

제목 : 소프트웨어 공부를 시작할 때 함께 알았으면 좋았을 것들.

부제 : 소프트웨어를 지탱하는 기술은 무엇인가

안녕하세요 멘토 홈런입니다.

오늘은 문득 떠오르는 이야기가 있어 여러분들께 소개해보려고 합니다.

우리는 수많은 문제 해결을 위해 수많은 소프트웨어를 만들고 있습니다. 대부분의 제작 과정은 'Code'라고 부르는 논리 문서를 작성한 뒤 컴파일을 수행하여 완성하는 순서로 되어있죠. 이 때 우리는 별다른 시스템에 대한 큰 고민을 하지 않아도 코드를 통해 훌륭한 소프트웨어를 만들어 낼 수 있습니다. 그렇다면 과연 무엇이 있기 때문에 코드라는 것은 작성하여 우리가 원하는 기능을 만들 수 있는 것일까요? 가장 근원부터 찾아보도록 하겠습니다.

## 1. Transistor(트랜지스터)

최초의 시작은 트랜지스터입니다. 트랜지스터는 전기적 신호 조절에 따라서 다른 전기 흐름의 On/Off를 결정할 수 있는 **소자(Element)**입니다. 표로 나타내면 다음과 같습니다.

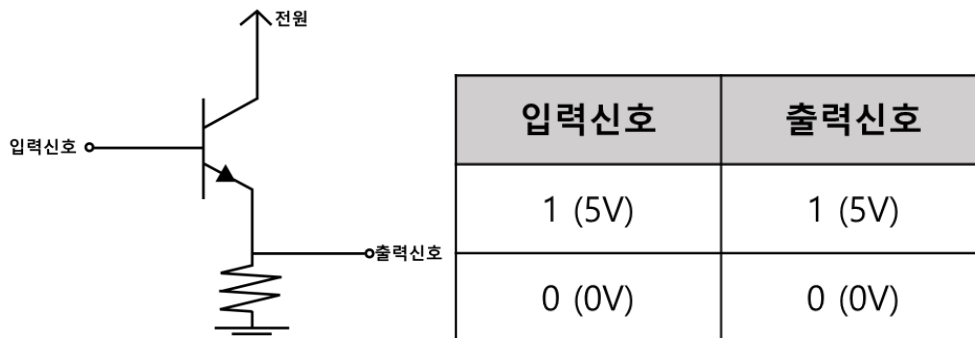


Figure 1. 트랜지스터(원) 및 주변회로와 입출력 신호 논리 표

다음과 같이 입력신호에 전기가 연결되면 출력신호에도 전기가 흐르고, 입력신호에 전기가 끊어지면 출력신호도 전기가 끊어지는 방식으로 트랜지스터는 동작합니다. 이러한 트랜지스터를 조합하면 가장 기본적인 논리 소자인 Not Gate를 만들 수 있게 됩니다.

## 2. Logic Gate(논리 게이트)

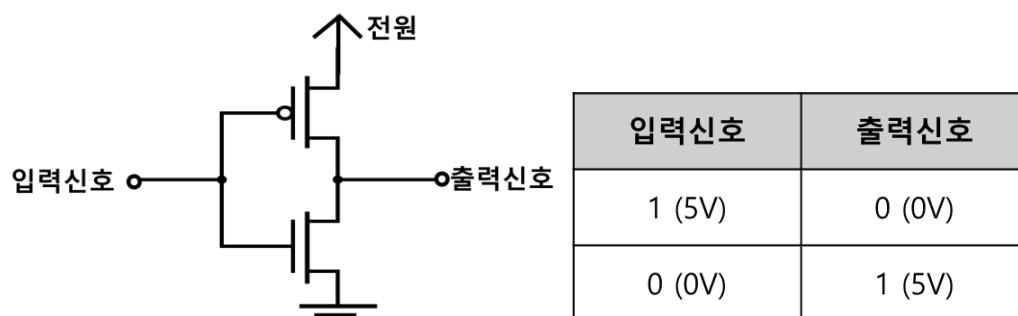


Figure 2. Not Gate와 입출력 신호 논리 표

소자 그림 앞에 붙어있는 작은 동그란 원은 Figure 1에 있는 트랜지스터와 반대로 동작한다는 뜻입니다. 이렇게 만들어진 Not Gate는 우리가 주로 사용하는 논리대수(Booleean algebra)중 Not을 구현하는데 이용하게 됩니다. 이와 마찬가지로 NAND와 NOR연산도 다음과 같은 회로 조합으로 만들 수 있습니다.

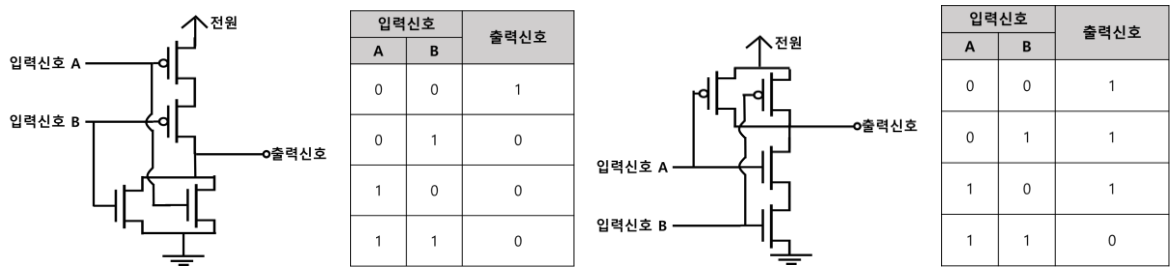


Figure 3. NOR Gate와 NAND Gate.

만들어진 회로 조합을 통해 우리는 모든 논리 게이트들을 만들 수 있게 됩니다.

Logic Gates									
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR		
Alg. Expr.	$\neg A$	$A \cdot B$	$\neg(A \cdot B)$	$A + B$	$\neg(A + B)$	$A \oplus B$	$\neg(A \oplus B)$		
Symbol									
Truth Table	A	X	B	A	X	B	A	X	B
	0	1	0	0	0	0	0	0	0
	0	1	0	1	0	0	1	1	0
	1	0	0	1	0	1	0	1	0
	1	0	1	0	1	1	0	1	1
	1	1	0	0	1	1	1	0	1
	1	1	1	0	0	1	1	1	0
	1	1	1	1	1	0	1	1	1

Figure 4. 기본적인 Logic Gate

모든 논리 게이트들을 만들 수 있게 되면서 우리는 트랜지스터와 전기신호로 Boolean algebra를 구현할 수 있게 되었습니다. 논리 대수(Booleean algebra)를 구현할 수 있게 되면 드디어 우리는 '수식을 연산' 할 수 있게 됩니다.

3. Arithmetic logic unit(산술 논리 장치)

가장 기초적인 연산인 1Bit끼리의 덧셈 회로는 다음과 같습니다.

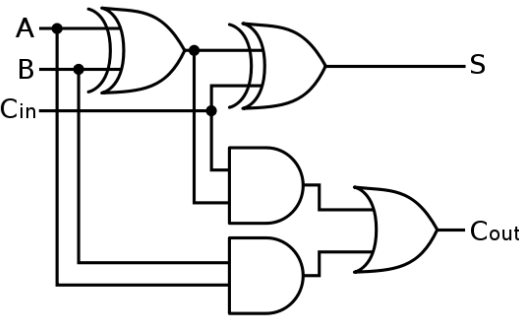


Figure 5. 1Bit 덧셈기 예시

회로의 A, B 부분은 1bit 값을 입력해주는 부분이며, C<sub>in</sub>은 덧셈에서 만들어지는 Carry 값을 의미합니다. 위 회로를 우리가 아는 수학 식으로 표현하면 다음과 같습니다.

$$\begin{array}{r}
 \text{Carry 값} \\
 \text{1} \\
 2 \quad 6 \\
 + \quad 1 \quad 5 \\
 \hline
 4 \quad 1
 \end{array}$$

Figure 6. 덧셈

Carry라는 것은 위와 같이 아래에 있는 숫자가 범위를 넘었을 경우 윗자리로 올려주는 값을 의미합니다.

여기서 1Bit연산기를 확장하여 int와 같은 32bit 값을 연산하기 위해서는 어떻게 해야 할까요? 바로 논리 회로로 만들어진 Figure 5와 같은 덧셈기를 32개 붙여주면 됩니다. (대부분의 하드웨어적인 확장은 그 크기만큼 붙여주면 됩니다)

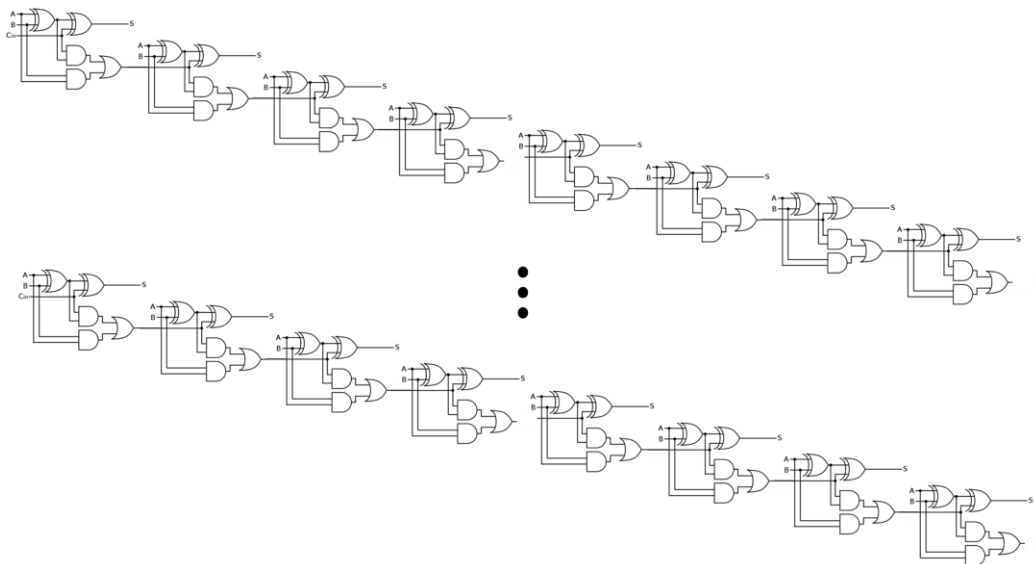


Figure 7. 32Bit 덧셈기 Logic Gate 예시

위 회로는 32개의 Bit로 이루어진 값들을 하나하나 비교를 수행하면서 덧셈의 결과를 만들어주

는 덧셈기입니다. 각 bit 및 carry값을 더해 결과를 만들어 주며, 32개가 연속적으로 있기 때문에 32bit 값을 더할 수 있는 것이지요.

이런 식으로 뺄셈, 곱하기, 나누기, 비교기 등을 모두 만들 수 있으며, 이것들을 조합하여 우리는 컴퓨터로 수식을 연산할 수 있게 됩니다. 그럼 이제 드디어 다음과 같이 수식을 입력하여 컴퓨터를 통해 계산할 수 있게 됩니다.

$$Reulst = (2 \times 3) + (3 \times 4) + 5$$

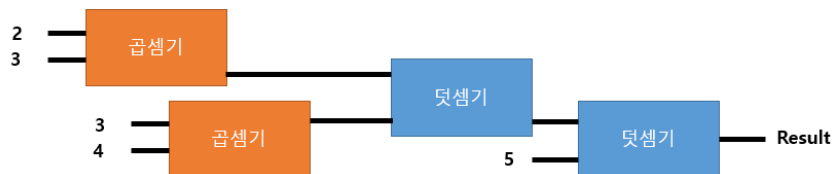


Figure 8 덧셈 식과 회로 표현 예시

위 수식을 하드웨어를 통해 계산했을 때의 예시입니다. 각 연산에 맞추어 곱셈기와 덧셈기를 조합하는 방식으로 구현하였습니다. 2와 3을 곱한 뒤, 3과 4를 곱하고 이를 두개 더한다음에 5를 더하는 식으로 구성되어 마지막에는 결과 값을 출력해 주게 됩니다.

이렇게 회로를 구성하여 구현하여도 좋겠지만, 한가지 아쉬운 점이 있습니다. 바로 재 사용성입니다. 하드웨어를 만드는 것은 고 비용의 일이기 때문에 가능하다면 이런 반복되는 부분을 줄여 효율성을 높일 필요가 있습니다. 그래서 좀더 효율적으로 연산을 하기위해 다음과 같은 회로를 고안해 내었습니다.

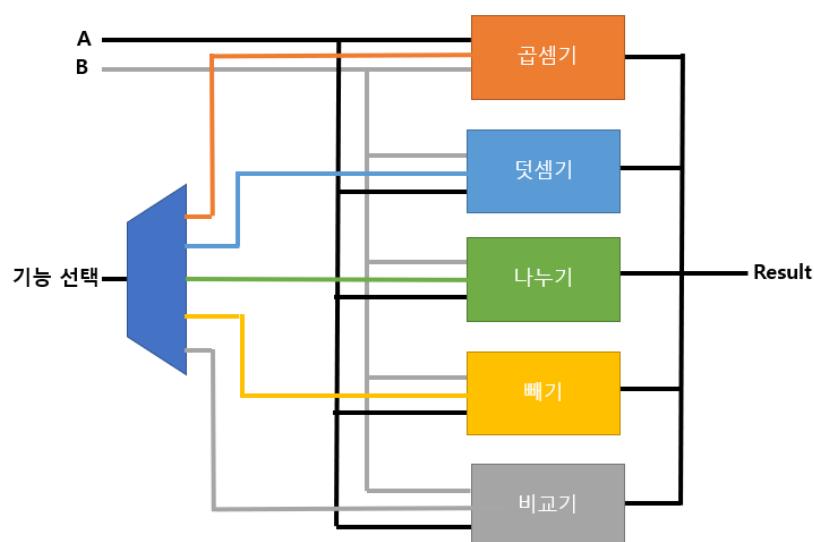


Figure 9. 좀더 효율적인 연산기의 조합

위 구조는 입력된 A,B값들이 기능선택 기능을 통해 선택된 연산기를 통해 결과가 출력되도록

만들어져 있습니다. 이런 식으로 구현할 경우 어떤 수식이 필요하더라도, 기능 선택부분의 값을 바꿔 줌으로써 A,B에 대한 모든 연산을 수행 할 수 있게 됩니다. 이러한 구조를 가진 회로 단위를 **산술 논리 장치 ALU(Arithmetic Logic Unit)**이라고 하며, 기능 선택 버튼을 **OP Code(Operation Code)**라고 합니다.

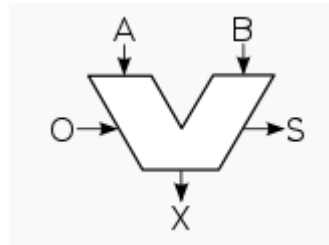


Figure 10. ALU의 회로 기호, (A,B는 입력, X는 출력, O는 Opcode, S는 상태신호)

그러면 우리는 이제 범용성을 갖춘 회로를 이용해 위 수식을 이렇게 작성할 수 있습니다.

#### 4. 최초의 컴퓨터 언어, 기계어

위 회로를 이용해 Figure 8번의 수식을 표현 해보겠습니다. 이제는 ALU 회로 한 개만을 이용해야 하기 때문에 순차적으로 값을 입력해줄 필요가 있습니다. 표를 이용해 나타내면 다음과 같습니다.

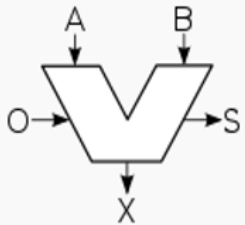
	Line	Operation Code	A	B
	0	곱하기	2	3
	1	0번 line 결과 저장		
	2	곱하기	3	4
	3	더하기	0번 line 결과	2번 line 결과
	4	더하기	3번 line 결과	5

Figure 11. ALU를 이용한  $(2 \times 3) + (3 \times 4) + 5$ 의 연산

각 연산 순서에 맞추어 OP code와 값을 작성하고 이를 순서대로 동작 시키게 되면 회로의 결과값은 우리가 연산하고 싶은 식의 결과 값이 됩니다. 이것이 바로 최초의 기계어이며, 우리가 매일 작성하고 있는 Code라는 것의 근원입니다. 여러분들이 작성하시는 C언어는 컴파일러를 통해 위 기계어로 번역되어지고, 그 기계어에 맞추어 순서대로 동작 하여 여러분이 원하시는 결과물을 만들어 주게 되는 것입니다.

우리가 작성한 코드가 동작하는 원리에 대하여 간단히 소개해 드렸습니다. 이러한 원리를 간단히나마 알고 계신다면 좀 더 효율적인 소프트웨어를 작성하시는데 도움이 될것이라 생각합니다.

추신

(1) Figure 10의 0line결과 저장 등은 실제로는 Register라는 것들을 이용해 구현됩니다.

- (2) 회로의 의미를 이해하실 필요는 없습니다. 그냥 이런것들이 있구나 하는 흐름만 기억해주세요.
- (3) 하드웨어와 밀접한 관계에 있는 소프트웨어를 작성할 경우, 실제 소프트웨어를 작성한 뒤에 하드웨어 단에서 최적화 시킬 수 있는 부분을 나누어 하드웨어 구현을 하기도 합니다. 이런 것을 하드웨어 가속 이라고 하며, 요즘 머신러닝에서 많이 사용하는 NPU같은 것들도 이 종류 중 하나입니다.