**10주차 예비보고서**

**전공: 생명과학과 학년: 4학년 학번: 20182186 이름: 김승원**

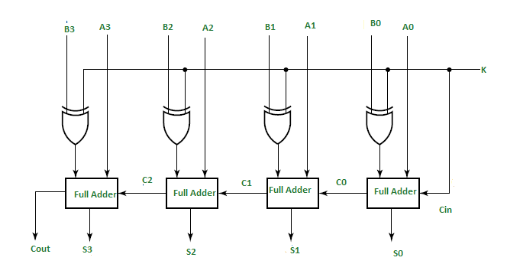
**1. 4-bit adder 및 subtractor 이진 병렬 연산기능 조사**

이진 병렬 가산기는 조합 회로만을 사용해 두 이진수의 덧셈과 뺄셈을 구하고자 하는 회로다. 병렬 가산기는 n개의 연결된 adder사용해서 n비트 이진수 2개를 더할 수 있다. 4-bit adder 및 subtractor는 ALU 중 하나이기도 하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

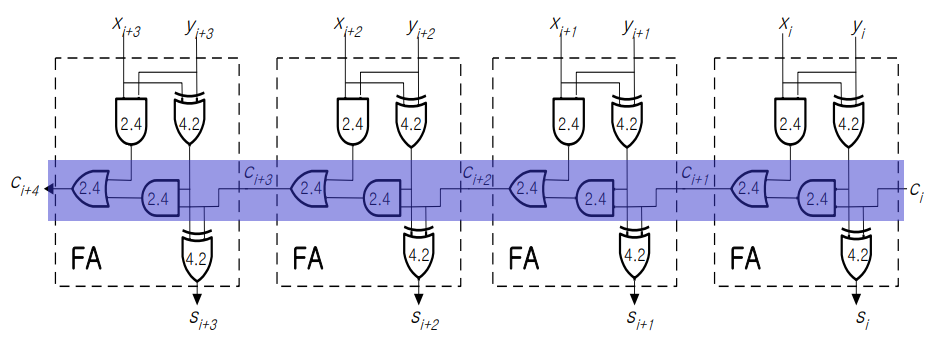
**<Figure 1. 4bit 이진수 계산 법>**



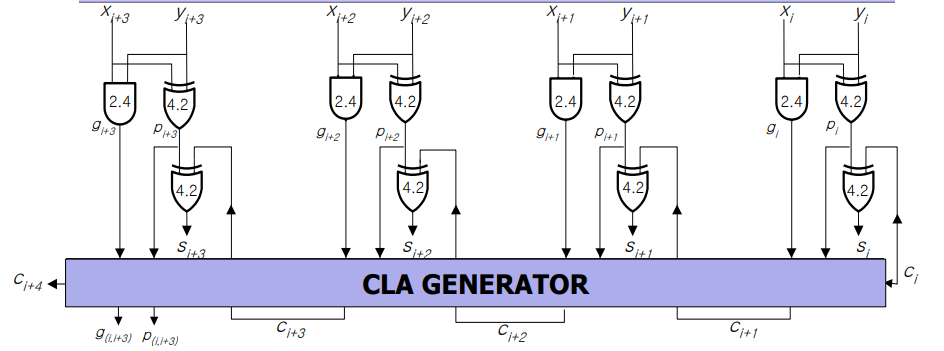
**<Figure 2. 4-bit full adder >**

먼저 **Figure 2**를 확인하면, A, B입력과 control line인 K를 확인할 수 있다. A0는 전가산기에 직접입력 되는 형태이고, B0와 K는 XOR gate를 거쳐서 Adder에 입력된다. 이때, K 값에 따라 덧셈을 할지 뺄셈을 할지 결정되는데, K가 0인 경우에는 B0 (XOR) K = B0로, 4개의 bit수에 대한 수행되는 연산이 덧셈이 되고, K가 1인경우에는 B0 (XOR) K = B0’ 로 B0의 complement 형태가 출력되면서 연산이 A+B0’가 된다. 따라서 K값이 1이면 4개의 bit 수에 대하여 수행되는 연산은 뺄셈이다. 그런 다음 C0와 S0를 보면, C0는 두번째의 Full adder에 직렬로 전달되고, S0는 덧셈이나 뺄셈을 한 결과로 최하위 비트로 기록된다. A1과 A2, A3는 각각 2~4번째 Full adder에 입력되고, 마찬가지로 B1~3과 K에 따라 각각의 bit가 결정된다. C0~2 또한 각각의 full adder에 입력되어 전달되고, C3는 모든 덧셈과 뺄셈에 대한 캐리가 된다. S0와 마찬가지로 S1~3까지도 기록되어 최종 결과를 확인할 수 있다.

**2. Look ahead carry에 대해서 조사**

****

**<Figure 3. Ripple carry adder>**

****

**<Figure 4. Carry look-ahead adder>**

Look ahead carry는 병렬 adder에서 각 자리의 가산기의 경우 한 자리 아래의 adder로부터 carry를 받아야 수행할 수 있다. 따라서 carry에 의해서 delay가 발생하게 되는데, 이러한 delay를 줄이고자 각 자리의 carry를 미리 계산하는 방법을 사용한다. Carry를 미리 계산해서 사용하여 모든 자리를 동시에 계산할 수 있어 delay를 줄인다.

**Figure 3**는 ripple carry adder이다. 각각의 gate를 지나거나 연산을 진행하면서 gate내에 delay가 항상 발생하게 된다. Ripple carry adder에서는 보라색영역에 의해서 delay가 상당히 많이 발생하게 된다. 총 4bit이니까 1개의 and gate와 1개의 or gate를 4번은 반드시 거치게 된다. 이러한 단점을 보완하기 위해 고안된 것이 **figure 4**의 Carry look-ahead adder이다. 하나씩 계산해서 아래자리에서부터 끝까지 연산 되어져 오는 것을 기다렸으나 Carry look-ahead의 경우에는 generator에 넣어서 바로 계산되어 carry output을 상당히 빠르게 계산할 수 있다. 자세한 계산 방법은 기타이론(ripple carry adder와 carry look-ahead adder의 비교)에서 다룬다.

• carry-generate function gi = xi yi

• carry-propagate function pi = xi ⊕ yi

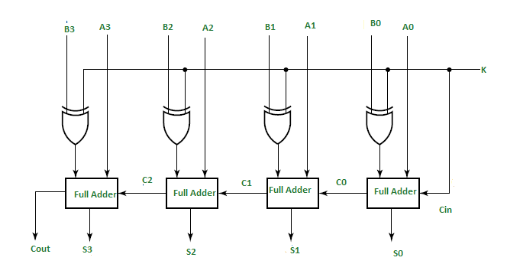
ci+1= gi+pi ci

ci+2=gi+1+pi+1gi+pi+1pici

ci+3=gi+2+pi+2gi+1+pi+2pi+1gi+pi+2pi+1pici

ci+4= gi+3+pi+3gi+2+pi+3pi+2gi+1+pi+3pi+2pi+1gi+pi+3pi+2pi+1pici

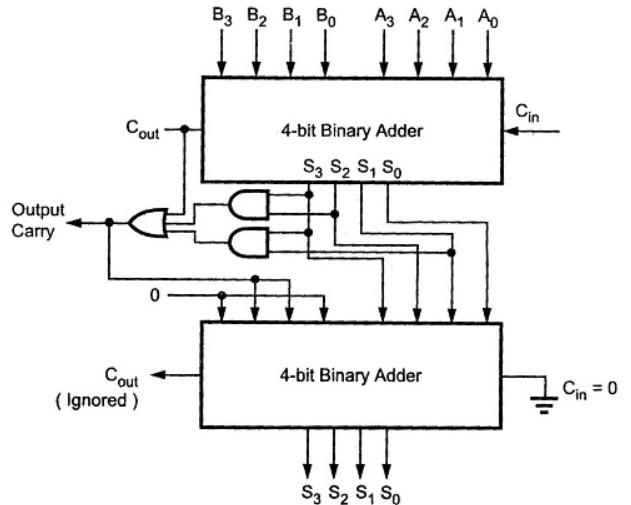
**3. XOR을 활용한 2’s complement 가감산에 대해 조사**



**<Figure 2. 4-bit full adder>**

XOR gate를 활용할 시 두입력이 다른 경우 1을 출력하고 같으면 0을 출력한다. **Figure 2**와 같이 첫번째 전가산기의 경우 control line을 직접 입력하고 입력 A0는 전가산기에 직접 입력된다. 그리고 B0와 K가 XOR gate로 입력되는데, 모든 입력으로부터 출력은 S0와 C0로 출력된다. K(control line)의 값이 만약 1이면 B0 (XOR) K = B0’(complement B0) 로 출력되고, 연산이 A+ B0’로 연산 된다. 그리고 두 숫자 A와 B에 대한 2의 보수 뺄셈은 A+B’으로 표시할 수 있다. 즉, K에 1이 입력되면 4개의 bit 수에 대해서 수행되는 연산이 뺄셈을 의미하는 것이다. 반대로 K에 0이 입력되면 B0로 (B0 (XOR) K = B0) 그대로 출력되어 4개의 bit에 대해 수행되는 연산은 덧셈을 의미한다.

**4. BCD 연산에 대해서 조사**

****

**<Figure 5. Block diagram of BCD adder>**

Digital system은 10진수를 이진 코드화 된 BCD형태로 처리한다. 두 개의 BCD number를 더해서 BCD에서도 합계를 생성해주는 BCD adder이다. 각 BCD 숫자는 4bit 이진수로 표시되고, 526을 BCD number로 표시하게 되면, 0101 0010 0110으로 표시할 수 있다.

이때 BCD는 9보다 클 수 없다. 즉, 3과 6을 더하면 다음과 같다.

0110 (3)

+ 0011 (6)

= 1001 (9)

덧셈은 일반 이진 덧셈처럼 수행된다. 하지만 캐리가 발생했을 때 문제가 생긴다.

0110 (6)

+ 1000 (8)

= 1110 (14) -> invalid BCD number

1110은 BCD number에 포함될 수 없다. 두 자리의 합이 9를 넘었기 때문이다. 따라서 이 문제를 해결하기 위해서 BCD number에 0110(6)을 추가해서 합계를 수정한다.

0110 (6)

+ 1000 (8)

= 1110 (14)

0110 (6)

0001 0100 (14)

위와 같이 4bit씩 묶어서 1과 4를 표현해서 BCD 연산을 진행해야 한다.

즉, 이진수를 더해 두 개의 BCD number를 추가하고, 4bit합계가 9보다 작다면 따로 수정할 필요 없지만, 만약 9가 넘어 carry가 발생한다면 수정작업이 필요한 것이다. 0110을 추가해서 적절한 BCD number가 될 수 있도록 한다.

BCD adder를 구현하기 위해서는 **Figure 5**에서 확인할 수 있듯이, 4bit adder와 9보다 큰 합을 검출할 수 있는 회로, 0110을 추가시켜줄 수 있는 4bit adder로 구성되는 것이다.

**5. ALU의 기능에 대해서 조사**

Arithmetic logic Unit(ALU)는 arithmetic과 logic 연산을 할 수 있는 회로다. 일부의 processor에서는 ALU는 Arithmetic unit과 logic unit의 두 단위로 나뉘기도 하고, 둘 이상의 Arithmetic unit이 포함되어 있기도 하다.

일반적으로 ALU는 process controller와 main memory 및 input/output device에 bus를 통해서 직접 입출력 접근이 가능하다. 대부분 CPU 연산은 ALU들에 의해 수행된다. ALU에 의해서 처리되어질 데이터들은 레지스터로부터 ALU로 입력된다. 연산코드와 명령어 그리고 피연산자들이 입력에 포함되고, ALU는 연산코드를 확인하여 어떠한 연산인지 파악한 후에 수행하고 결과를 출력 레지스터 중 하나로 저장해준다.

ALU의 각 내부 구성요소들은 다음 과 같은 기능을 수행한다.

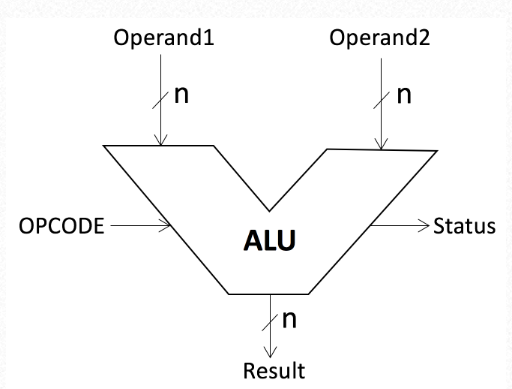
- 산술 연산장치: +, -, \*, /의 사칙연산 수행

- 논리 연산장치: AND, OR, XOR, NOT 논리 연산 수행

- 이동 레지스터: 비트를 오른쪽이나 왼쪽으로 이동시키는 기능을 수행

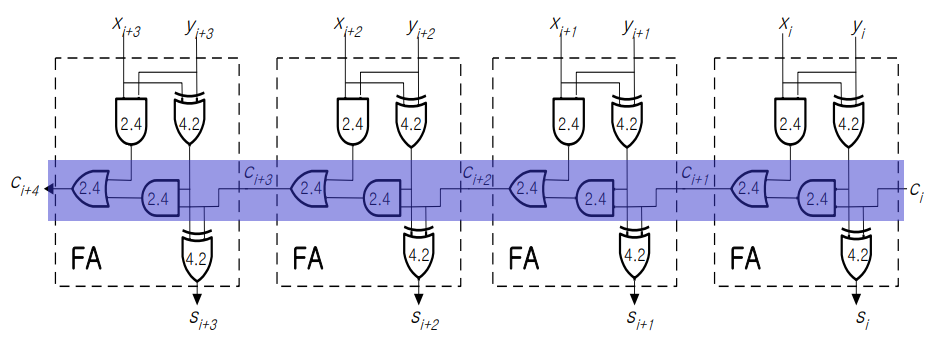
- 보수기: 이진 data의 보수를 취하는 회로

- 상태 레지스터: 연산 결과 상태를 나타내어주는 플래그들을 저장

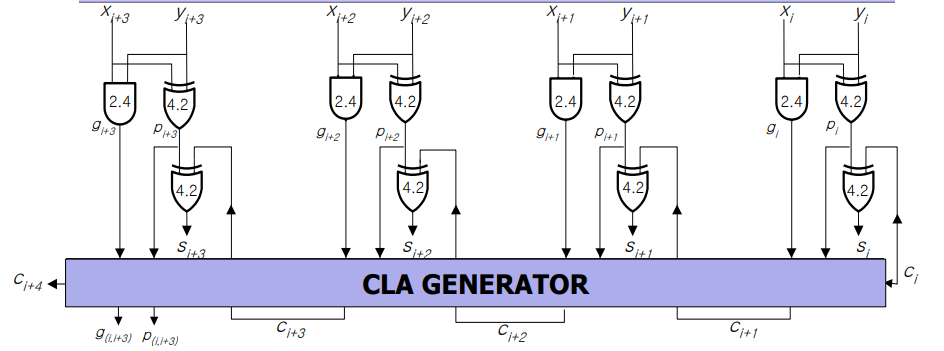
****

**<Figure 6. A symbolic representation of an ALU>**

**6. 기타이론 – Ripple carry adder와 carry look-ahead의 delay 비교**

****

**<Figure 3. Ripple carry adder>**

****

**<Figure 4. Carry look-ahead adder>**

Ripple carry adder와 carry look-ahead adder의 gate delay를 비교했을 때, carry look-ahead adder가 현저히 빠르다. 가령, gate 하나당 1Δ만큼의 delay를 가진다고 했을 때, ripple carry adder의 경우 x, y input으로 들어오면서 1Δ, 그리고 Carry 계산에서 OR gate, AND gate 하나로 총 4번이니까 8Δ 그리고 S output에서 XOR gate에 의해 1Δ로 총 10Δ 만큼의 delay가 생기게 된다. 하지만 carry look-ahead adder의 경우 CLA generator에서 carry 계산 부분의 OR gate와 AND gate 연산을 한 번만 진행해서 8Δ에서 2Δ로 줄어들어서 x, y input에서부터 s output까지 총 4Δ만큼의 delay밖에 생기지 않는다. 이로써 carry look-ahead adder가 현저히 빠르게 계산될 수 있음을 알 수 있다.