1의 개수가 홀수일 때 1을 반환하고 1의 개수가 짝수일 때 0을 반환하므로 xor연산 사용한다. Carry는 두 입력이 모두 1인 경우에 1을, 나머지 경우에는 0을 반환하므로 and연산을 사용한다. 이를 통해 Sum=A^B, Carry=A&B로 식을 나타낼 수 있다.



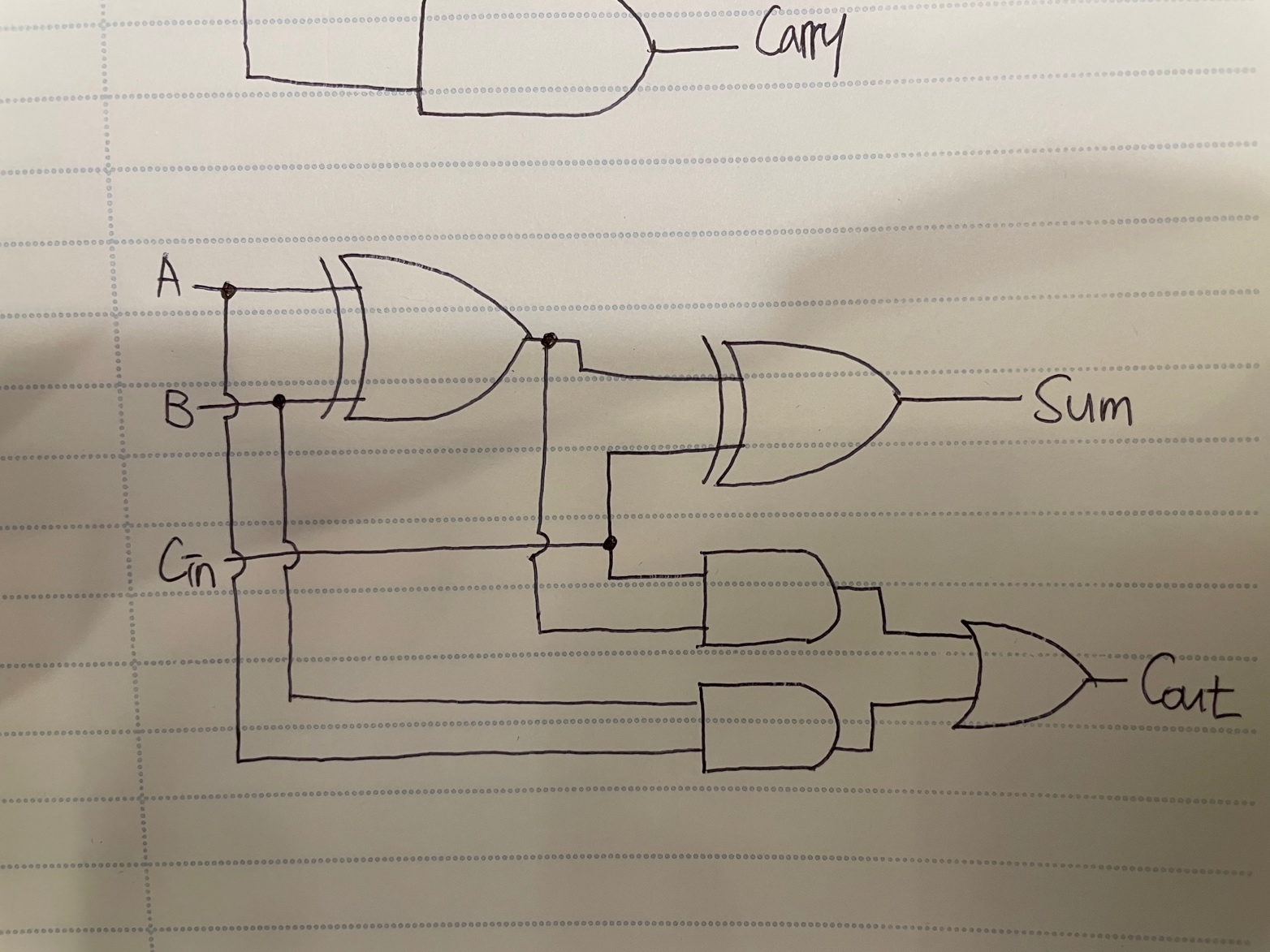
위는 반가산기의 회로도를 표현한 그림이다.

전 가산기는 Full-Adder로, 3개의 입력과 2개의 출력으로 구성된다. 3개의 입력은 2개의 1비트 입력과 1개의 하위 carry값으로 구성되며, 2개의 출력은 입력값의 합 Sum과 표현범위를 넘어가는 수를 표현하는 Carry로 구성된다.

테이블이(가) 표시된 사진

자동 생성된 설명

위는 전 가산기의 진리표이다. 3개의 입력 중 1의 개수가 0개인 경우 sum과 carry값이 모두 0이 나오며, 1의 개수가 1개인 경우 sum은 1을, carry는 0을 반환한다. 1의 개수가 2개인 경우 sum은 0을 carry는 1을 반환하며, 모든 입력이 1인 경우 sum과 carry가 모두 1임을 알 수 있다. 이를 통해 논리식으로 나타낸다면, sum=A^B^Cin, carry=Cin&(A^B)+A&B으로 나타낼 수 있다.



위는 전 가산기를 회로도로 표현한 그림이다. Xor연산과 and연산이 2개(반 가산기 2개)와 or연산 1개로 이루어짐을 알 수 있다.

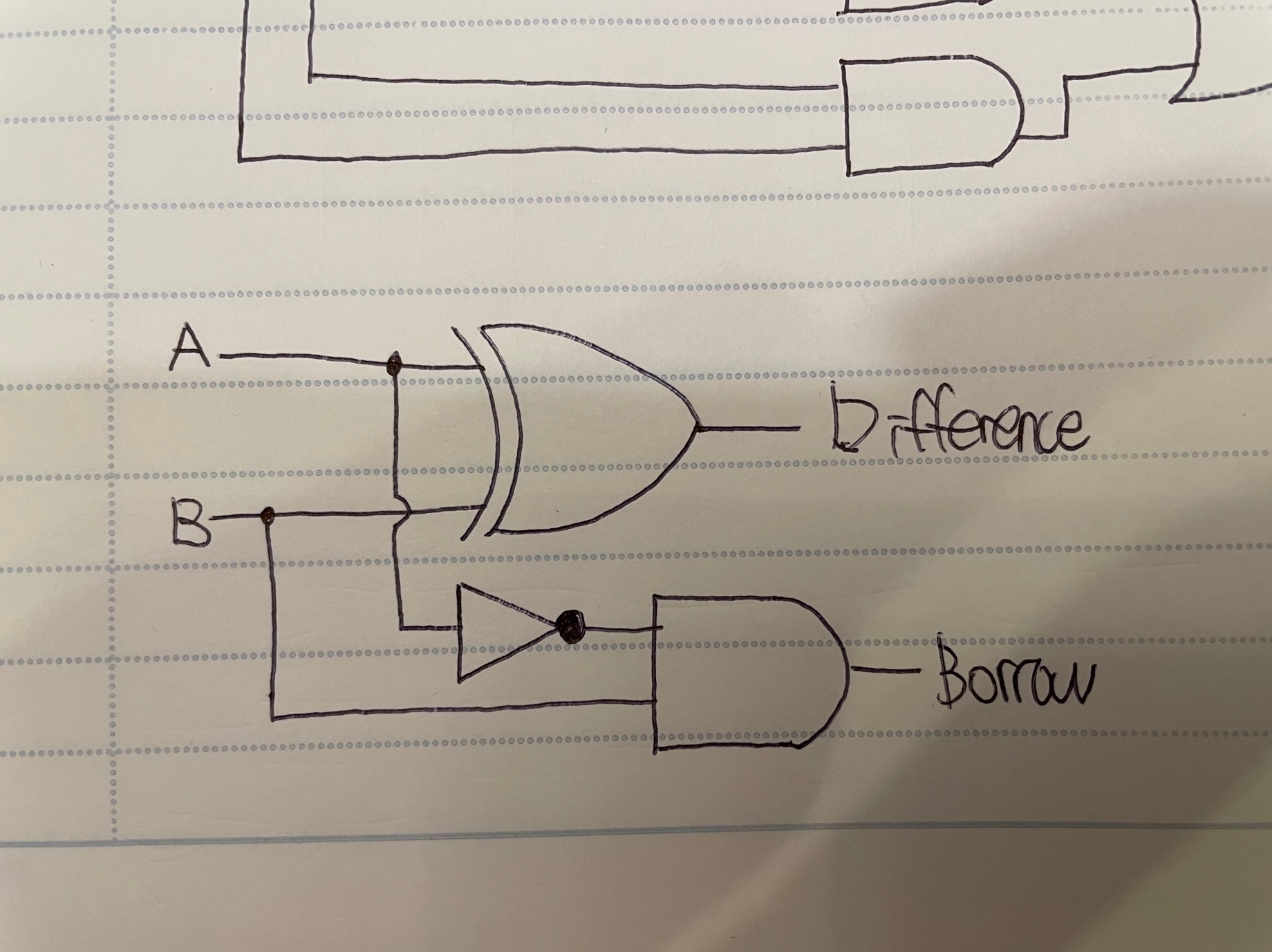
2. 전 감산기 및 반 감산기에 대해 조사(예시 포함)

감산기는 1비트 수의 뺄셈을 수행하는 논리회로로, 반 감산기와 전 감산기가 있다.

반 감산기는 half-subtractor로 2개의 입력과 2개의 출력으로 구성된다. 2개의 입력은 1비트 값 2개 a-b에서 a에 해당하는 피감수값과 b에 해당하는 감수값으로 구성되며, 출력은 a-b값에 해당하는 Difference(D)와 다음 자리수로부터 빌려오는지의 여부에 대해 나타내는 Borrow(B)로 구성된다.



위는 반 감산기의 진리표이다. A-B의 연산을 한 결과로 입력이 모두 0이면 0-0=0이므로, difference와 borrow 모두 0이 된다. A=0,b=1인 경우 0-1=-1이므로, 뺄셈으로 인해 범위를 벗어나므로 다음 자리수로부터 빌려와야 한다. 따라서 borrow는 1이 되며, difference 또한 1이 된다. A=1,b=0인 경우 1-0=1이므로, 다음 자리수로부터 빌려올 필요가 없어 borrow는 0이 되고, difference는 1이 된다. 마지막으로 a=1,b=1인 경우 1-1=0이므로, borrow와 difference모두 0이 된다. 이를 논리식으로 표현한다면, difference=A^B, borrow=(~A)&B로 나타낼 수있다.



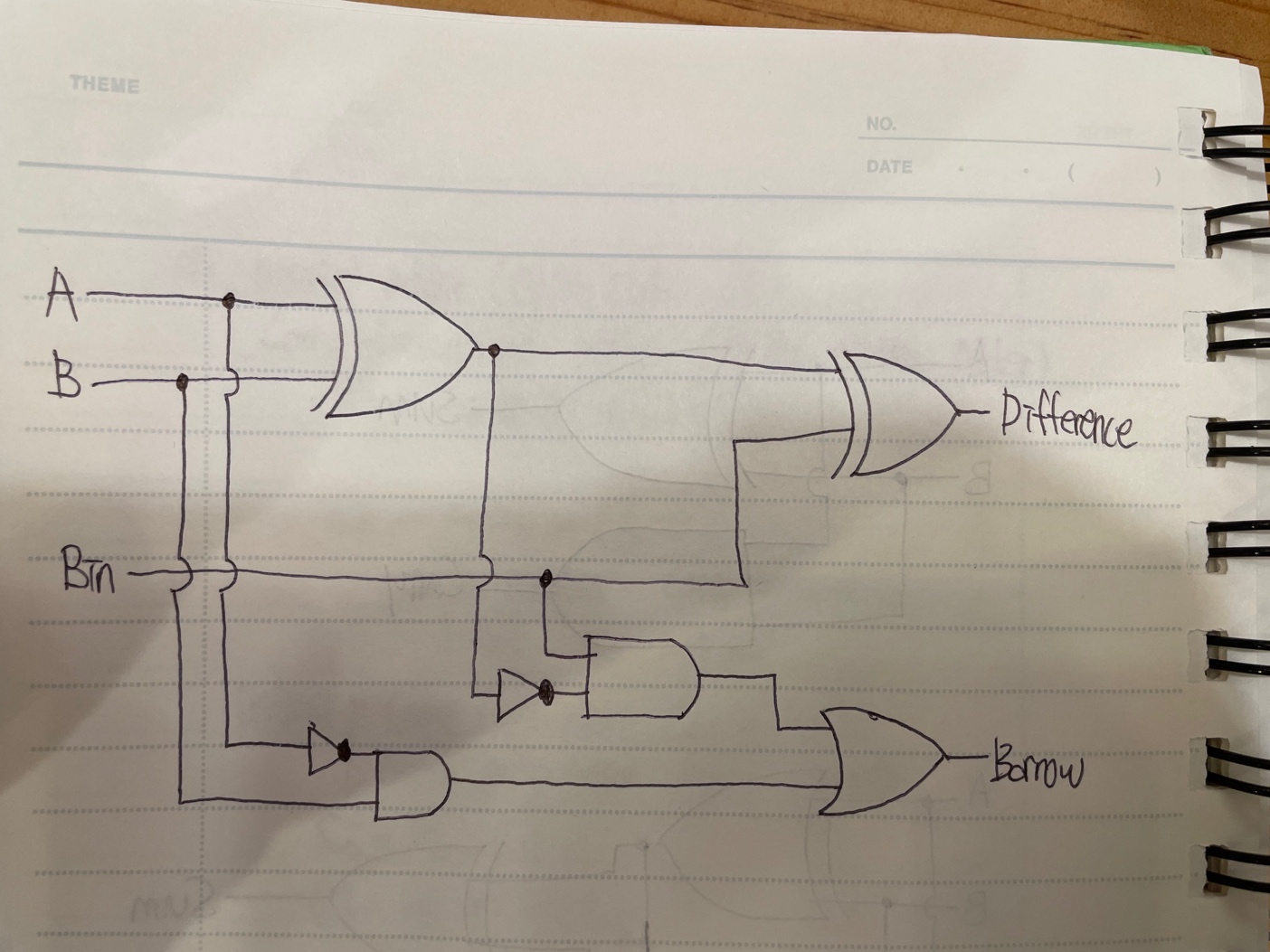
위 그림은 반감산기를 회로도로 표현한 것이다. Xor연산, NOT연산, AND연산이 사용되었다.

전감산기는 Full-Subtractor로, 3개의 입력과 2개의 출력으로 구성된다. 3개의 입력은 3개의 입력은 피감수값 A, 감수값 B, 이전의 연산이 A로부터 값을 빌려갔는지에 대해 나타내는 Bin으로 구성되며, 2개의 출력은 a-b값에 해당하는 Difference와 다음 자리수로부터 값을 빌려오는지 여부를 나타내는 Borrow로 구성된다.

테이블이(가) 표시된 사진

자동 생성된 설명

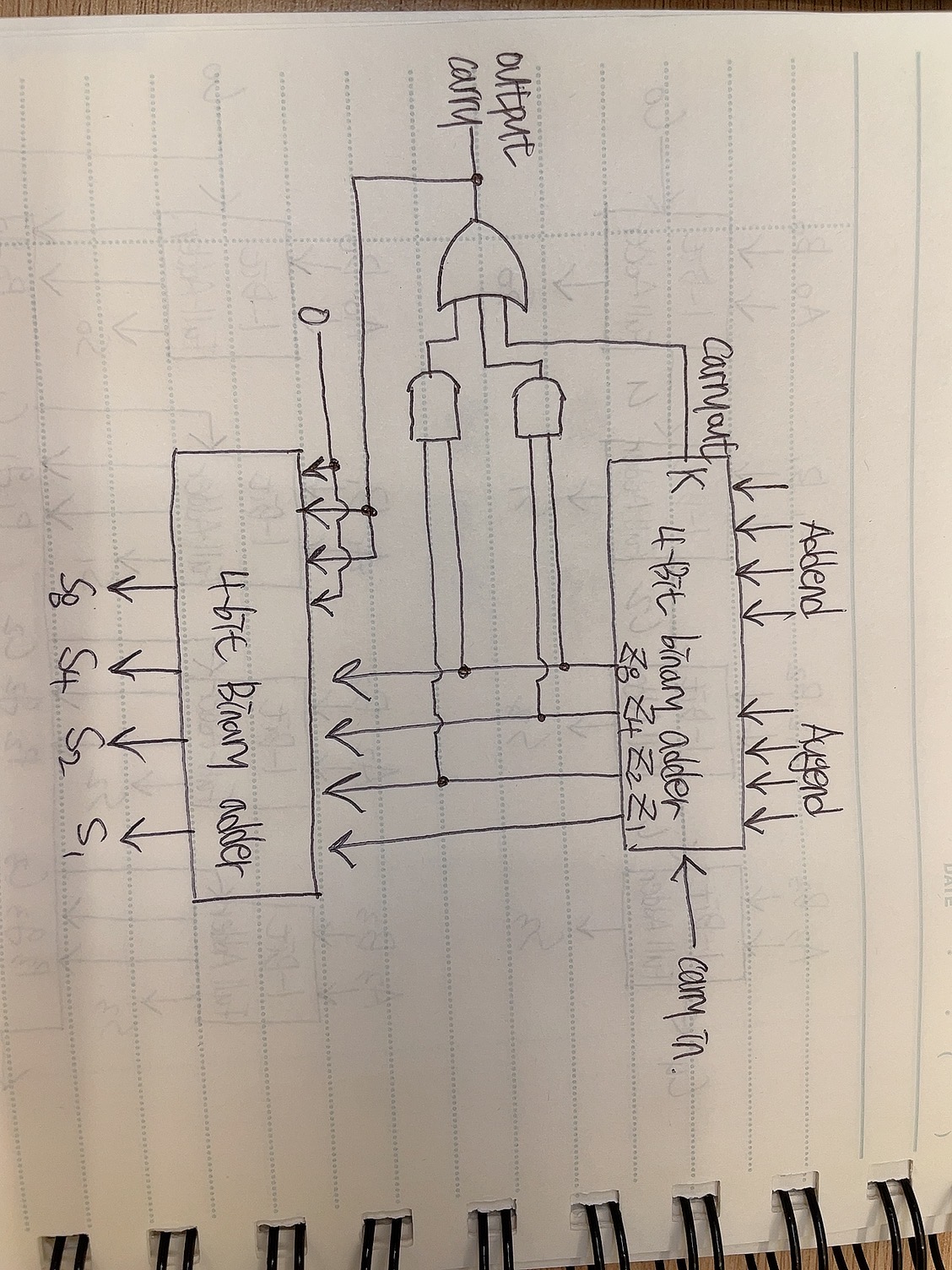
위는 전 감산기를 진리표로 나타낸 것이다. 이를 논리식으로 나타내면, Difference=A^B^Bin, Bout=(~A)&Bin | (~A)&B | B&Bin으로 나타낸다.



위 그림은 전 감산기를 회로도로 표현한 것이다. 2개의 xor,and,not연산(2개의 반 감산기)과 1개의 or 연산이 사용되었다.

3.

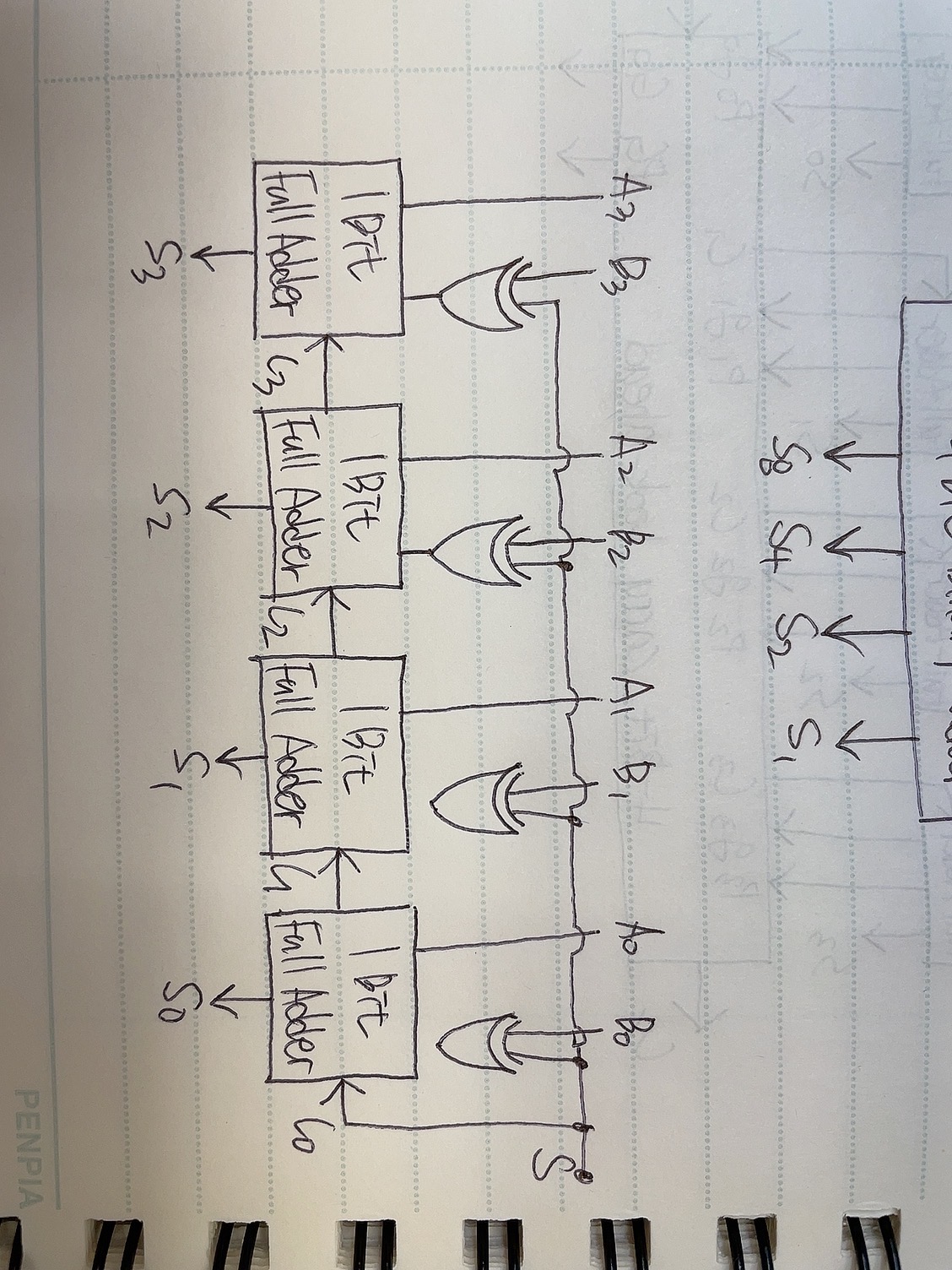
BCD가산기는 BCD 코드로 표현된 2개의 값과 이전의 carry 값 총 3개 입력을 더하여 BCD코드로 반환하는 가산기이다. BCD코드의 특징이 0부터 9까지의 숫자를 4개의 비트를 사용하여 나타낸다는 점이다. 즉 0000부터 1001까지의 코드를 사용하고, 1010부터 1111까지의 코드는 사용하지 않는다. 연산 과정은 두개의 BCD코드와 이전 carry값을 더한다. 이때, 가산의 결과가 10미만인 경우 그대로 결과값을 반환한다. 만약 10이상이 결과값이라면 6(0110)을 더하여 반환한다.



위는 BCD 가산기의 회로도이다.

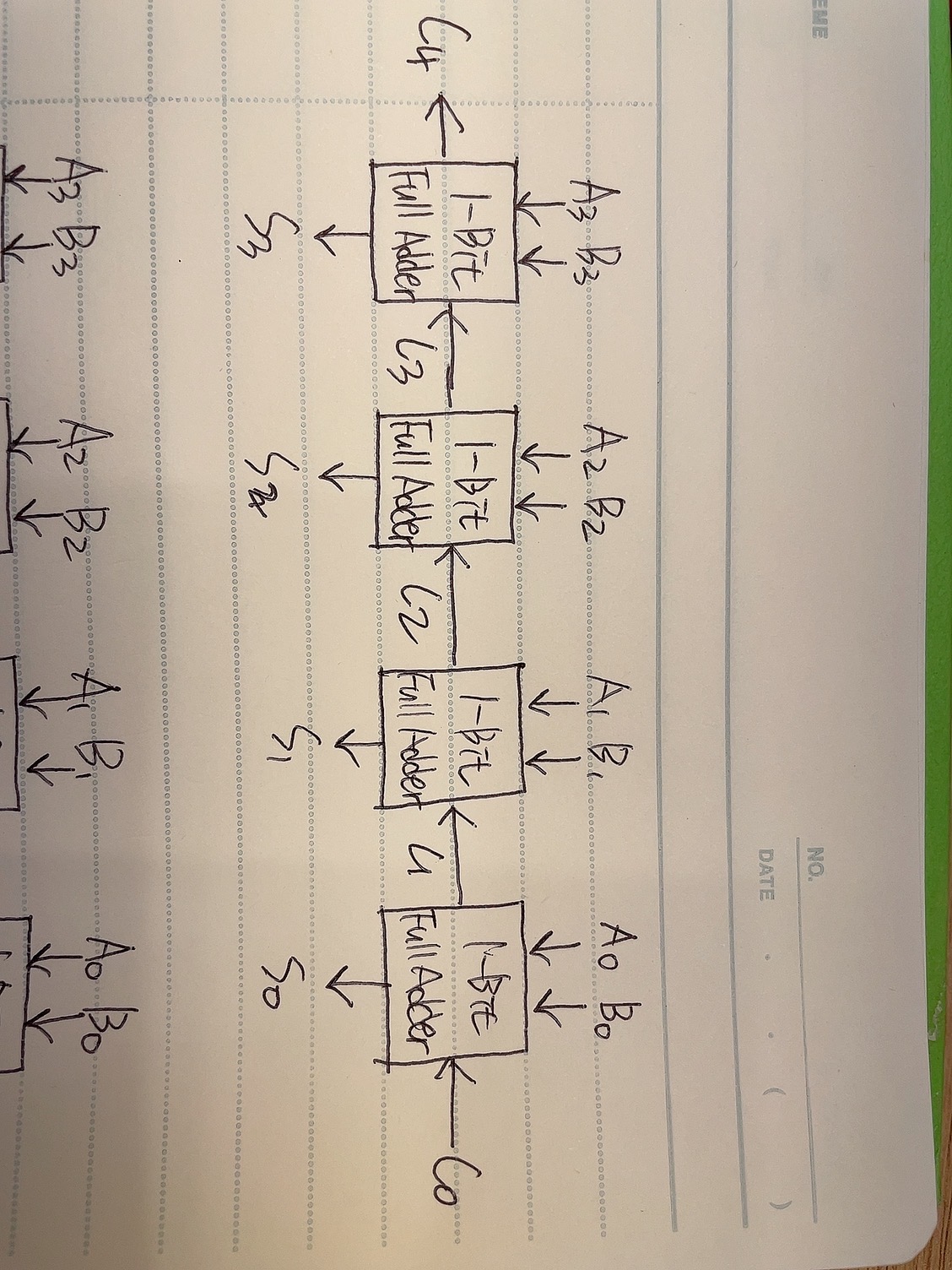
4.

병렬 가감산기는 제어신호에 따라 가산과 감산을 하는 회로이다.



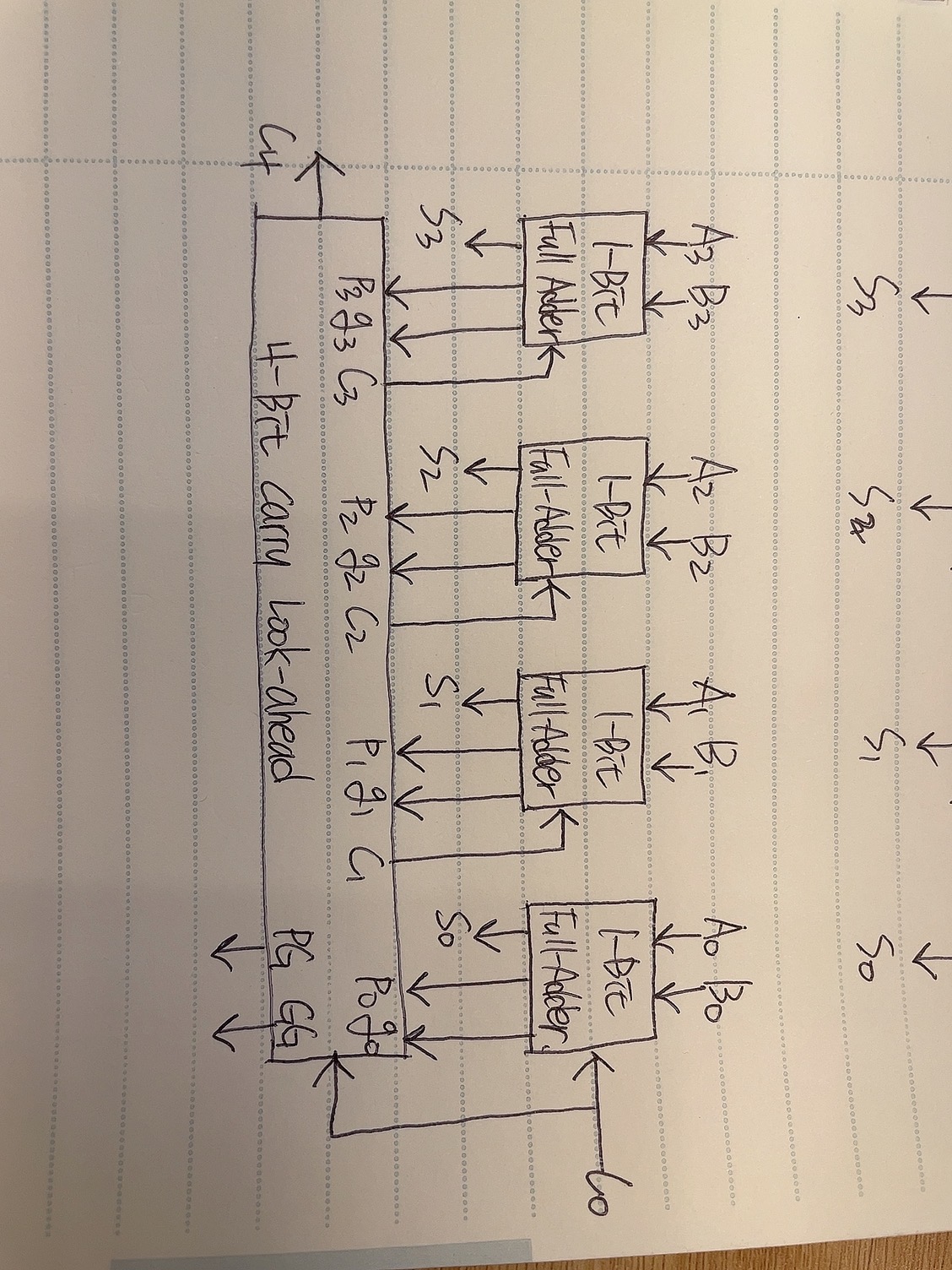
위 그림은 4비트의 병렬 가감산기의 회로도이다. S기호의 신호를 통해 가산과 감산을 구분한다. 0이면 가산을 1이면 감산을 실행하는데, 이때 두번쨰 피연산자항 b와 xor연산을 통해 덧셈과 뺼셈에 맞는 상태로 신호가 변한다. 뺄셈의 경우 c0의 carry에도 s 신호 1이 더해져, 2의 보수 연산을 가능케하도록 한다.

5.

Ripple carry adder는 여러 개의 전 가산기가 각각 입력을 받고 각각 출력을 하고, 자리올림수가 다음 전 가산기로 옮겨가는 과정을 보인다. 여러 자리의 연산을 수행할 수 있다는 장점이 있으나, 전 가산기의 입력 중에 이전 전 가산기의 자리 올림수가 입력으로 필요하기 때문에, 이전 전 가산기의 연산을 모두 기다려야 한다는 단점이 있다. 이로 인해 계산해야할 비트 수가 증가하면 연산 시간이 증가한다. 

위는 4비트 연산을 위한 ripple carry adder의 회로도이다. C1이 다음 전 가산기의 입력으로 들어가고, 이 전 가산기에서의 carry값 c2가 다음 전 가산기의 입력이 되는 모습을 알 수 있다.

Carry look ahead adder는 자리 올림 수인 carry값을 예측을 통해 한 번에 연산한다. 이는 이전의 전 가산기의 연산을 기다려야 하는 ripple carry adder의 단점을 보완한 것이다. 이로 인해 carry look ahead adder는 연산 시간을 줄일 수 있다는 장점을 보이고 있다. 하지만 이로 인해 carry 값을 따로 계산하는 회로를 만들어야하고, 그 회로가 비트 수가 늘어날 수록 복잡해 비용과 전력이 증가한다는 점은 ripple carry adder와 비교하면 단점으로 드러난다.



위 그림은 4비트 연산을 위한 carry look ahead adder의 회로도이다. Carry 값 연산을 위해 새로운 회로를 사용하는 모습을 볼 수 있다.

6.

Carry look ahead adder에서는 자리 올림수인 carry값을 예측하는 연산을 사용한다. 이는 입력에 대해 G값과 P값을 구하여야 한다. G값은 carry generation으로 자리 올림수 생성값이다. 입력값의 and연산으로 알 수 있다. P값은 carry proparation으로 자리 올림수 전달을 나타내는 값이다. 입력값의 xor 연산으로 알 수 있다.

7. 참고문헌

컴퓨터공학설계2 교재

Alan B. Marcvitz, introduction to logic design, McGraw-Hill (2010).

A. Bayrakci and A. Akkas, "Reduced Delay BCD Adder," 2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP), 2007, pp. 266-271.