

진법과 전자계산의 원리

한남대학교 수학과
김상배 교수

<http://sbk.hnu.kr>

진법이란

컴퓨터는 전기의 on 과 off 에 의하여 숫자를 표현하고 계산한다.

전자계산의 원리를 이해하려면, 2진법을 (혹은 n진법)을 이해할 필요가 있다.

수의 표시

1부터 시작하여 순서대로 무한이 늘어 서 있는 자연수는 꼭 10진법으로 이름을 부여할 필요는 없다. 자연수들에 이름을 주는 방법은 무수히 많다.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (10진법)

1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 (16진법)

I II III IV V VI VII VIII IX X XI XII XIII XIV XV XVI XVII XVIII XIX XX

n진법(n개의 기호로 숫자를 표시하는 방법)

4진법(4개의 기호로 숫자를 표시)

A B C AD AA AB AC BD BA BB BC CD CA CB CC CDD

1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 100

2진법 (2개의 기호로 숫자 표시)

- 1자리 0, 1 (자연수 2개 표시가능)
- 2자리 00, 01, 10, 11 (자연수 4개 표시가능)
- 3자리 000, 001, 010, 011, 100, 101, 110, 111 (자연수 8개 표시가능)
-

자리수를 늘리면 0과 1만으로도 모든 자연수에 이름을 줄 수 있다.

0, 1, 2, 3, 4, 5, 6, 7, 8, ... (10진법)

0, 1, 10, 11, 100, 101, 110, 111, 1000, ... (2진법)

노트에 필기하다보면 2진법이 자리를 많이 차지 한 것 같지만, 컴퓨터 기억장치에는 어차피 전기적 on 과 off 상태를 저장하기 때문에 결국 10진법도 두 가지 상태로 분해하여 저장하니 2진법이나 마찬가지이다.

컴퓨터의 숫자 표시

컴퓨터는 모든 10진법의 숫자를 2진법으로 표시하여 저장하고 계산한다.

사람에게 보여줄 때만 10진법으로 다시 바꾸어 보여 준다.

한남대학교 수학과 김상배교수

2진법 셈하기

덧셈, 뺄셈, 곱셈, 나눗셈의 기초는 덧셈이다.

2진법 덧셈하기

$$(0)_2 + (0)_2 = (0)_2$$

$$(0)_2 + (1)_2 = (1)_2$$

$$(1)_2 + (0)_2 = (1)_2$$

$$(1)_2 + (1)_2 = (10)_2$$

$$(10)_2 + (1)_2 = (11)_2$$

$$(10)_2 + (10)_2 = (100)_2$$

$$(11)_2 + (10)_2 = (101)_2$$

$$(11)_2 + (11)_2 = (110)_2$$

논리계산과 진리표

모든 논리는 3가지 (NOT, AND, OR) 연산만으로 표시될 수 있다.

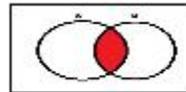
NOT 진리표

A	\bar{A}
0	1
1	0



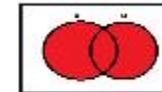
AND 진리표

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1



OR 진리표

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1



참고: $A \wedge B$ 를 AB 로 $A \vee B$ 를 $A + B$ 로 표현하기도 함.

복합연산인 XOR 진리표 $A \oplus B = (A \wedge \bar{B}) \vee (\bar{A} \wedge B)$

A	B	\bar{A}	\bar{B}	$A \wedge \bar{B}$	$\bar{A} \wedge B$	$A \oplus B$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0



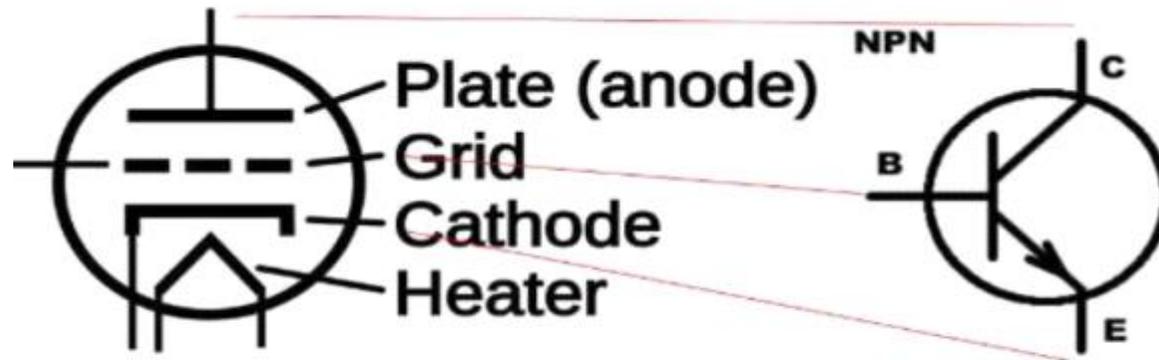
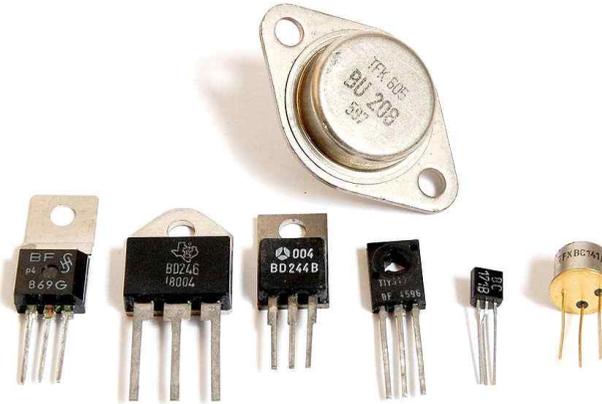
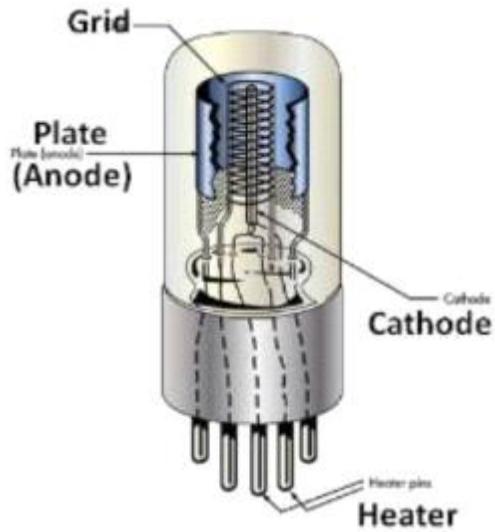
XOR
= Exclusive OR
= 배타적 논리합

논리계산으로 2진법 덧셈 가능

$$C = A \wedge B \quad S = A \oplus B$$

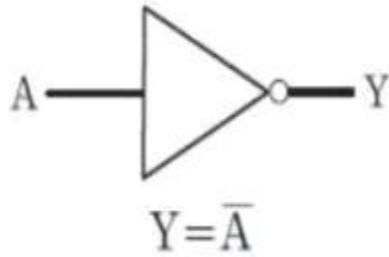
A	B	C	S	A + B
0	0	0	0	00
0	1	0	1	01
1	0	0	1	01
1	1	1	0	10

3극진공관과 트랜지스터

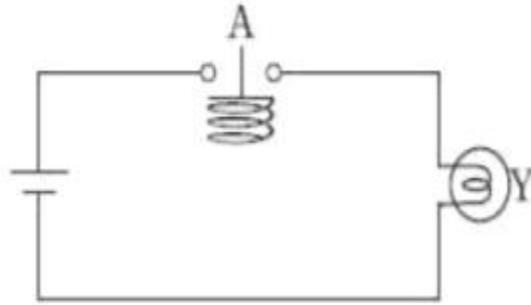


진공관 Transistor
 Plate = C: Collector
 Grid = B: Base
 Kathode = E: Emitter

논리회로(NOT 스위치)



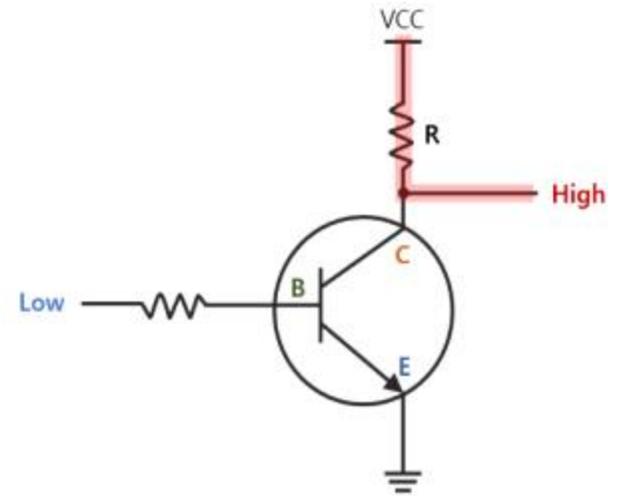
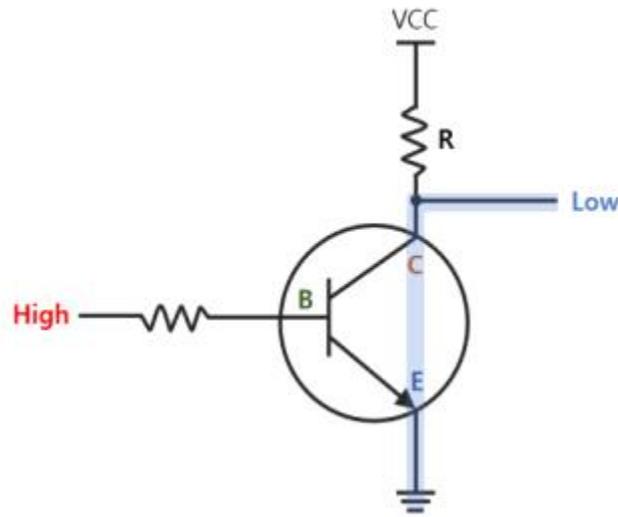
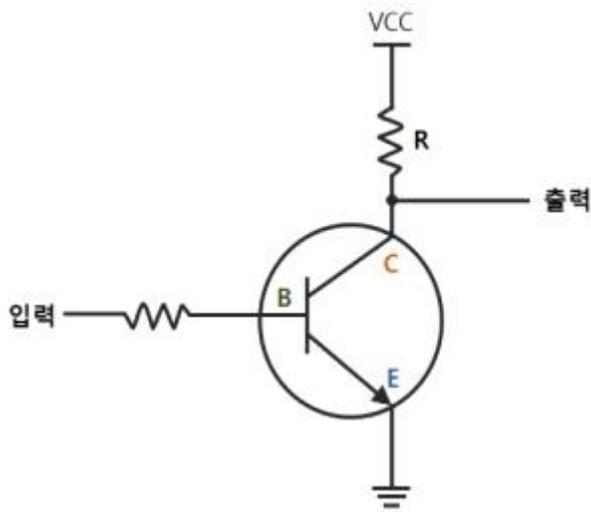
(a) 기호와 논리식



(b) 스위치 회로

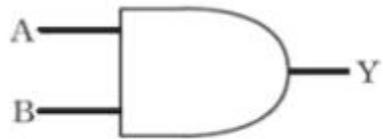
A	Y
0	1
1	0

(c) 진리표



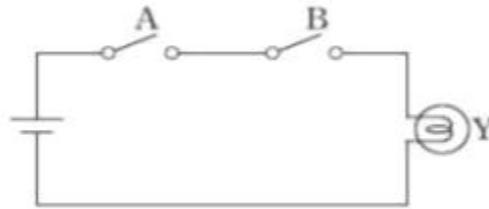
(d) 트랜지스터 회로

논리회로(AND 스위치)



$$Y = A \cdot B$$

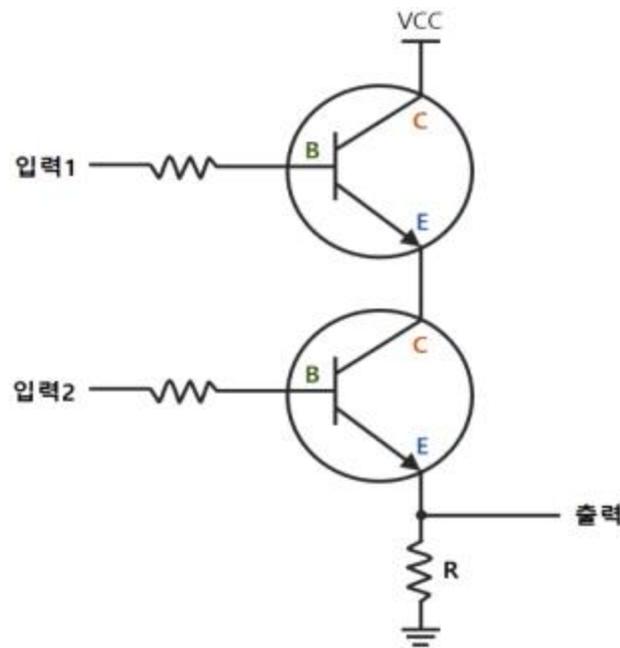
(a) 기호와 논리식



(b) 스위치 회로

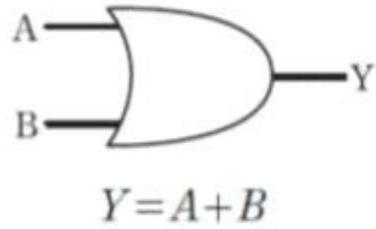
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(c) 진리표

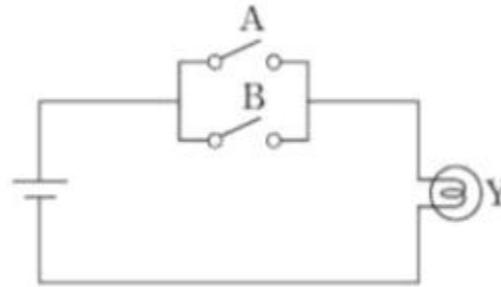


(d) 트랜지스터 회로

논리회로(OR 스위치)



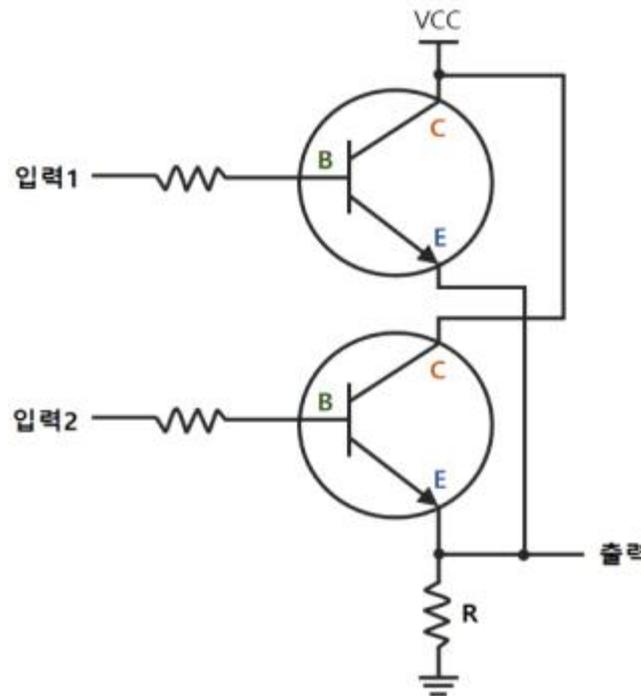
(a) 기호와 논리식



(b) 스위치 회로

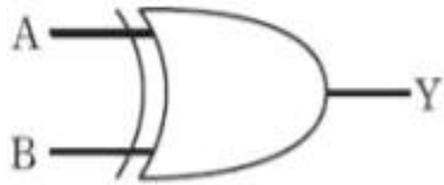
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(c) 진리표



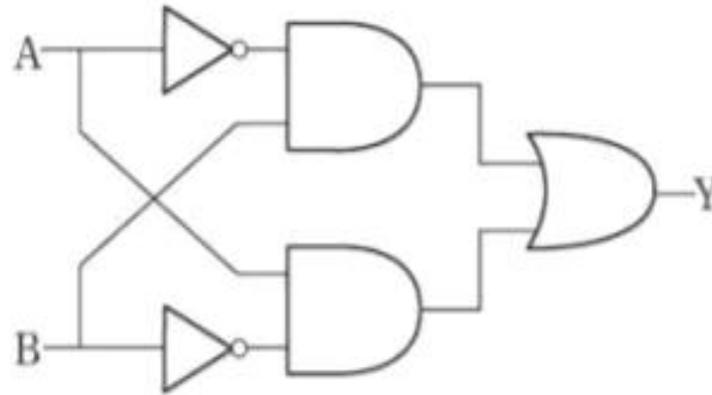
(d) 트랜지스터 회로

논리회로 (XOR 스위치 : $Y = A \oplus B = A\bar{B} + \bar{A}B = A \wedge \bar{B} + \bar{A} \wedge B$)



$Y = A \oplus B$
 또는 $Y = A \cdot \bar{B} + \bar{A} \cdot B$

(a) 기호와 논리식



(b) 개념도

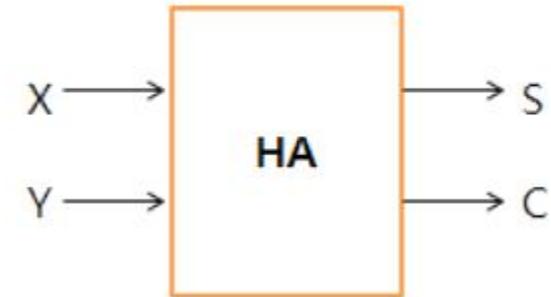
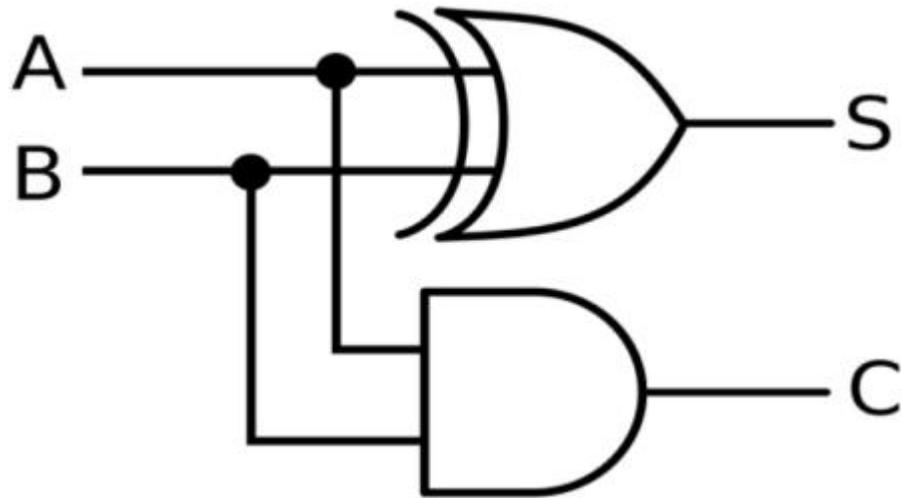
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(c) 진리표

A	B	\bar{A}	\bar{B}	$A\bar{B}$	$\bar{A}B$	$Y = A\bar{B} + \bar{A}B$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

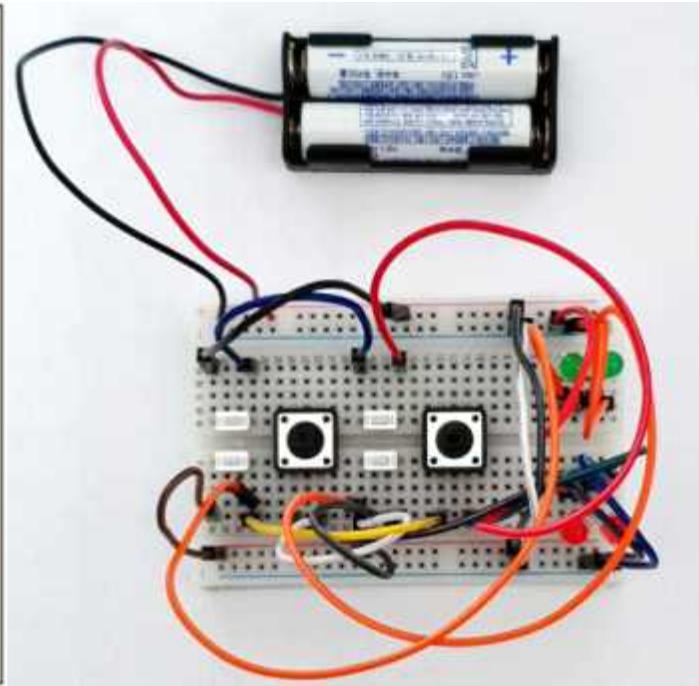
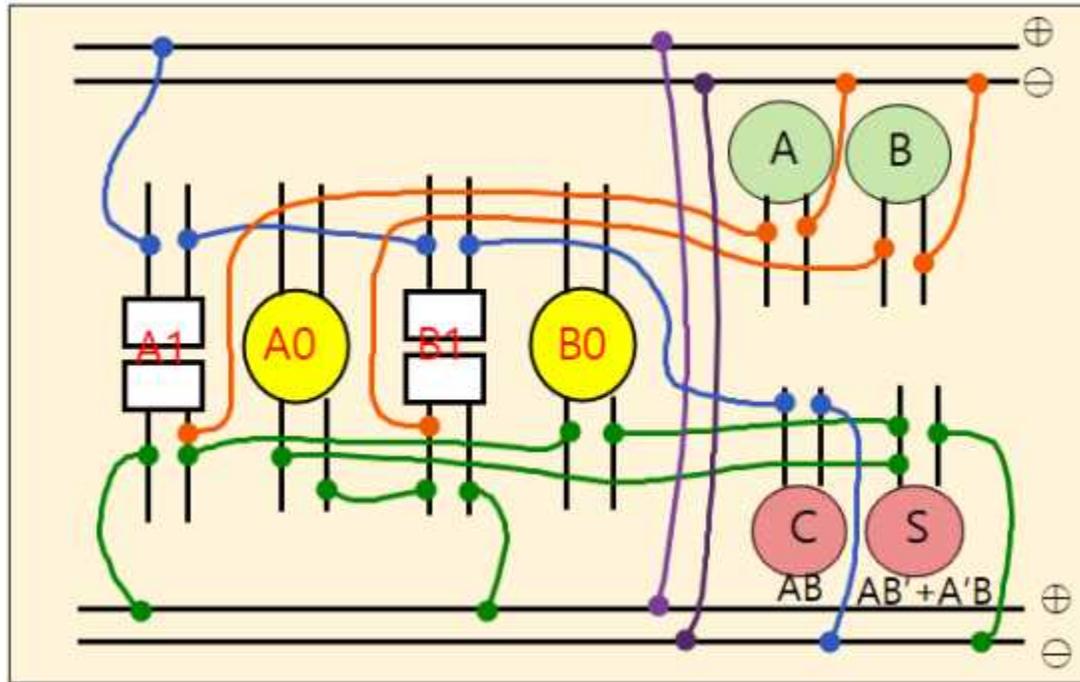
논리회로 시뮬레이션 : <https://javalab.org/logic/>

논리게이트로 구현하는 반가산기(Half Adder)



A	B	$C = AB$	$S = A \oplus B$	CS
0	0	0	0	00
0	1	0	1	01
1	0	0	1	01
1	1	1	0	10

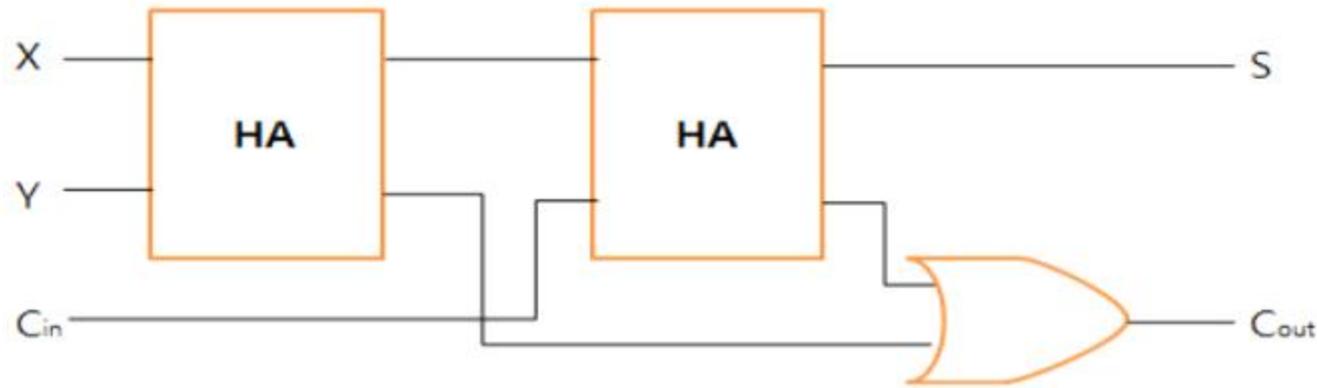
스위치로 만드는 반가산기



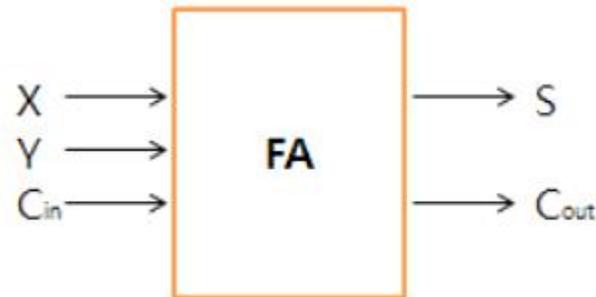
A	B	CS
0	0	00
0	1	01
1	0	01
1	1	10

전가산기 (Full Adder : 반가산기 2개와 OR게이트를 사용)

아래 자리수에 올라온 C_{in} 까지 더하여 $X + Y + C_{in}$ 을 수행한다.



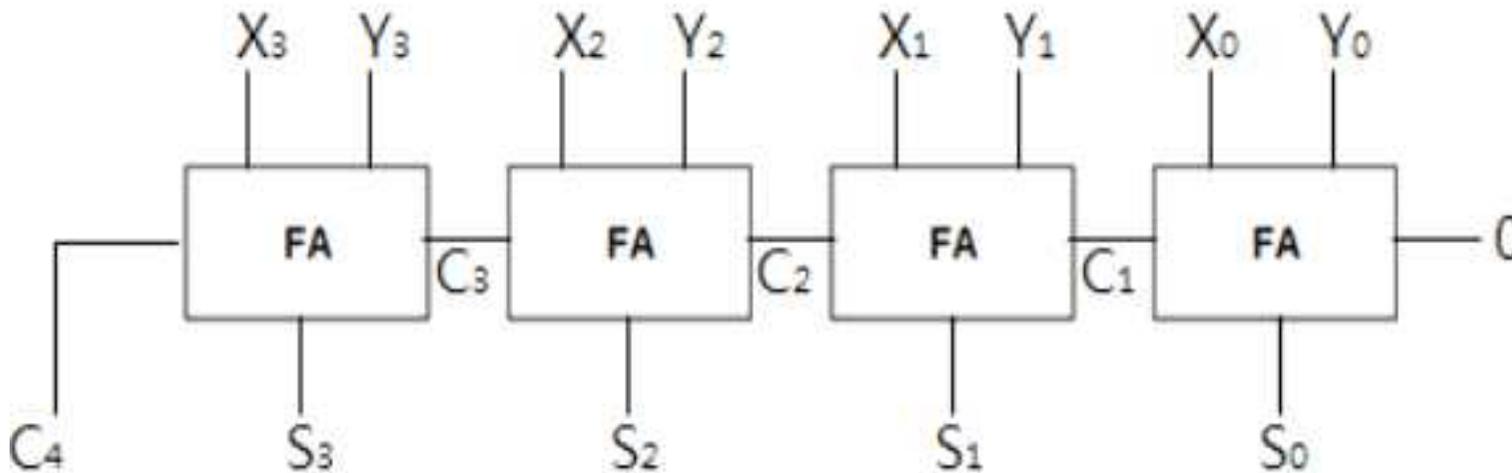
입력			출력	
X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



참고: 첫째 HA에서 둘째 자리수가 $C=1$ 이면 첫째 자리수가 $S=0$ 이 되므로 둘째 HA에서는 둘째 자리수 $C=1$ 이 나올 수가 없어서, C_{out} 는 $1+1$ 이 되어 셋째 자리수로 올라가는 일은 없다.

병렬가산기(전가산기 n 개 연결하여 n 비트 2진수 덧셈)

전가산기 여러개를 병렬로 연결하면 n 비트 가산기를 만들 수 있다. 아래 예는 4비트 가산기인데, 4비트의 2진수 $X_3X_2X_1X_0$ 와 $Y_3Y_2Y_1Y_0$ 를 합한 결과는 2진수 $S_3S_2S_1S_0$ 이며, 최종 윗자리수로 올라가는 수는 C_4 이다.



10의 보수를 이용한 십진법의 뺄셈

10의 (보수)를 덧셈한 후, (쉬운 10)을 뺄셈한다.

1) 결과가 양수인 경우

예) $7 - 4 = 7 + (6 - 10) = (7 + 6) - 10 = 13 - 10 = 3$

7에 4의 보수인 6를 더하여 13을 얻는다.

13에서 10을 빼서 3을 얻는다. 즉 자리 수가 올라갈 경우는

13에서 10의 자리를 버리고 1의 자리인 3만 취한다.

2) 결과가 음수인 경우

예) $2 - 4 = 2 + (6 - 10) = (2 + 6) - 10 = 8 - 10 = -2$

2에 4의 보수인 6를 더하여 8을 얻는다.

8에서 10을 빼서 -2를 얻는다. 즉 자리 수가 올라가지 않는 경우는

8에서 보수인 2를 구하고 마이너스를 붙인다.

2의 보수의 정의

주어진 2진수 x 보다, 한 자리 높은 **2의 제곱수**(최고 자리가 1이고 나머지 모든 비트가 0인 2진수)에서, x 를 빼서 얻은 수가 x 의 **2의 보수**이다. 과연 2진수 x 와 2의 보수의 합이 2의 제곱수가 되는 것을 확인해 보자. 우선 x 의 2의 보수를 얻는 방법은 x 의 모든 자리의 숫자를 반전(0을 1로, 1을 0으로)시킨 뒤 여기에 1을 더하는 것이다. 예를 들어, 01001011 의 모든 자리의 수를 반전시키고 마지막에 1을 더하면 2의 보수 10110101 를 얻을 수 있는데 이 수에 원래의 수를 더하여 보면

$$\begin{array}{r} 01001011 \quad (= 75_{(10)}) \\ +) 10110101 \quad (= -75_{(10)}) \\ \hline 10000000 \quad (= 256_{(10)} = 2^8_{(10)}) \leftarrow \text{2의 제곱수} \end{array}$$

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

와 같이 **2의 제곱수**가 나온다. 그런데 2진수의 각 비트는 **XOR** 게이트를 이용하면 쉽게 반전된다. (XOR진리표 참조)

2의보수를 이용한 뺄셈

10진법 계산으로는 $100 - 75 = 100 + (-75) = 25$ 인데,

2진법 계산에서 2의 보수를 이용하면 같은 결과를 얻을 수 있을까?

$$\begin{array}{r} 01100100 \quad (= 100_{(10)}) \\ +) \underline{10110101} \quad (= -75_{(10)}) \end{array}$$

100011001 올림수를 1을 버리고 나머지 00011001를 취한다.

$$00011001 \quad (= 25_{(10)})$$

10진법 계산으로는 $50 - 75 = 50 + (-75) = -25$ 인데,

2진법 계산에서 2의 보수를 이용하면 같은 결과를 얻을 수 있을까?

$$\begin{array}{r} 00110010 \quad (= 50_{(10)}) \\ +) \underline{10110101} \quad (= -75_{(10)}) \end{array}$$

$$11100111 \quad (= 231_{(10)} = 256_{(10)} - 25_{(10)} = 2^8_{(10)} - 25_{(10)})$$

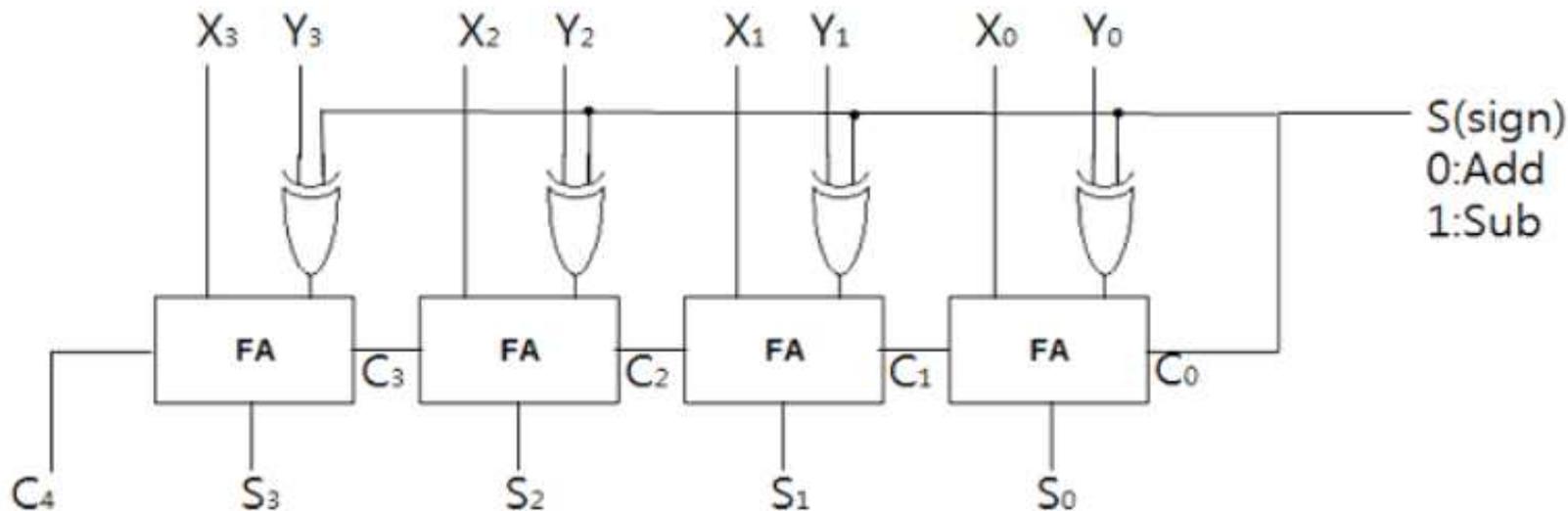
올림수가 없으므로 11100111의 2의보수를 구하고 (-)부호를 붙인다.

$$-00011001 \quad (= -25_{(10)})$$

병렬 가감산기(덧셈과 뺄셈 가능)

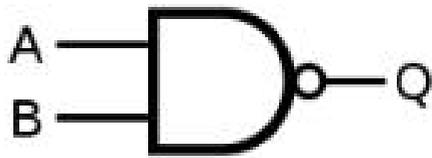
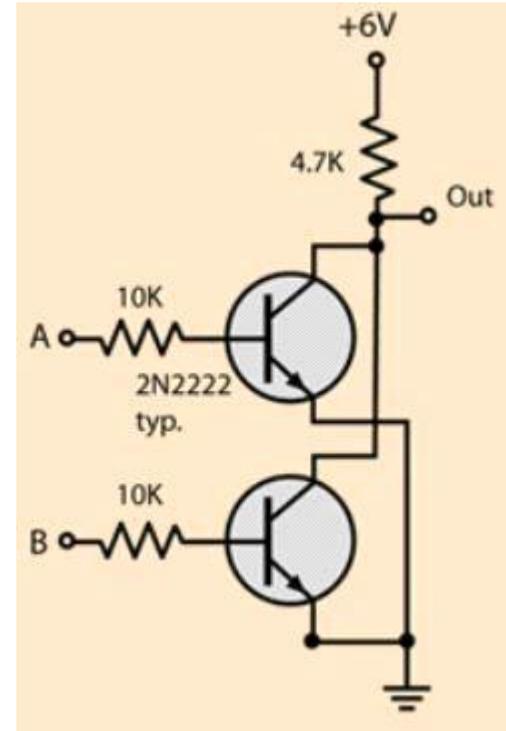
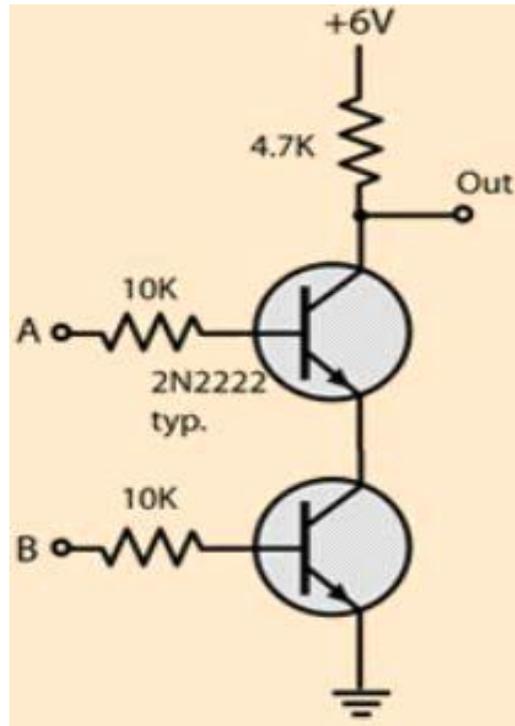
2진법의 보수를 사용하면 가산기를 사용하여 뺄셈 연산을 할 수 있다

- ① 감수 Y 의 각 비트를 반전한 후 마지막 비트에 1을 더하여 **2의보수**(Y 와 더하면 최고 자리만 1이고 나머지 비트들은 0이 되게 하는 수)를 구한다.
- ② 피감수 X 와 감수 Y 의 **2의보수**를 더한다.

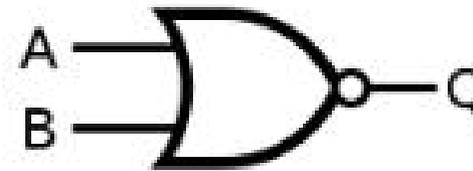


- ③ 올림수 C_4 가 0이 아니면 버리고, 나머지 자리의 값($S_3S_2S_1S_0$)을 취한다.
- ④ 올림수 C_4 가 0이면 $S_3S_2S_1S_0$ 에 대한 **2의보수**를 구하여 **(-)부호**를 붙인다.
(실제로는, **2의보수**를 구하지 않고, $0S_3S_2S_1S_0$ 를 **음수**로 간주하기도 한다.)

범용 게이트 (Universal Gates : NAND, NOR)



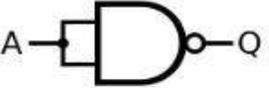
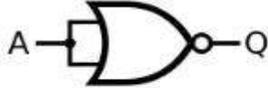
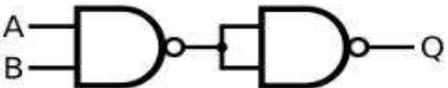
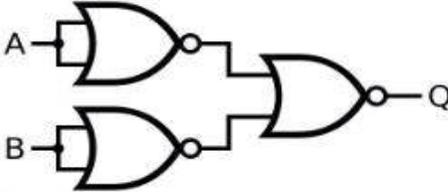
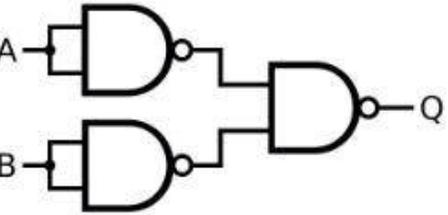
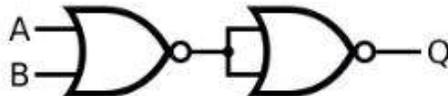
$$\overline{A \wedge B}$$



$$\overline{A \vee B}$$

범용 게이트 (Universal Gates : NAND, NOR)

하나의 게이트로 NOT, AND, OR를 표현할 수 있다.

type	NAND construction	NOR construction
NOT		
AND		
OR		

$$\overline{A \wedge A} = \overline{A}$$

$$\overline{A \vee A} = \overline{A}$$

$$\overline{\overline{A \wedge B}} = A \wedge B \text{ (부정의 부정)}$$

$$\overline{\overline{A \vee B}} = A \vee B \text{ (드모르간의 법칙)}$$

$$\overline{\overline{A} \wedge \overline{B}} = A \vee B \text{ (드모르간의 법칙)}$$

$$\overline{\overline{A} \vee \overline{B}} = A \wedge B \text{ (부정의 부정)}$$