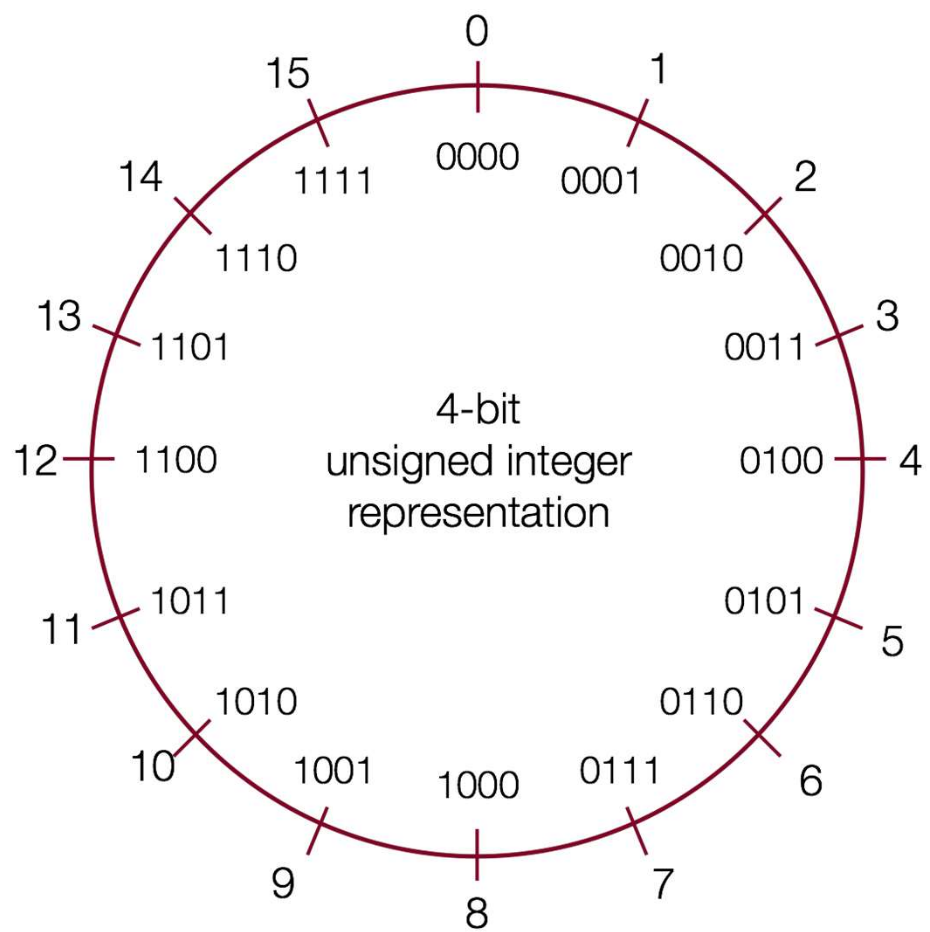
Unsigned Integers

For positive (unsigned) integers, there is a 1-to-1 relationship between the decimal representation of a number and its binary representation. If you have a 4-bit number, there are 16 possible combinations, and the unsigned numbers go from 0 to 15:

`0b0000 = 0`	`0b0001 = 1`	`0b0010 = 2`	`0b0011 = 3`
`0b0100 = 4`	`0b0101 = 5`	`0b0110 = 6`	`0b0111 = 7`
`0b1000 = 8`	`0b1001 = 9`	`0b1010 = 10`	`0b1011 = 11`
`0b1100 = 12`	`0b1101 = 13`	`0b1110 = 14`	`0b1111 = 15`

The range of an unsigned number is $0 \rightarrow 2^w - 1$, where w is the number of bits in our integer. For example, a 32-bit `int` can represent numbers from 0 to $2^{32} - 1$, or 0 to 4,294,967,295.

Unsigned Integers



How to Represent A Signed Value

A **signed** integer is a negative, 0, or positive integer.

How can we represent both negative *and* positive numbers in binary?

Signed Integers

- A **signed** integer is a negative integer, 0, or a positive integer.
- *Problem:* How can we represent negative *and* positive numbers in binary?

Idea: let's reserve the *most significant bit* to store the sign.

Sign Magnitude Representation

$$\underbrace{0}_{\text{positive}}\underbrace{110}_{6}$$

$$\underbrace{1}_{\text{negative}}\underbrace{011}_{3}$$

Sign Magnitude Representation

$$\underbrace{0}_{\text{positive}}\underbrace{000}_{0}$$

$$\underbrace{1}_{\text{negative}}\underbrace{000}_{0}$$

Sign Magnitude Representation

1 000 = -0	0 000 = 0
1 001 = -1	0 001 = 1
1 010 = -2	0 010 = 2
1 011 = -3	0 011 = 3
1 100 = -4	0 100 = 4
1 101 = -5	0 101 = 5
1 110 = -6	0 110 = 6
1 111 = -7	0 111 = 7

- We’ve only represented 15 of our 16 available numbers!

Sign Magnitude Representation AKA Ones Complement

- **Pro:** easy to represent, and easy to convert to/from decimal.
- **Con:** +-0 is not intuitive
- **Con:** we lose a bit that could be used to store more numbers
- **Con:** arithmetic is tricky: we need to find the sign, then maybe subtract (borrow and carry, etc.), then maybe change the sign. This complicates the hardware support for something as fundamental as addition.

Can we do better?

Now Lets Try a Better Approach!

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

$$\begin{array}{r} 0101 \\ +\,{\color{red}????} \\ \hline 0000 \end{array}$$

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

$$
\begin{array}{r}
0101 \\
+\ {\color{red}1011} \\
\hline
0000
\end{array}
$$

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

$$\begin{array}{r} 0011 \\ +\ ???? \\ \hline 0000 \end{array}$$

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

$$
\begin{array}{r}
0011 \\
+\,1101 \\
\hline
0000
\end{array}
$$

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

```
  0000
+ ????
------
  0000
```

A Better Idea

- Ideally, binary addition would *just work* **regardless** of whether the number is positive or negative.

$$
\begin{array}{r}
0000 \\
+\,{\color{red}0000} \\
\hline
0000
\end{array}
$$

There Seems Like a Pattern Here…

$$
\begin{array}{r}
0101 \\
+\,1011 \\
\hline
0000
\end{array}
\qquad
\begin{array}{r}
0011 \\
+\,1101 \\
\hline
0000
\end{array}
\qquad
\begin{array}{r}
0000 \\
+\,0000 \\
\hline
0000
\end{array}
$$

- The negative number is the positive number **inverted, plus one!**

There Seems Like a Pattern Here…

A binary number plus its inverse is all 1s.

$$\begin{array}{r} 0101 \\ +\ 1010 \\ \hline 1111 \end{array}$$

Add 1 to this to carry over all 1s and get 0!

$$\begin{array}{r} 1111 \\ +\ 0001 \\ \hline 0000 \end{array}$$

Another Trick

- To find the negative equivalent of a number, work right-to-left and write down all digits *through* when you reach a 1. Then, invert the rest of the digits.

$$
\begin{array}{r}
100100 \\
+\;{\color{red}??????} \\
\hline
000000
\end{array}
$$

Another Trick

- To find the negative equivalent of a number, work right-to-left and write down all digits *through* when you reach a 1. Then, invert the rest of the digits.

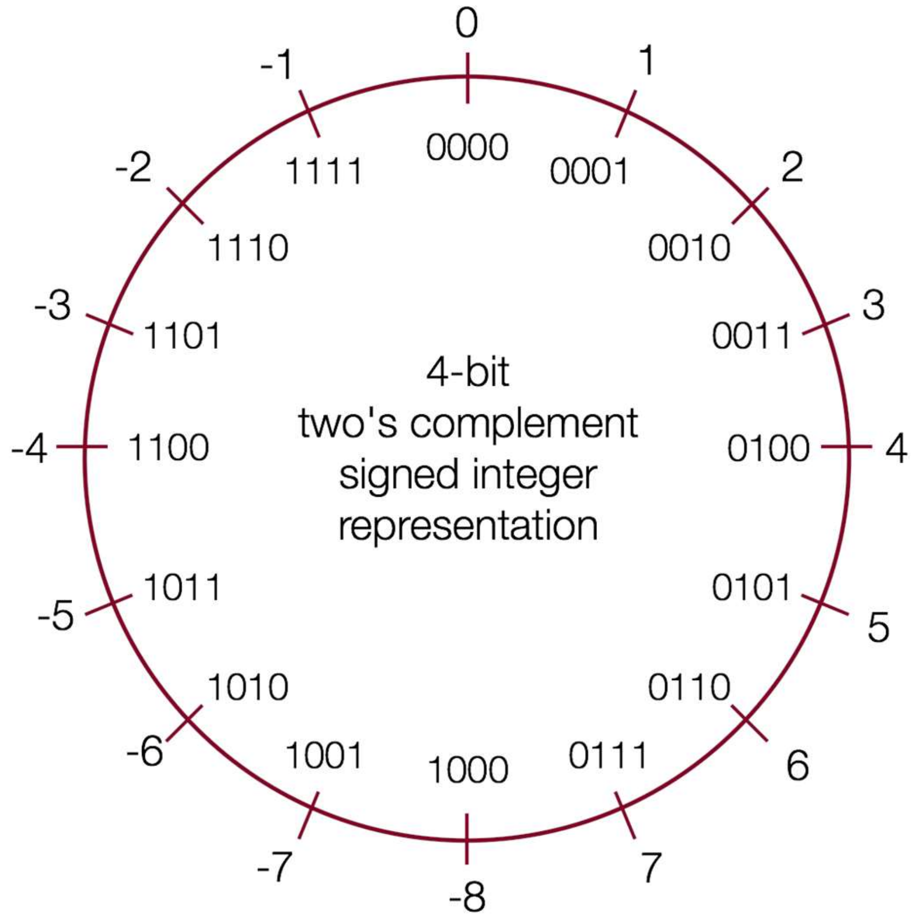
$$
\begin{array}{r}
100100 \\
+\;???100 \\
\hline
000000
\end{array}
$$

Another Trick

- To find the negative equivalent of a number, work right-to-left and write down all digits *through* when you reach a 1. Then, invert the rest of the digits.

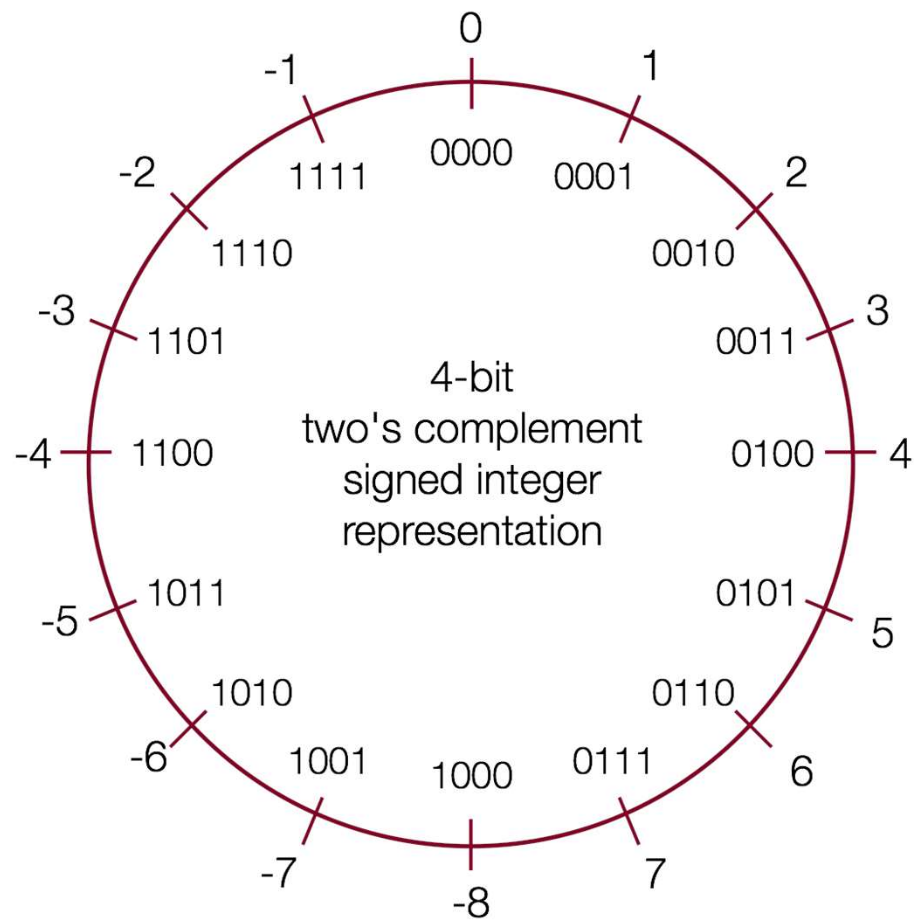
$$
\begin{array}{r}
100100 \\
+\ \underline{011100} \\
000000
\end{array}
$$

Two’s Complement



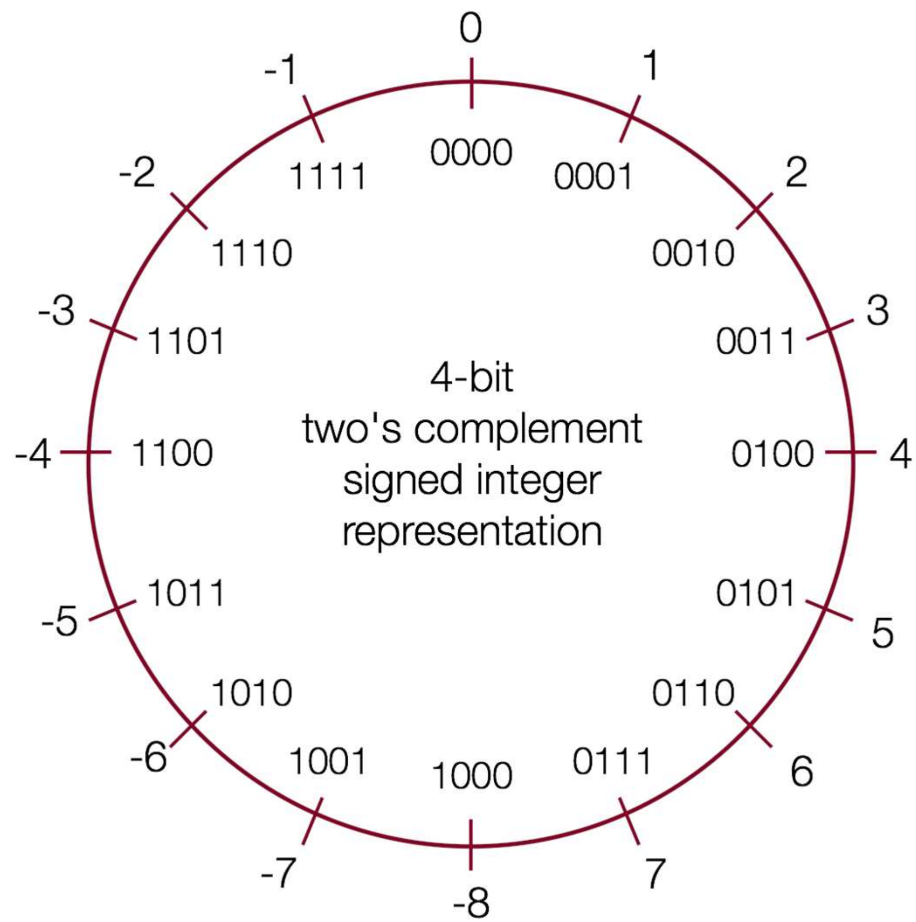
Two’s Complement

- In **two’s complement**, we represent a positive number as **itself**, and its negative equivalent as the **two’s complement of itself.**
- The **two’s complement** of a number is the binary digits inverted, plus 1.
- This works to convert from positive to negative, **and** back from negative to positive!



Two’s Complement

- **Con:** more difficult to represent, and difficult to convert to/from decimal and between positive and negative.
- **Pro:** only 1 representation for 0!
- **Pro:** all bits are used to represent as many numbers as possible
- **Pro:** the most significant bit still indicates the sign of a number.
- **Pro:** addition works for any combination of positive and negative!



Two's Complement

- Adding two numbers is just…adding! There is no special case needed for negatives. E.g. what is 2 + -5?

	0010	2
+	1011	-5
	1101	-3

Two's Complement

- Subtracting two numbers is just performing the two's complement on one of them and then adding. E.g. 4 – 5 = -1.

$$\begin{array}{rr} & 0100 \\ - & 0101 \\ \hline \end{array} \qquad \begin{array}{r} 4 \\ 5 \end{array} \qquad \longrightarrow \qquad \begin{array}{rr} & 0100 \\ + & 1011 \\ \hline & 1111 \end{array} \qquad \begin{array}{r} 4 \\ -5 \\ -1 \end{array}$$

How to Read Two's Complement #s

- Multiply the most significant bit by -1 and multiply all the other bits by 1 as normal

$$\underline{1}\quad\underline{1}\quad\underline{1}\quad\underline{0}$$

$$2^3\quad 2^2\quad 2^1\quad 2^0$$

$= \mathbf{1}*-8 + \mathbf{1}*4 + \mathbf{1}*2 + \mathbf{0}*1 = -2$

How to Read Two's Complement #s

- Multiply the most significant bit by -1 and multiply all the other bits by 1 as normal

$$\underline{0}\quad \underline{1}\quad \underline{1}\quad \underline{0}$$

$$2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$= \mathbf{0}*-8 + \mathbf{1}*4 + \mathbf{1}*2 + \mathbf{0}*1 = 6$$

Practice: Two’s Complement

What are the negative or positive equivalents of the numbers below?

a) -4 (1100)

b) 7 (0111)

c) 3 (0011)

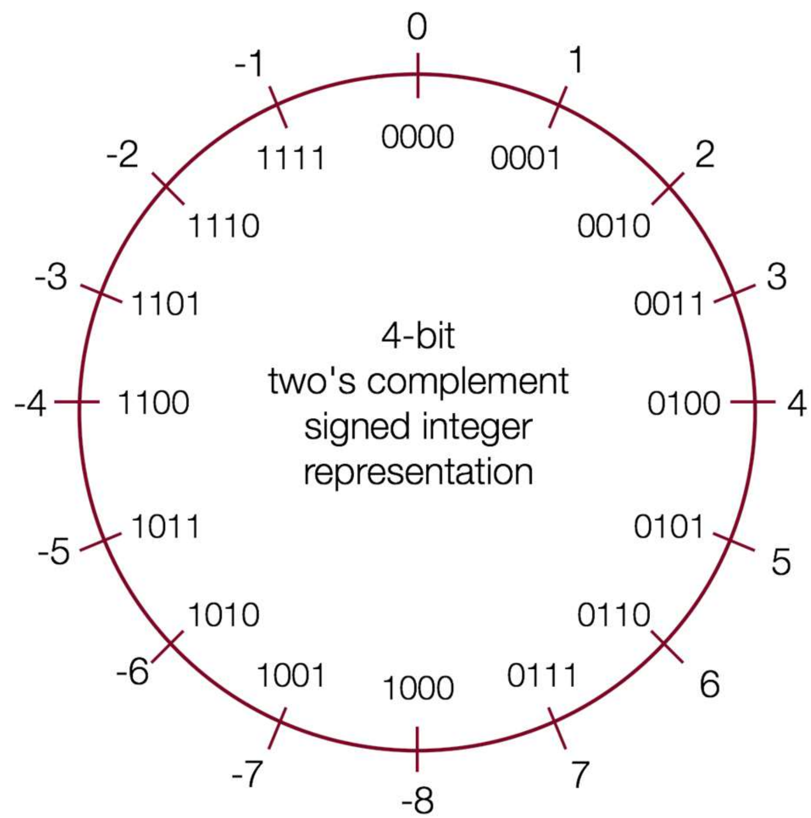
Practice: Two’s Complement

What are the negative or positive equivalents of the numbers below?

a) -4 (1100) -> 4 (0100)

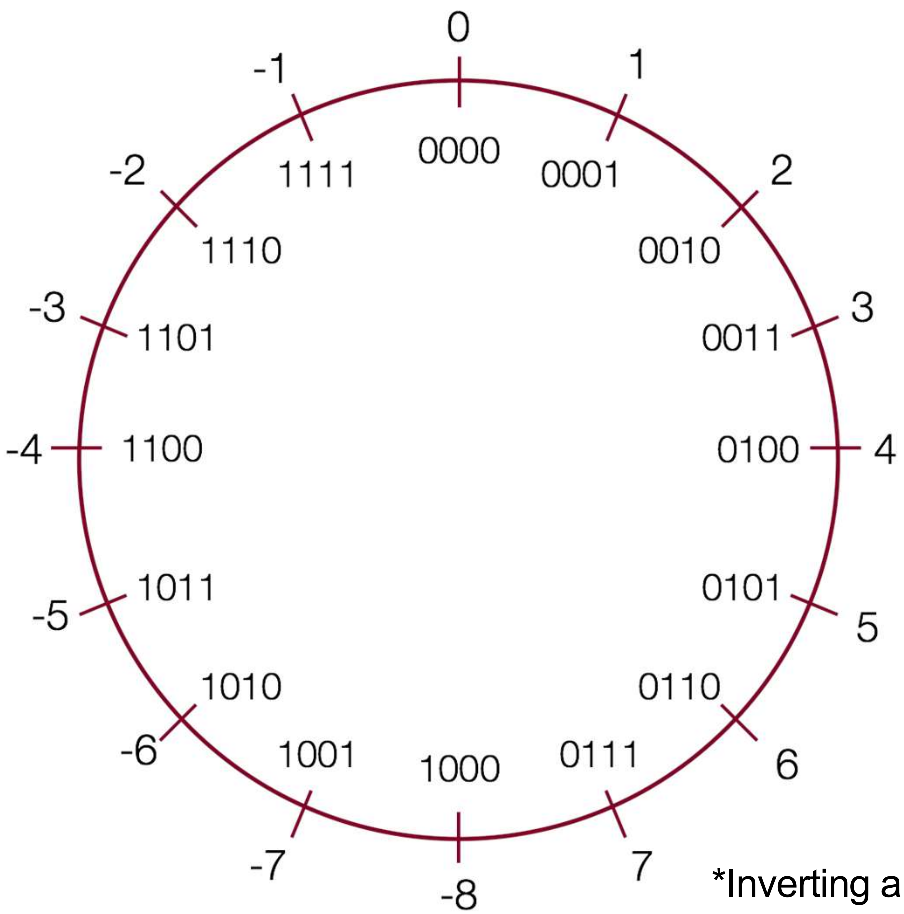
b) 7 (0111) -> (1001)

c) 3 (0011) -> (1101)



Some Extra Slides for Review

Two's Complement



In practice, a negative number in two's complement is obtained by inverting all the bits of its positive counterpart*, and then adding 1, or: `x = ~x + 1`

Example: The number 2 is represented as normal in binary: 0010

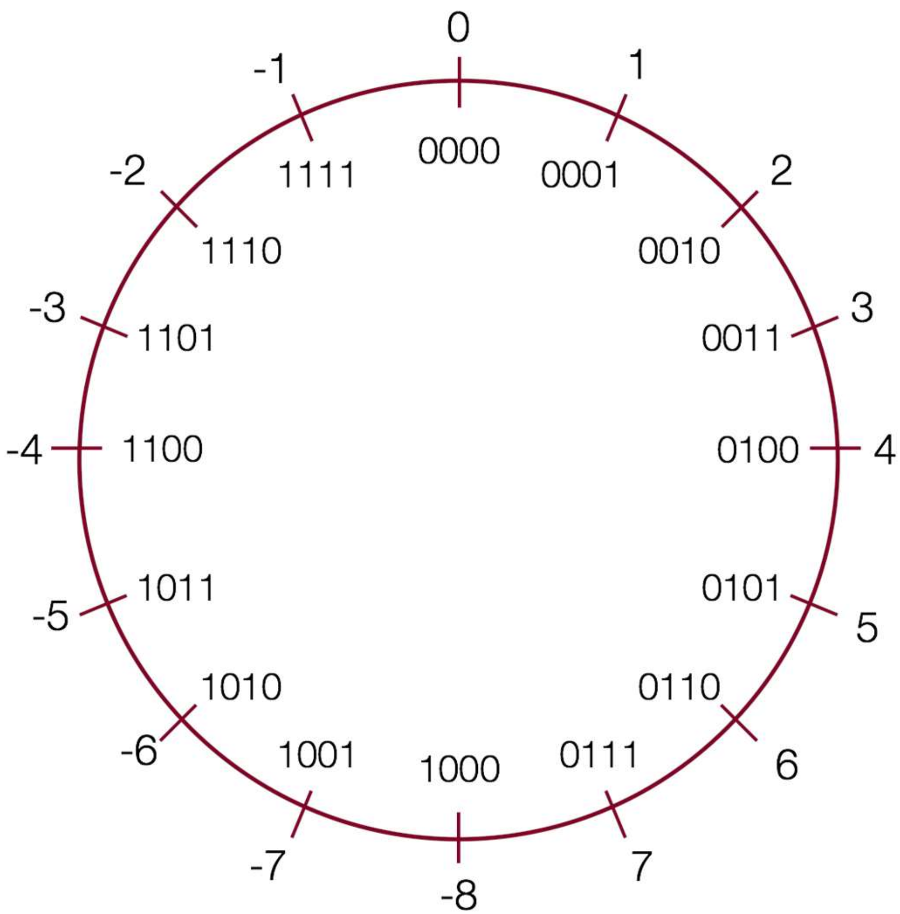
-2 is represented by inverting the bits, and adding 1:

0010 ☞ 1101

$$\begin{array}{r} 1101 \\ +\quad 1 \\ \hline 1110 \end{array}$$

*Inverting all the bits of a number is its "one's complement"

Two's Complement



To convert a negative number to a positive number, perform the same steps!

Example: The number -5 is represented in two's complements as: 1011

5 is represented by inverting the bits, and adding 1:

1011 ☞ 0100

```
 0100
+   1
-----
 0101
```

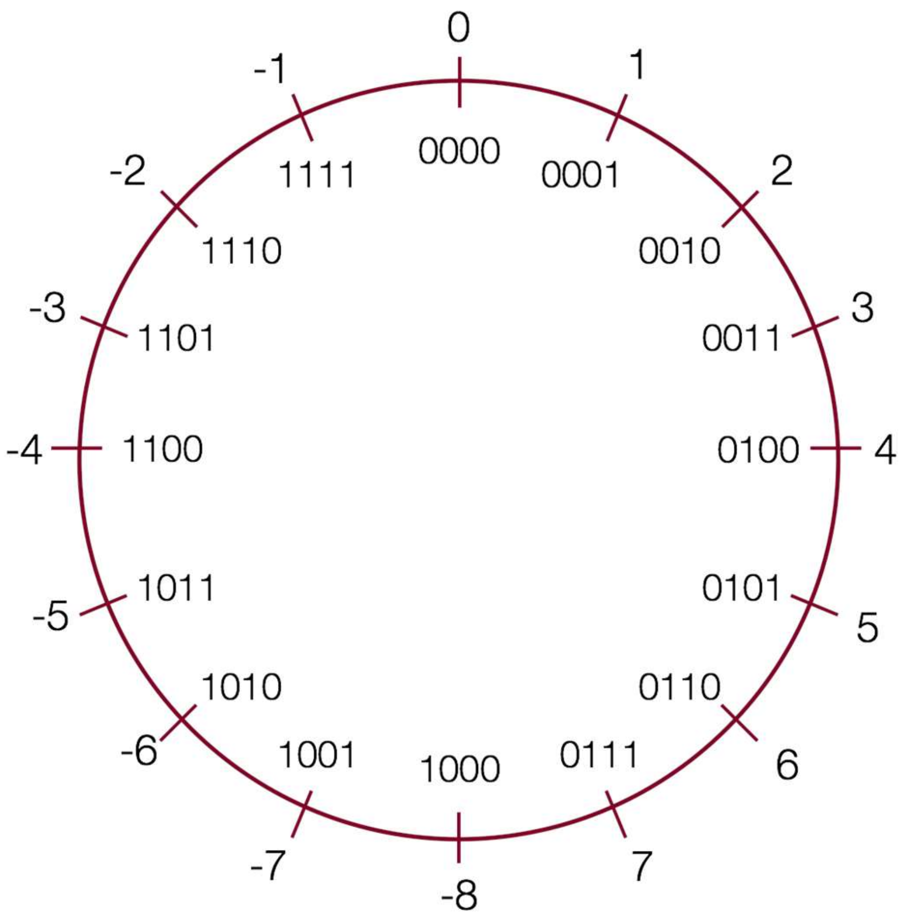
Shortcut: start from the right, and write down numbers until you get to a 1:

1

Now invert all the rest of the digits:

0101

Two's Complement: Neat Properties

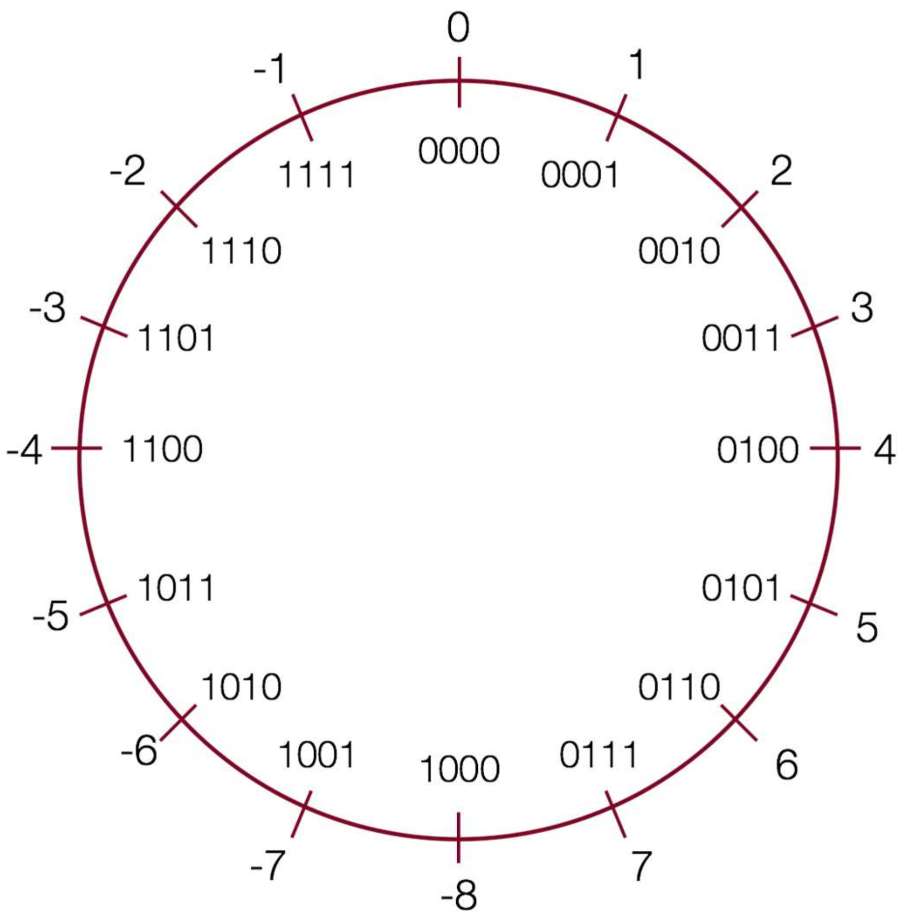


There are a number of useful properties associated with two's complement numbers:

1. There is only one zero (yay!)
2. The highest order bit (left-most) is 1 for negative, 0 for positive (so it is easy to tell if a number is negative)
3. Adding two numbers is just…adding! Example:
   2 + -5 = -3

  0010 ☞ 2
<u>+1011</u> ☞ -5
  1101 ☞ -3 decimal (wow!)

Two's Complement: Neat Properties



More useful properties:

4. Subtracting two numbers is simply performing the two's complement on one of them and then adding.
   Example:
   4 - 5 = -1

0100 ☞ 4, 0101 ☞ 5

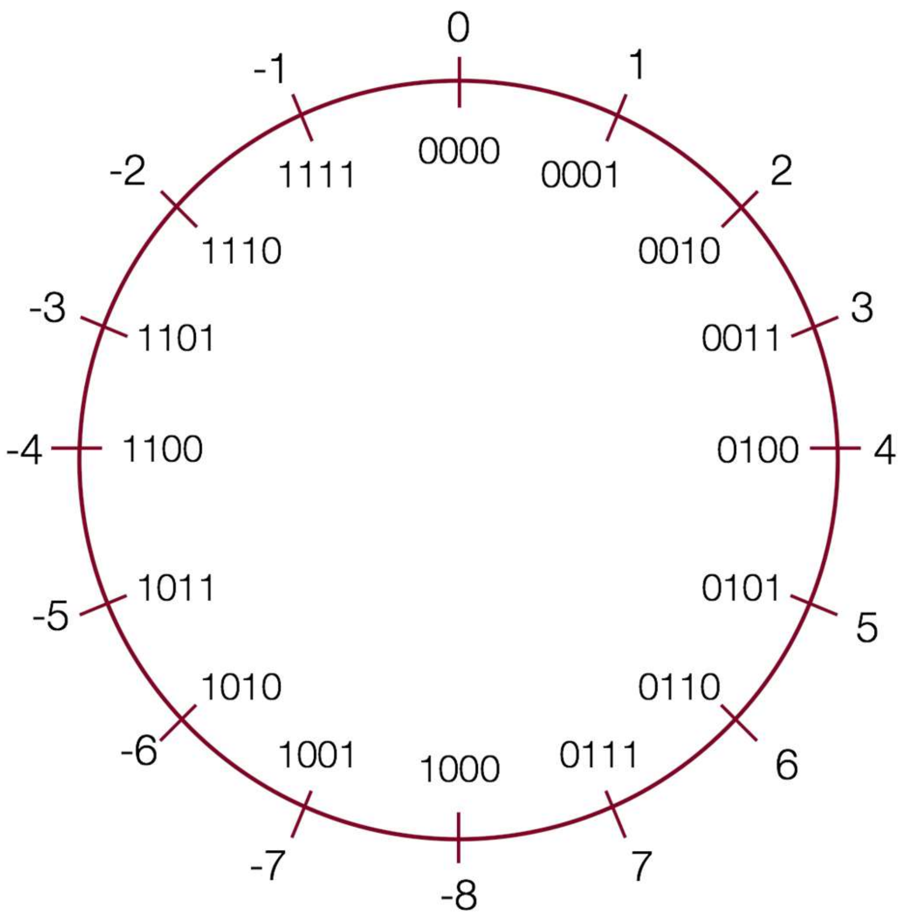
Find the two's complement of 5: 1011
add:

0100 ☞ 4

+1011 ☞ -5

1111 ☞ -1 decimal

Practice



Convert the following 4-bit numbers from positive to negative, or from negative to positive using two's complement notation:

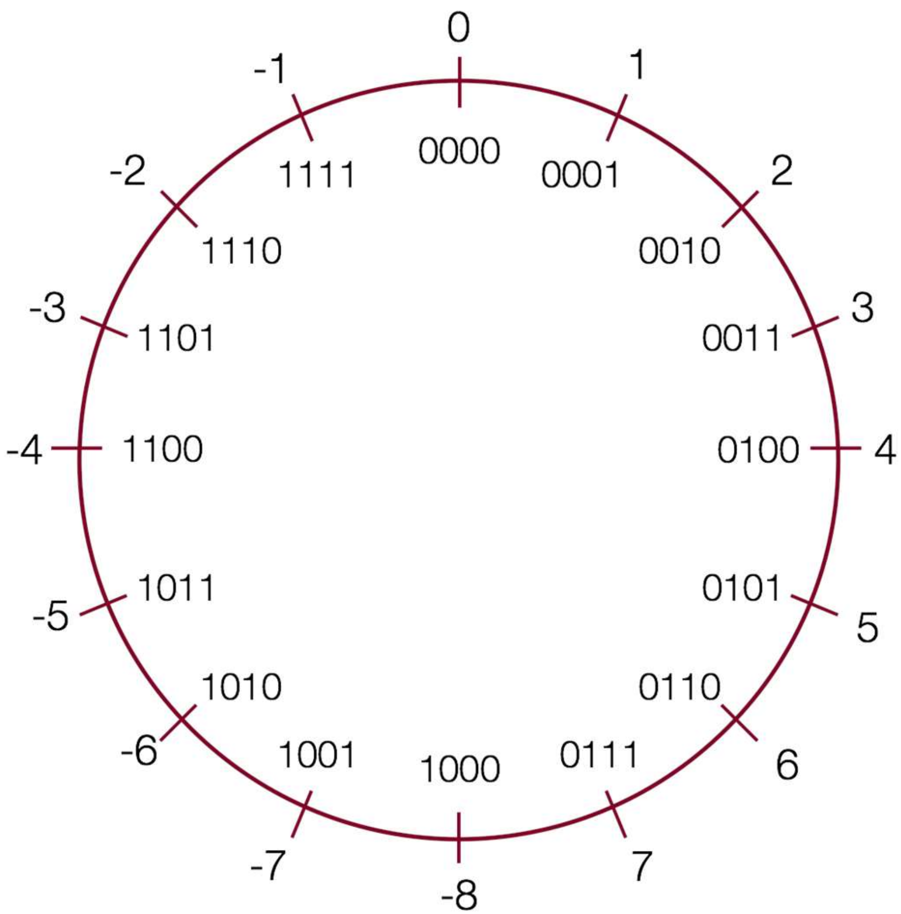
a. -4 (1100) ☞

b. 7 (0111) ☞

c. 3 (0011) ☞

d. -8 (1000) ☞

Practice



Convert the following 4-bit numbers from positive to negative, or from negative to positive using two's complement notation:

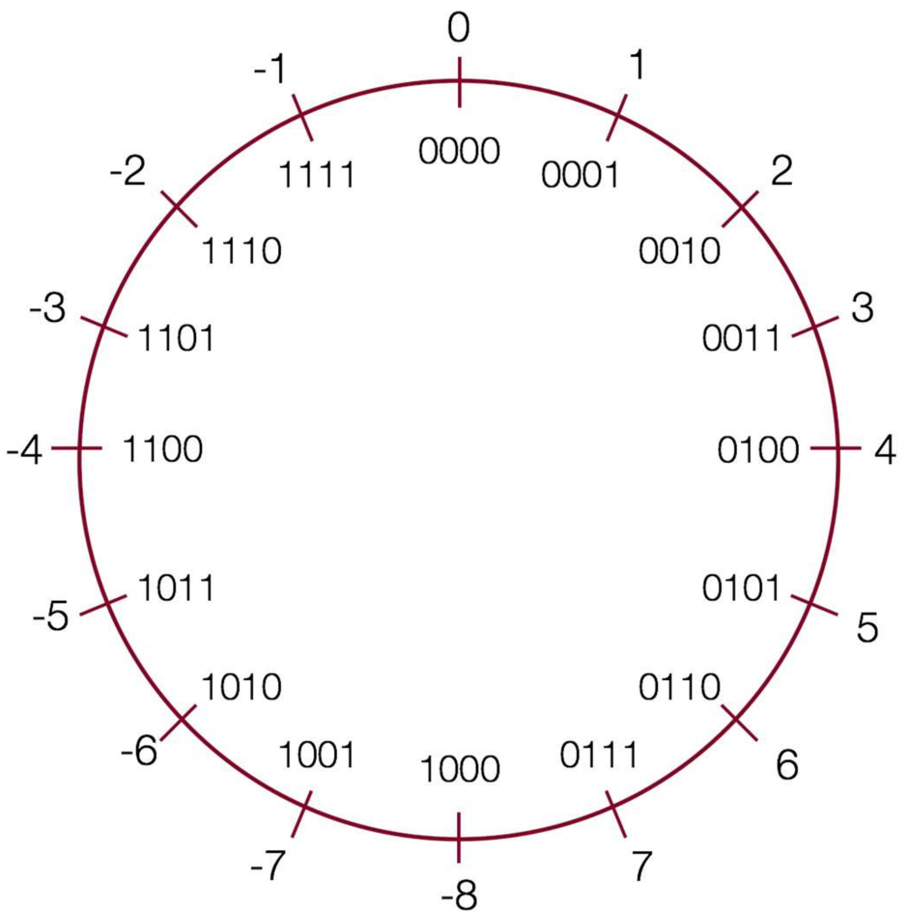
a. -4 (1100) ☞ 0100

b. 7 (0111) ☞ 1001

c. 3 (0011) ☞ 1101

d. -8 (1000) ☞ 1000 (?! If you look at the chart, +8 cannot be represented in two's complement with 4 bits!)

Practice



Convert the following 8-bit numbers from positive to negative, or from negative to positive using two's complement notation:

a. -4 (11111100) ☞ 00000100

b. 27 (00011011) ☞ 11100101

c. -127 (10000001) ☞ 01111111

d. 1 (00000001) ☞ 11111111

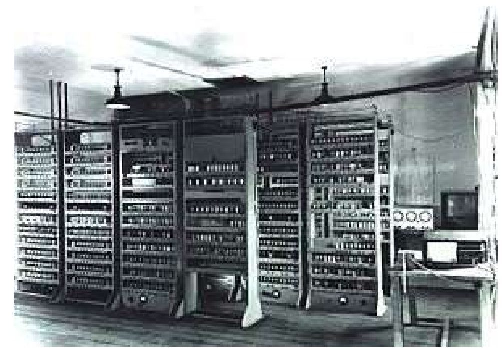
History: Two's complement

- The binary representation was first proposed by John von Neumann in *First Draft of a Report on the EDVAC* (1945)
  - That same year, he also invented the merge sort algorithm
- Many early computers used sign-magnitude or one's complement

```
+7    0b0000 0111
-7    0b1111 1000
```

8-bit one's complement

- The System/360, developed by IBM in 1964, was widely popular (had 1024KB memory) and established two's complement as the dominant binary representation of integers



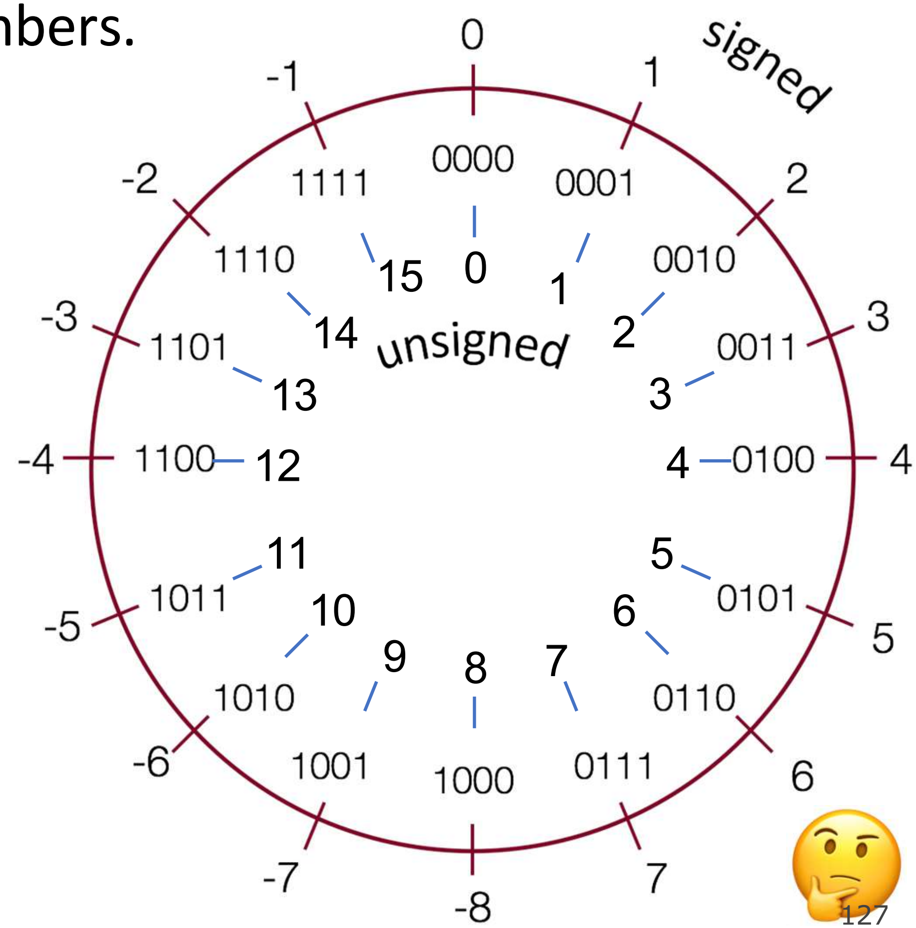
EDSAC (1949)

System/360 (1964)

Overflow

- What is happening here? Assume 4-bit numbers.

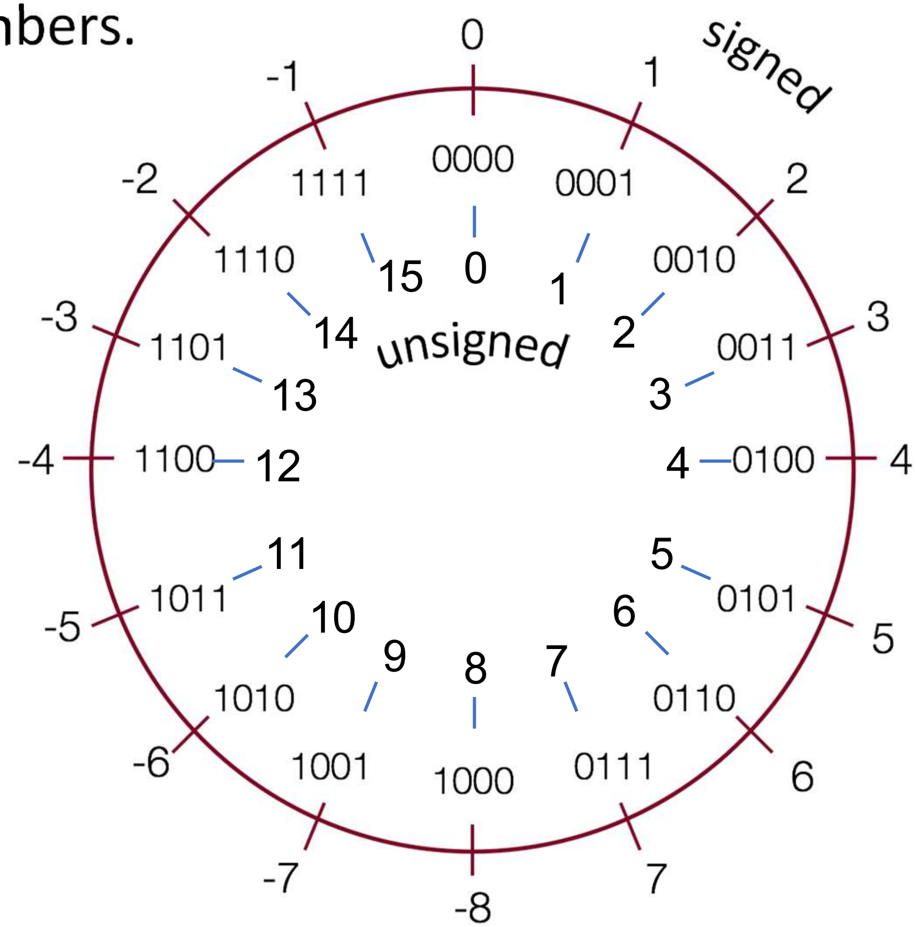
```
  0b1101
+ 0b0100
```



Overflow

- What is happening here? Assume 4-bit numbers.

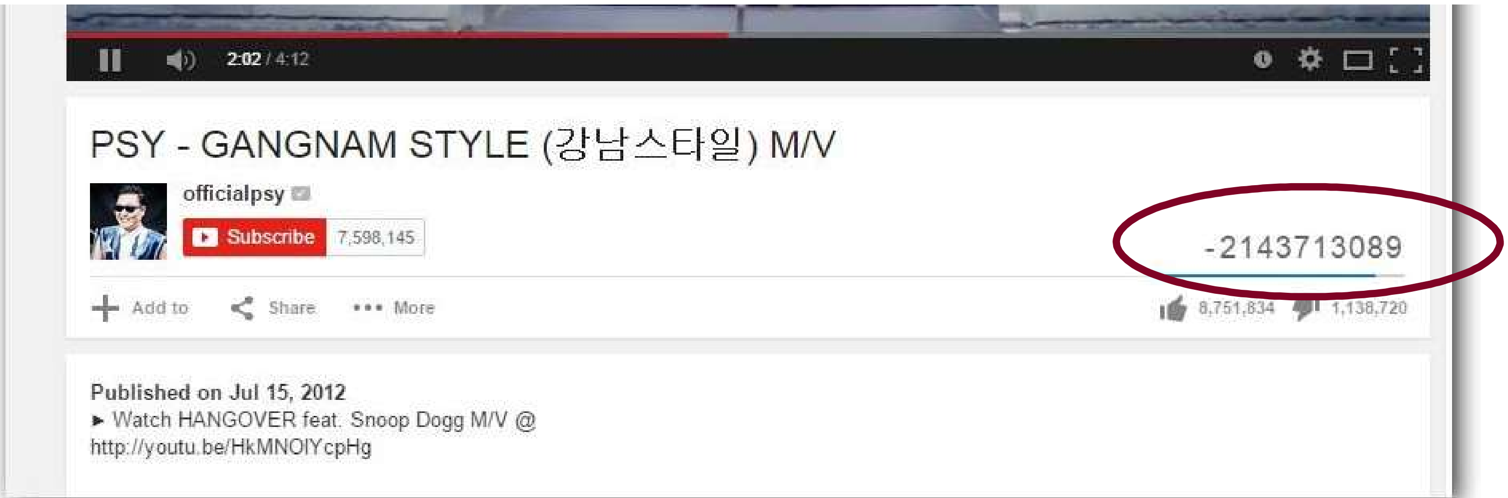
```
  0b1101
+ 0b0100
```



Signed	Unsigned
-3 + 4 = 1	13 + 4 = 1
No overflow	Overflow

Overflow in Signed Addition

Signed overflow wraps around to the negative numbers:



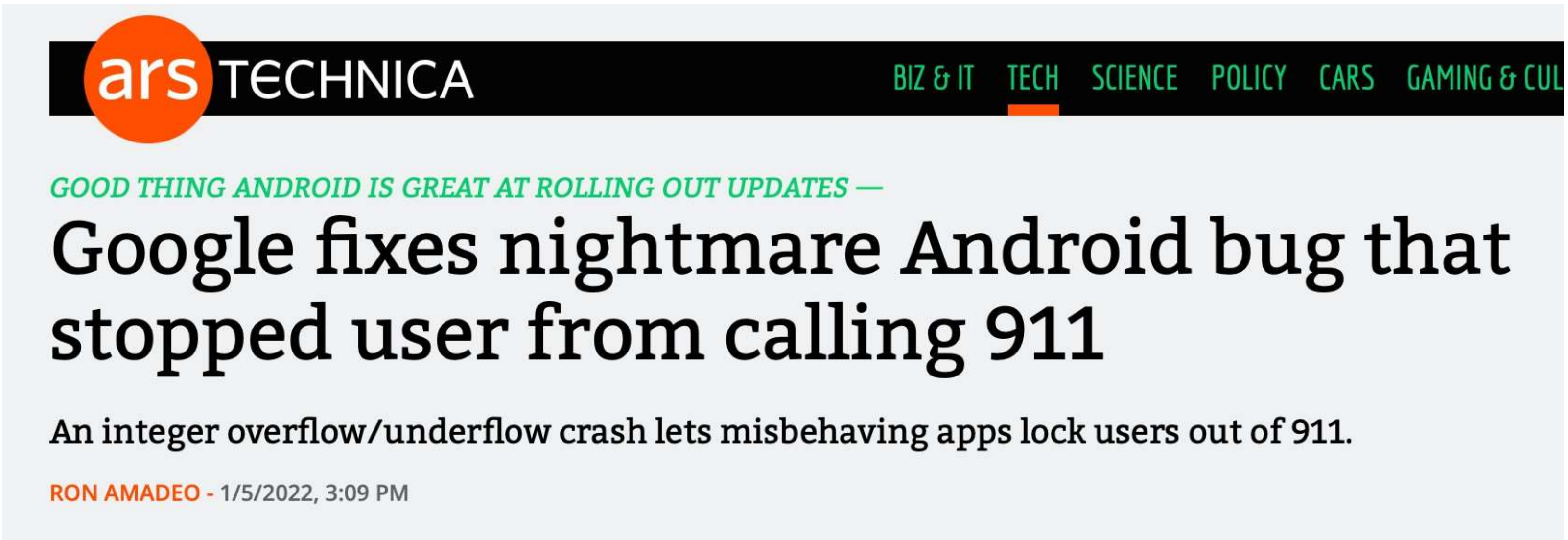
YouTube fell into this trap — their view counter was a signed, 32-bit int. They fixed it after it was noticed, but for a while, the view count for Gangnam Style (the first video with over `INT_MAX` number of views) was negative.

Overflow In Practice: PSY

YouTube: “We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views), but that was before we met PSY. "Gangnam Style" has been viewed so many times we had to upgrade to a 64-bit integer (9,223,372,036,854,775,808)!”

Overflow in Signed Addition

In the news on January 5, 2022 (!):



ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CUL

GOOD THING ANDROID IS GREAT AT ROLLING OUT UPDATES —

Google fixes nightmare Android bug that stopped user from calling 911

An integer overflow/underflow crash lets misbehaving apps lock users out of 911.

RON AMADEO - 1/5/2022, 3:09 PM

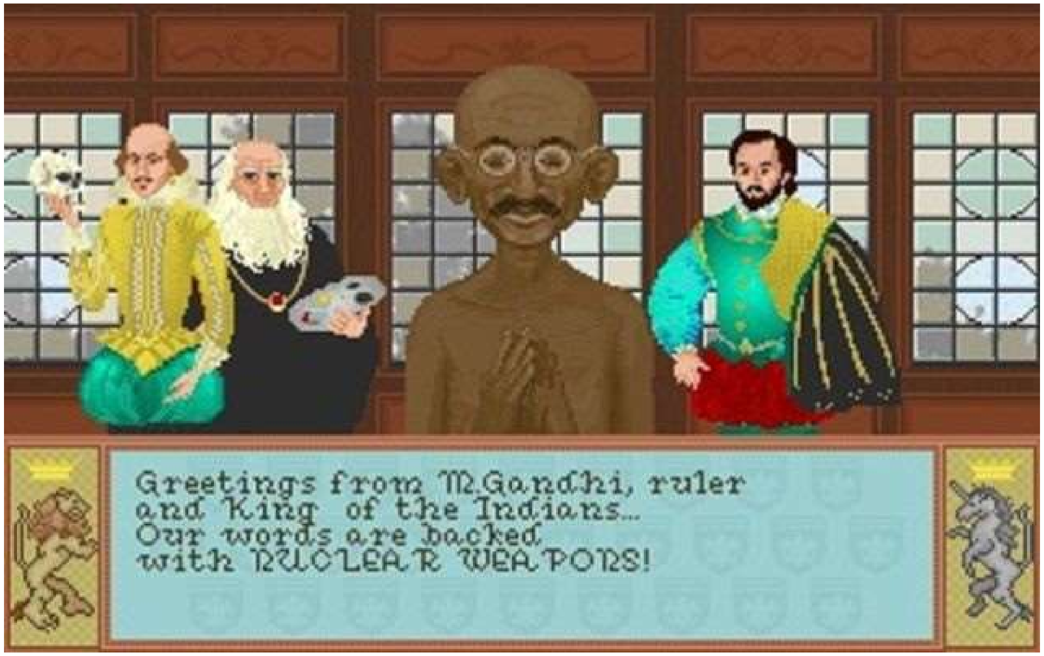
https://arstechnica.com/gadgets/2022/01/google-fixes-nightmare-android-bug-that-stopped-user-from-calling-911/

Overflow In Practice: Timestamps

- Many systems store timestamps as **the number of seconds since Jan. 1, 1970** in a **signed 32-bit integer**.
- **Problem:** the latest timestamp that can be represented this way is 3:14:07 UTC on Jan. 13 2038!

Overflow In Practice: Gandhi

- In the game “Civilization”, each civilization leader had an “aggression” rating. Gandhi was meant to be peaceful, and had a score of 1.
- If you adopted “democracy”, all players’ aggression reduced by 2. Gandhi’s went from 1 to **255**!
- Gandhi then became a big fan of nuclear weapons.



https://kotaku.com/why-gandhi-is-such-an-asshole-in-civilization-1653818245

Overflow in Practice:

- Pacman Level 256
- Make sure to reboot Boeing Dreamliners every 248 days
- Comair/Delta airline had to cancel thousands of flights days before Christmas
- Reported vulnerability CVE-2019-3857 in libssh2 may allow a hacker to remotely execute code
- Donkey Kong Kill Screen