그렇다면 뺄셈은 어떨까요?

윤경담
스마트팜학과

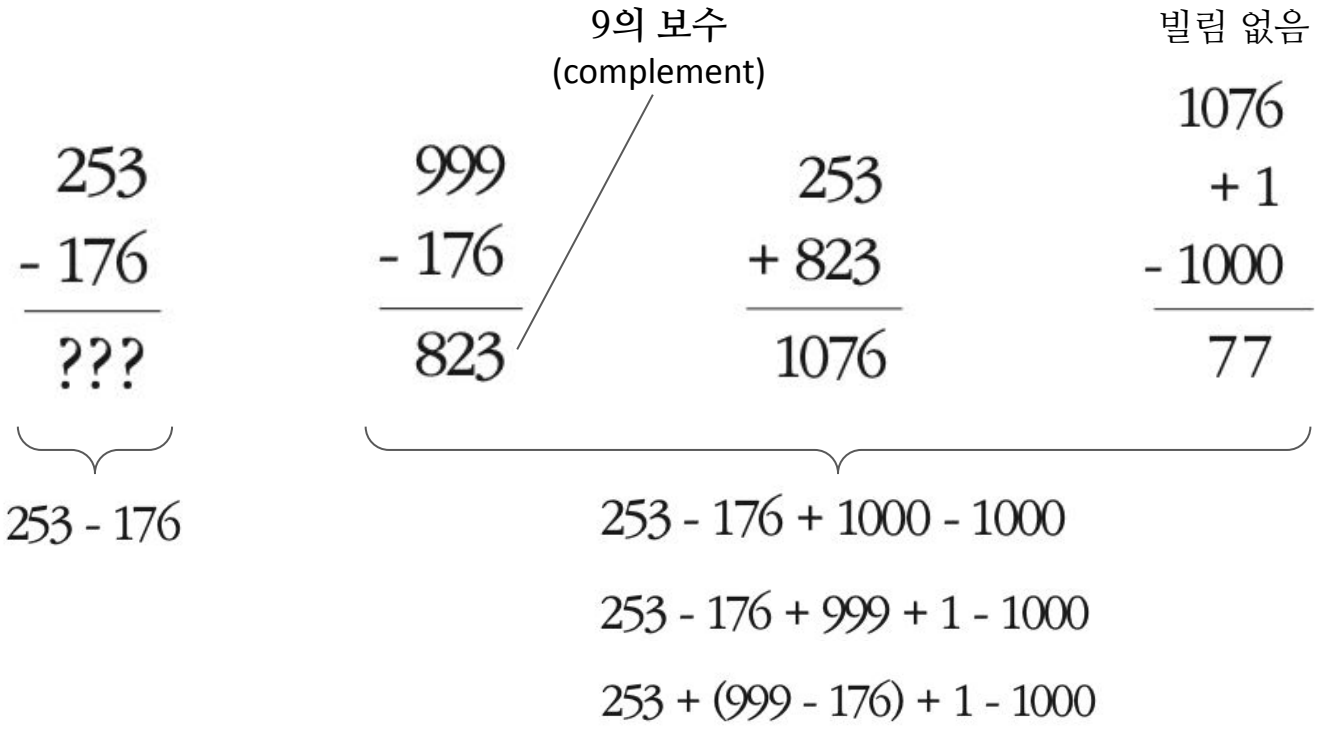
빌림수 (borrow)

$$\begin{array}{r} 253 \\ -\ 176 \\ \hline ??? \end{array}$$

뺄셈

	빼어지는 수	(subtrahend)
-	빼는 수	(minuend)
	두 수의 차	(difference)

뺄셈: 큰수 - 작은수


9의 보수
(complement)
빌림 없음
253
- 176
???
999
- 176
823
253
+ 823
1076
1076
+ 1
- 1000
77
253 - 176
253 - 176 + 1000 - 1000
253 - 176 + 999 + 1 - 1000
253 + (999 - 176) + 1 - 1000

뺄셈: 작은수 - 큰수

빌림 없음

176	176	999	176	922	922
- 253	- 253	- 253	+ 746	+1	- 999
- 77	???	746	922	-1000	-77
				???	

이진수 뺄셈: 큰수 - 작은수

```
  253
- 176
-----
  ???
```

```
  11111101
- 10110000
----------
  ????????
```

```
  11111101
+ 01001111
----------
  101001100
```

```
  101001100
+         1
-----------
  101001101
```

```
  101001101
- 100000000
-----------
    1001101
```

```
  11111111
- 10110000
----------
  01001111
```

1의 보수
(complement)

이진수 뺄셈: 작은수 - 큰수

```
  176
- 253
-----
  ???
```

```
  10110000
- 11111101
----------
  ????????
```

```
  10110000
+ 00000010
----------
  10110010
```

```
- 11111111
  10110010
+         1
-----------
  10110011
```

```
  10110011
-100000000
----------
  ????????
```

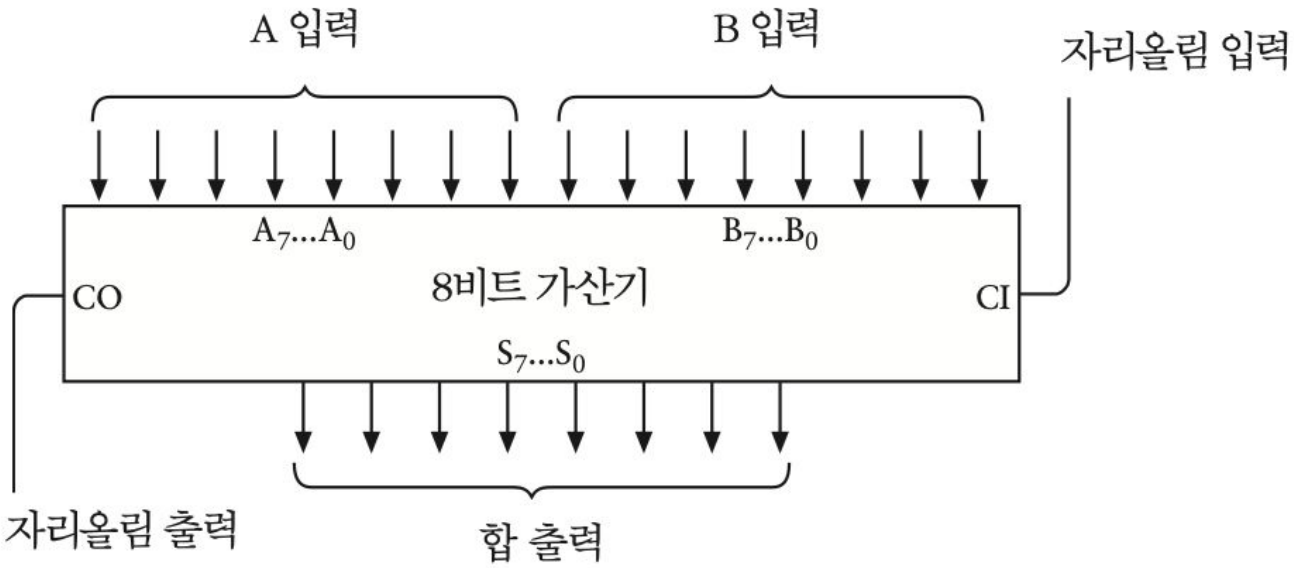
```
  11111111
- 11111101
----------
  00000010
```

1의 보수
(complement)

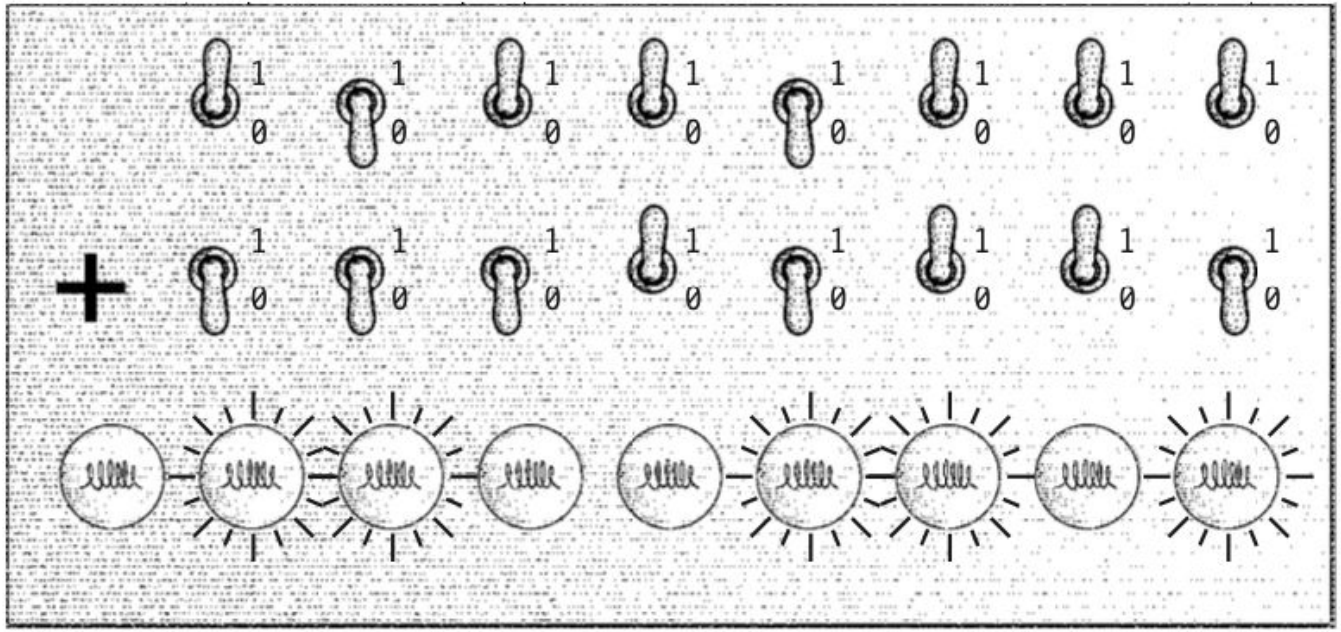
```
  11111111
- 10110010
----------
- 01001101
```

빌림 없음

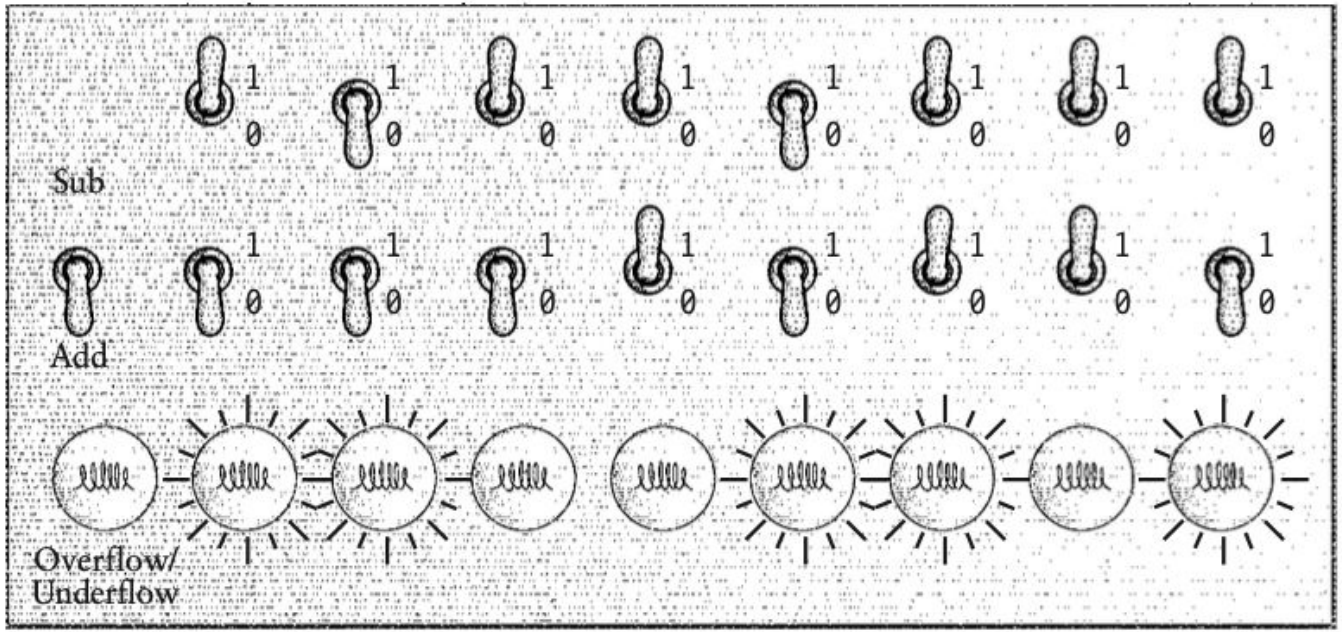
가산기 (덧셈기)


A 입력
B 입력
자리올림 입력
$A_7 \ldots A_0$
$B_7 \ldots B_0$
CO
8비트 가산기
CI
$S_7 \ldots S_0$
자리올림 출력
합 출력

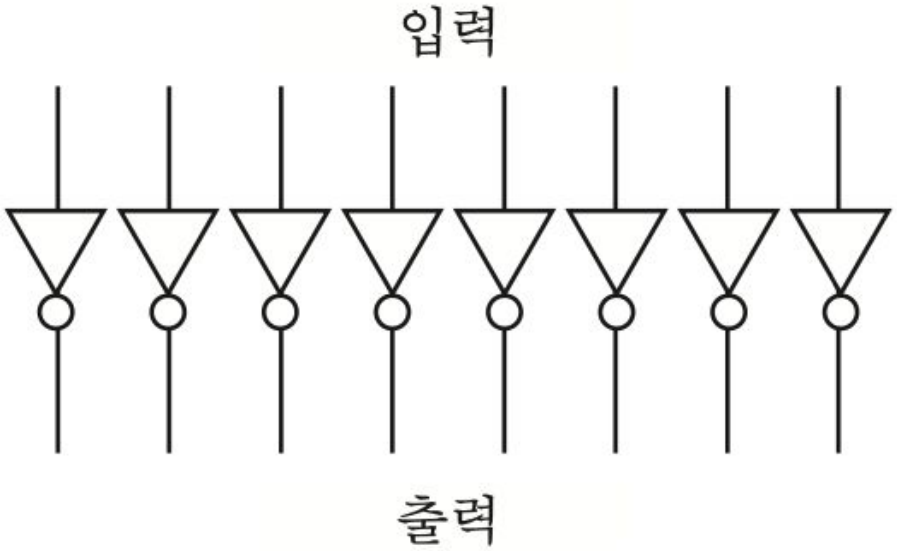
가산기: 제어판



가감산기: 제어판

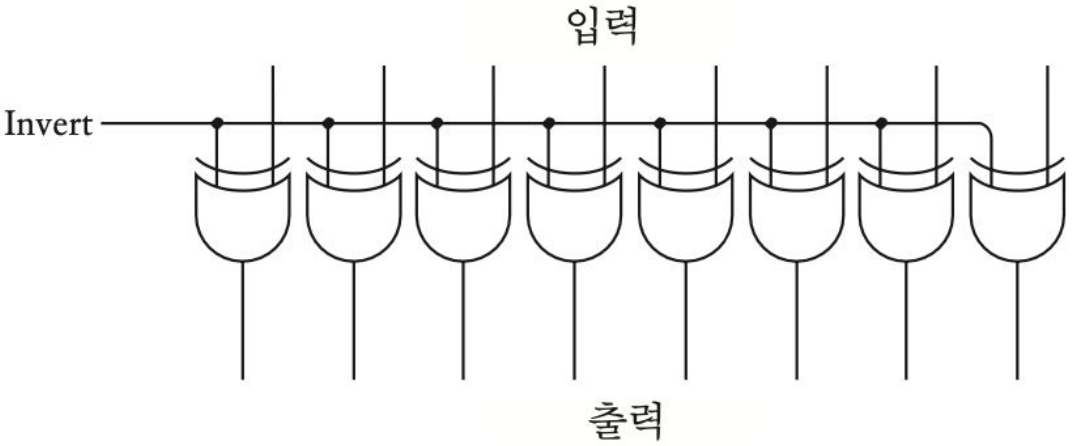


보수: 논리회로



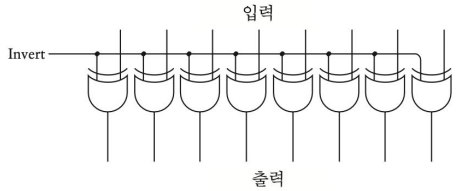
8비트 숫자에 대한 1의 보수 (complement)

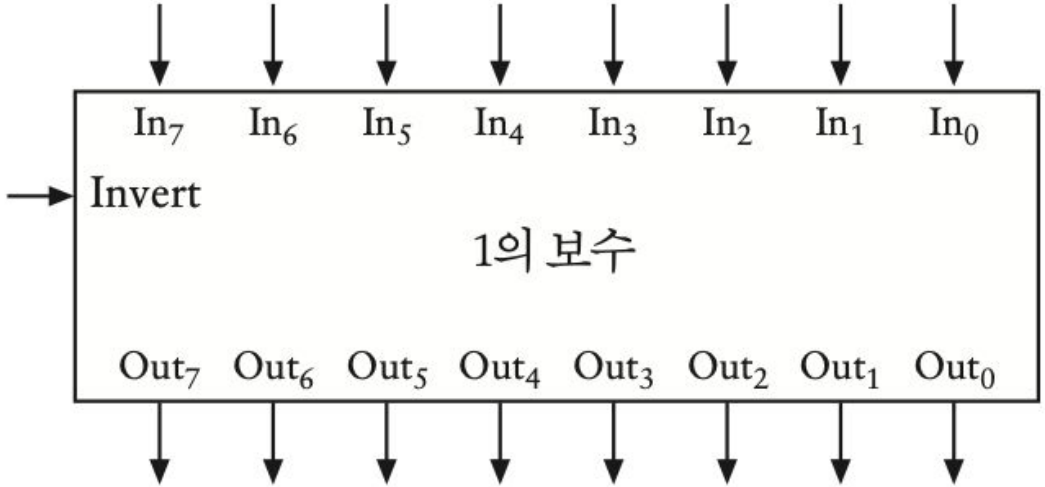
보수: 논리회로



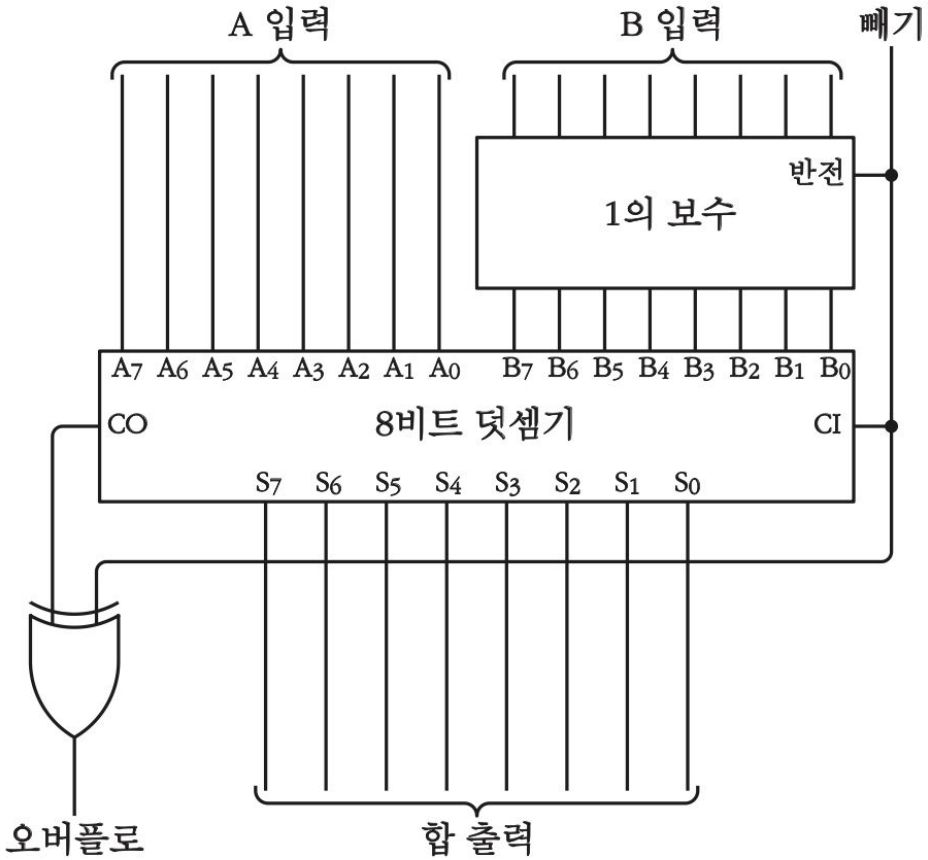
XOR	0	1
0	0	1
1	1	0

보수: 논리회로


입력
Invert
출력


In_7 In_6 In_5 In_4 In_3 In_2 In_1 In_0
Invert
1의 보수
Out_7 Out_6 Out_5 Out_4 Out_3 Out_2 Out_1 Out_0

가감산기: 논리회로


A 입력
B 입력
빼기
반전
1의 보수
A7 A6 A5 A4 A3 A2 A1 A0
B7 B6 B5 B4 B3 B2 B1 B0
CO
8비트 덧셈기
CI
S7 S6 S5 S4 S3 S2 S1 S0
오버플로
합 출력

음수의 표현

이 장에서 음수에 대하여 몇 번 언급이 되었지만, 어떤 형태로 음수를 표현하는지에 대해서 정확히 말씀 드린 것은 아니었습니다. 우선 십진수에서 음수의 표현에 사용했던 방식인 숫자 앞에 음수 기호(마이너스 기호죠)를 적어주는 방법을 생각해 볼 수 있습니다. 예를 들어 -77은 이진수로 -1001101이라 쓸 수도 있겠지요. 실제로 이렇게 할 수도 있지만 이진수를 사용할 때 음수 기호를 포함한 모든 것을 0과 1로만 표현될 수 있도록 만드는 것이 중요한 목표였지요.

음수의 표현: 부호숫자 (sign digit)

물론 음수 기호를 위하여 한 비트를 추가하는 방법을 사용할 수 있습니다. 즉 한 비트를 더 두어서 1인 경우 음수를, 0인 경우 양수를 표현하게 하는 것이지요. 이 방식은 충분히 많이 바뀐 것은 아니지만 표기하는 데 문제는 없습니다.[3] 위에서 설명된 방식 이외에 음수를 표현하는 또 다른 방식이 있는데, 이 방식은 양수와 음수를 같은 방식으로 더할 수 있기 때문에 상당히 편리한 방식이지요. 하지만 이 방식은 숫자를 표현하기 위해서 얼마나 많은 비트를 사용할 것인지 미리 결정해 두어야 한다는 단점이 있습니다.

3 (옮긴이) 이런 방식을 SD(Sign Digit) 표기 방식이라 이야기하고, 일부 특수한 경우에서는 약간 변형된 형태로 실제 사용되기도 합니다. 이 방식의 단점으로는 0에 대해서 +0과 -0의 서로 다른 표현이 존재하게 되어 부호에 대한 효율이 좀 떨어지고 양수와 음수를 같이 더하는 데 약간의 처리가 필요하다는 점이 있습니다.

음수의 표현: 십진수

-500을 표현하기 위해서 500을 사용합니다.

-499를 표현하기 위해서 501을 사용합니다.

-498을 표현하기 위해서 502를 사용합니다.

중간 생략

-2를 표현하기 위해서 998을 사용합니다.

-1을 표현하기 위해서 999를 사용합니다.

0을 표현하기 위해서 000을 사용합니다.

1을 표현하기 위해서 001을 사용합니다.

2를 표현하기 위해서 002을 사용합니다.

생략하고

497을 표현하기 위해서 497을 사용합니다.

498을 표현하기 위해서 498을 사용합니다.

499를 표현하기 위해서 499를 사용합니다.

-500 -499 -498 ... -4 -3 -2 -1 0 1 2 3 4 ... 497 498 499

500 501 502 ... 996 997 998 999 000 001 002 003 004 ... 497 498 499

10의 보수 (10's complement)

= 9의 보수 + 1

음수의 표현: 예제

-500 -499 -498 ... -4 -3 -2 -1 0 1 2 3 4 ... 497 498 499

500 501 502 ... 996 997 998 999 000 001 002 003 004 ... 497 498 499

은행 계좌에 잔고가 $143라고 가정해 봅시다. 계좌에서 $78를 인출했다고 생각하면, 이 말은 $143에 음수 $78를 더한 것과 마찬가지 입니다. -78은 10의 보수로 표현될 때 999 – 078 + 1이므로 922로 표현될 수 있습니다. 따라서 은행 잔고는 $143 + $922가 되며 이는 (자리올림을 무시할 것이므로) $65가 됩니다. 만일 추가로 $150를 인출했다면 -150을 더해야 하고 이는 10의 보수로 850이 됩니다. 따라서 은행 잔고는 065 더하기 850의 결과인 915가 됩니다. 이는 계좌의 잔고가 실제로 -$85라는 것을 의미하죠.

음수의 표현: 이진수

이진수	십진수
10000000	-128
10000001	-127
10000010	-126
10000011	-125
⋮	
11111101	-3
11111110	-2
11111111	-1
00000000	0
00000001	1
00000010	2
⋮	
01111100	124
01111101	125
01111110	126
01111111	127

이진수에서 위와 동일한 시스템이 바로 2의 보수를 사용하는 것입니다. 8비트 숫자를 가지고 연산을 한다고 가정해 봅시다. 가능한 수의 범위는 00000000에서 11111111까지이며 이는 십진수로 0에서 255의 범위가 됩니다. 하지만 음수를 표현하고 싶다면 아래 표에서 나타낸 것과 같이 1로 시작하는 8비트 숫자들을 이용해서 음수를 표현하면 됩니다.

숫자의 범위는 이제 -128에서 +127까지로 바뀌었습니다. 부호 있는 숫자를 나타낼 때 가장 왼쪽에 있는 비트(보통 MSB, most significant bit라 합니다)는 부호 비트라 이야기합니다. 이 비트가 1이 되면 음수를 나타내고 0인 경우에는 양수를 나타냅니다.[4]

2의 보수 (2's complement) = 1의 보수 + 1

2의 보수를 계산하기 위해서는 우선 1의 보수를 구하고 그 결과에 1을 더하면 됩니다. 이는 모든 비트를 반전시킨 후에 1을 더하는 것과 동일하지요. 예를 들어, 십진수 125는 01111101입니다. 2의 보수를 이용하여 -125를 표현하려면 일단 01111101의 모든 비트를 반전시켜 10000010으로 만들고, 그 다음 1을 더해서 10000011을 만듭니다. 이 결과는 앞에서 보신 표를 통하여 검증해 볼 수 있습니다. 다시 반대로 하더라도 마찬가지입니다. 즉, 모든 비트를 다시 반전시키고 1을 더하면 되는 것이지요.

2의 보수: 양수와 음수

이러한 시스템은 음수 기호를 따로 사용하지 않으면서도 양수와 음수를 모두 표현할 수 있는 방법을 제공해 줍니다. 게다가 덧셈의 규칙만을 이용하여 아주 간편하게 양수와 음수를 더할 수 있도록 해줍니다. 예를 들어 -127과 124에 해당하는 이진수들을 더하는 경우를 살펴봅시다. 앞에서 보신 표를 참고해서 쉽게 이진수들을 다음과 같이 표현할 수 있습니다.

$$\begin{array}{r} 10000001 \\ +\ 01111100 \\ \hline 11111101 \end{array}$$

이 결과는 십진수의 -3과 같은 값입니다.

2의 보수: 오버플로/언더플로 (overflow/underflow)

여기서 오버플로와 언더플로를 발생시키는 조건에 대해서는 주의를 기울이실 필요가 있습니다. 즉, 덧셈의 결과가 127보다 커지는 경우 혹은 -128보다 작아지는 경우라 할 수 있습니다. 예를 들어, 125 자신을 더하는 경우를 생각해 볼까요.

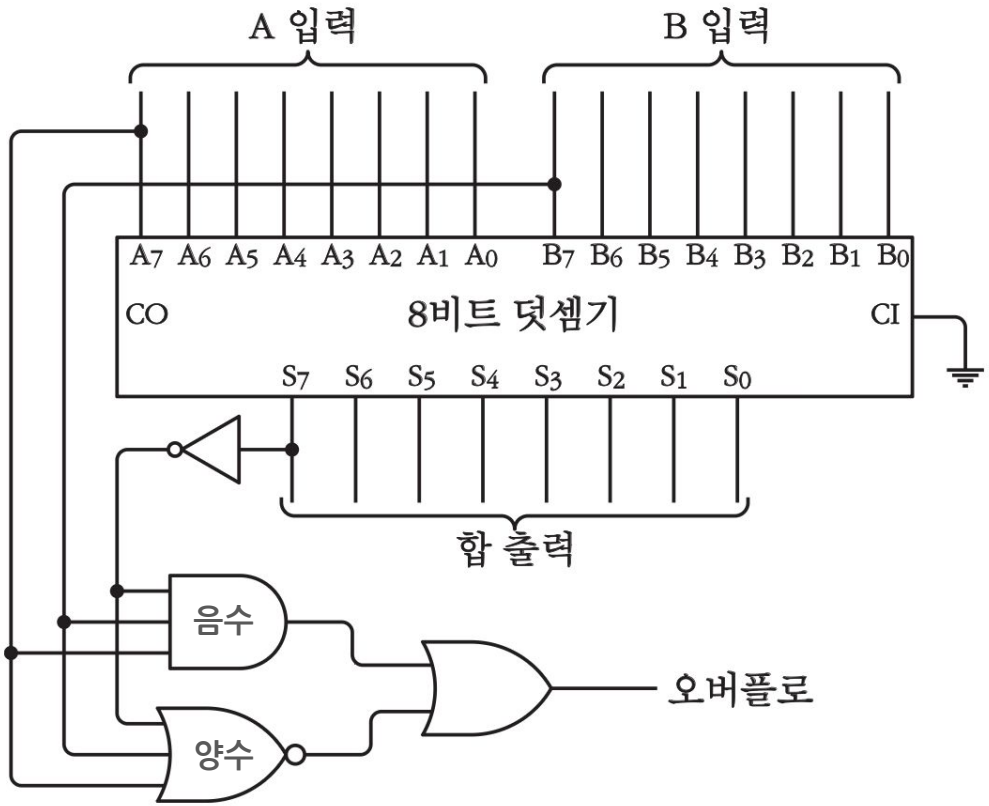
$$\begin{array}{r} 01111101 \\ +\ 01111101 \\ \hline 11111010 \end{array}$$

최상위 비트가 1이 되었으므로 이 결과는 음수로 해석될 수 있으며, 이진수로 나타난 수는 -6에 해당하지요. 이와 비슷한 현상이 -125끼리 더했을 때도 발생합니다.

$$\begin{array}{r} 10000011 \\ +\ 10000011 \\ \hline 100000110 \end{array}$$

8비트 숫자를 이용하겠다고 결정한 상태이기 때문에 가장 왼쪽의 비트는 무시되어야 합니다. 따라서 남은 8비트의 값은 십진수 6과 동일합니다.

2의 보수: 가산기


A 입력
B 입력
A7 A6 A5 A4 A3 A2 A1 A0
B7 B6 B5 B4 B3 B2 B1 B0
CO
8비트 덧셈기
CI
S7 S6 S5 S4 S3 S2 S1 S0
합 출력
음수
양수
오버플로

부호가 있는/없는 (signed/unsigned)

지금까지 이진수를 사용하는 다른 두 가지 방법에 대해서 배웠습니다. 즉, 이진수는 부호가 있는 숫자(signed)를 표현할 수도 있고 부호가 없는(unsigned) 숫자를 표현할 수도 있습니다.

부호가 없는 8비트 숫자는 0에서 255까지의 범위를 가지고 있으며, 부호가 있는 8비트 숫자는 -128에서 127까지를 표현할 수 있습니다. 하지만, 이진수 자체로는 해당 숫자가 부호 있는 숫자인지, 부호가 없는 숫자인지 판단할 수 있는 방법이 없습니다. 예를 들어 "10110110이라는 8비트 숫자가 어떤 값을 가지지요? 10진수로는 어떻게 나타낼 수 있을까요?"라는 질문에 대하여 "부호 있는 숫자인가요? 아니면 부호가 없는 숫자인가요? -74 혹은 182 모두 될 수 있습니다."라고 답할 수밖에 없는 것이지요.

이러한 부분이 어찌 보면 비트 사용의 어려움이지요. 비트는 0과 1만을 나타내며 그 자체로 어떤 것을 나타내는지는 알려주지 않으니까요.

부호가 있는/없는 정수 (signed/unsigned integer)

정수의 크기	부호 없는 정수의 범위	부호 있는 정수의 범위
8비트	0~255	−128~127
16비트	0~65,535	−32,768~32,767
32비트	0~4,294,967,295	−2,147,483,648~2,147,483,647
64비트	0~18,446,744,073,709,551,615	−9,223,372,036,854,775,808~ 9,223,372,036,854,775,807