



CS107 Lecture 3

Bits and Bytes, Integer Representations, Overflow

Reading: Bryant & O'Hallaron, Ch. 2.2-2.3 (skim)

Ed Discussion: <https://edstem.org/us/courses/28214/discussion/1842423>

Overflow

If you exceed the **maximum** value of your bit representation, you *wrap around* or *overflow* back to the **smallest** bit representation.

$$0b1111 + 0b1 = 0b0000$$

$$0b1111 + 0b10 = 0b0001$$

If you go below the **minimum** value of your bit representation, you *wrap around* or *overflow* (or rather, *underflow*) back to the **largest** bit representation.

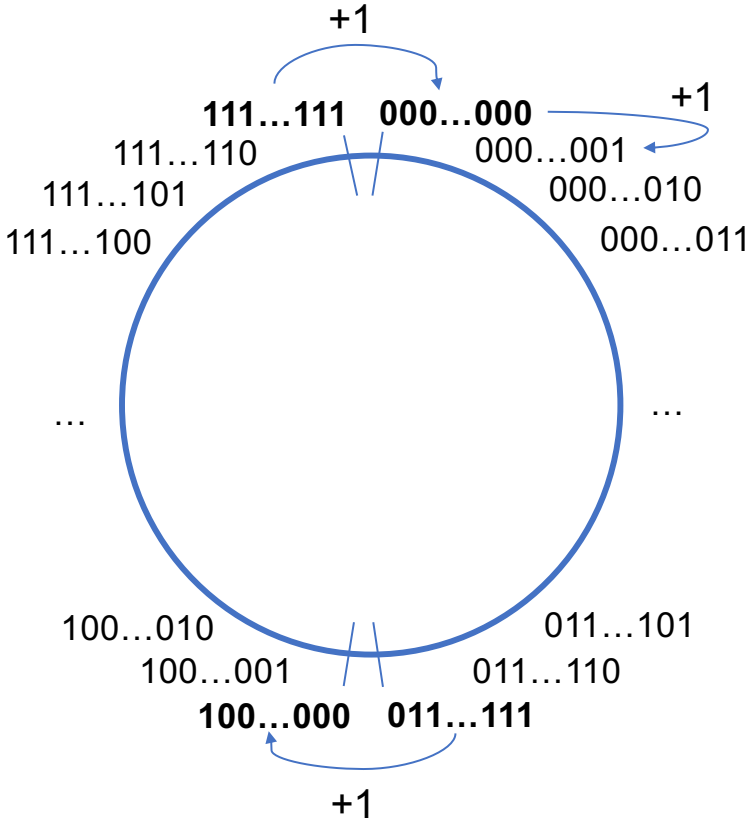
$$0b0000 - 0b1 = 0b1111$$

$$0b0000 - 0b10 = 0b1110$$

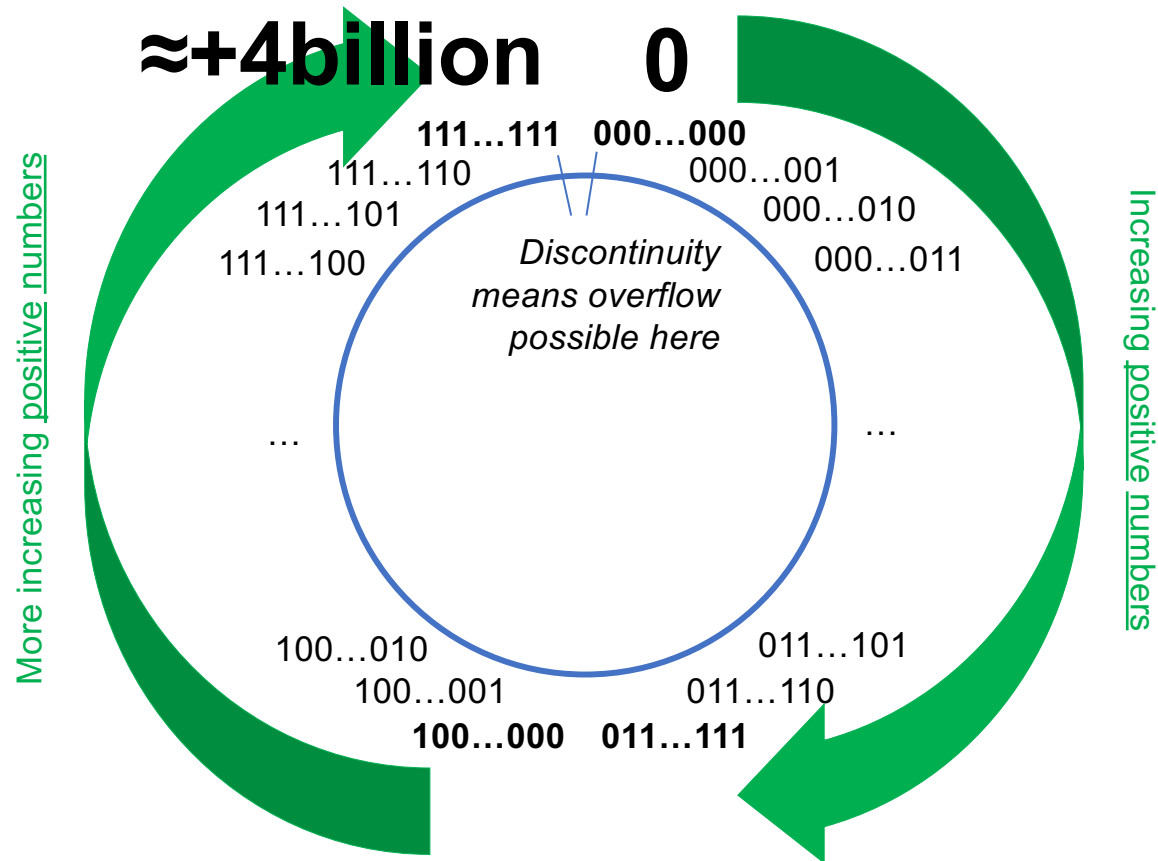
Min and Max Integer Values

Type	Size	Minimum	Maximum
char	1	-128 (SCHAR_MIN)	127 (SCHAR_MAX)
unsigned char	1	0	255 (UCHAR_MAX)
Separator			
short	2	-32768 (SHRT_MIN)	32767 (SHRT_MAX)
unsigned short	2	0	65535 (USHRT_MAX)
Separator			
int	4	-2147483648 (INT_MIN)	2147483647 (INT_MAX)
unsigned int	4	0	4294967295 (UINT_MAX)
Separator			
long	8	-9223372036854775808 (LONG_MIN)	9223372036854775807 (LONG_MAX)
unsigned long	8	0	18446744073709551615 (ULONG_MAX)

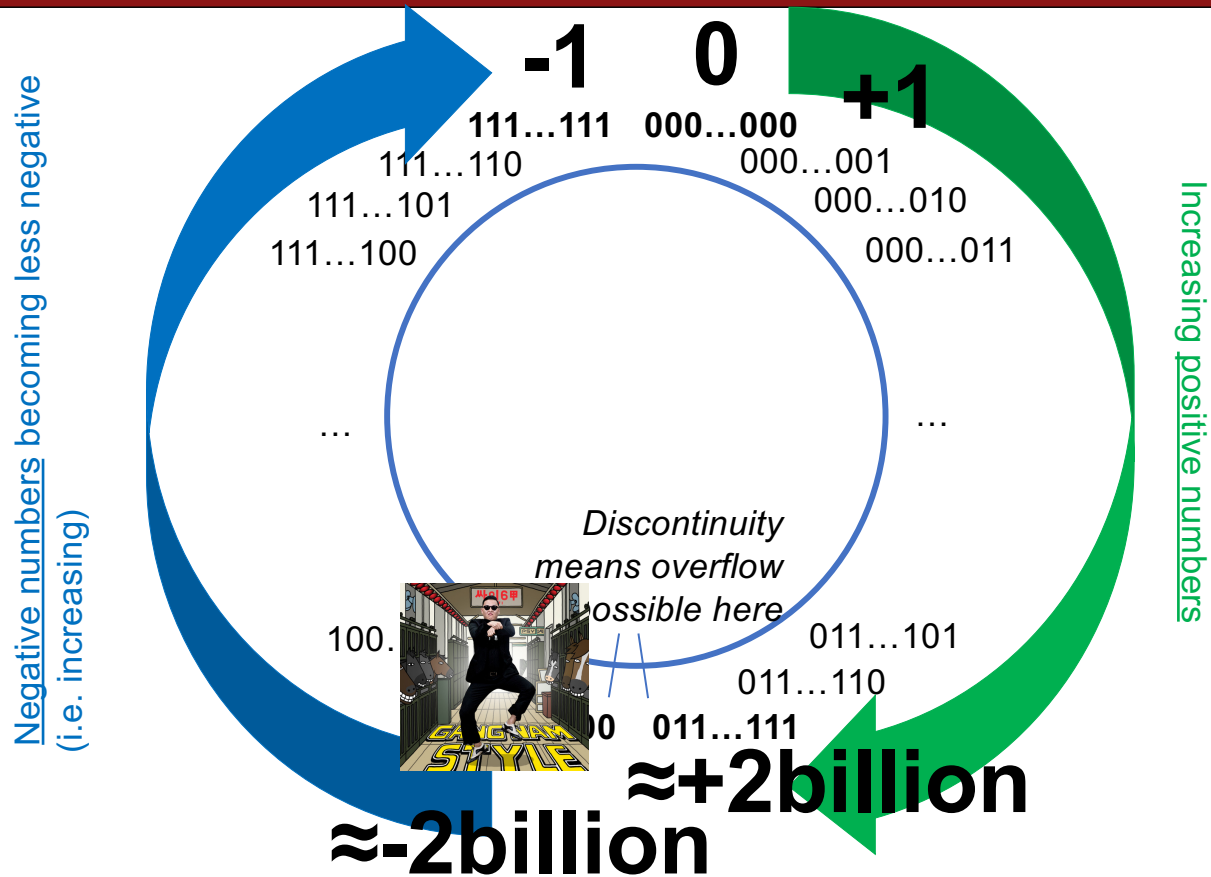
Overflow



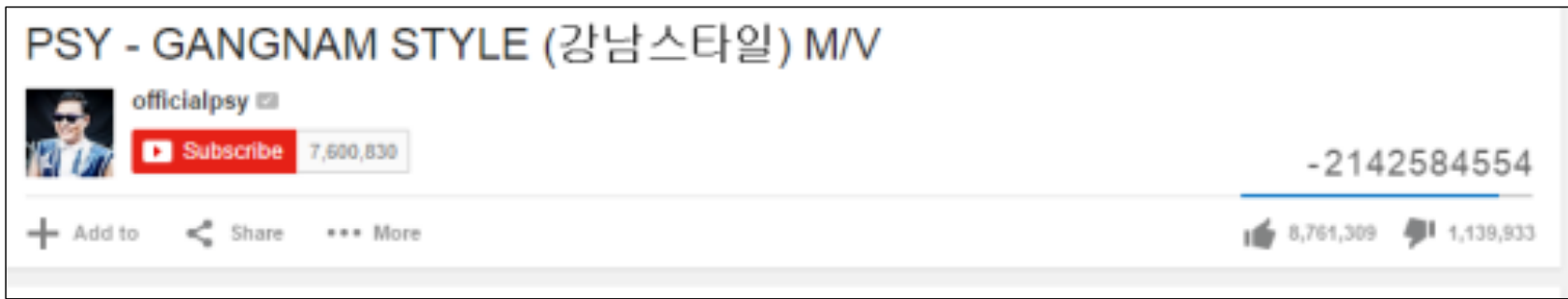
Unsigned Integers



Signed Numbers



Overflow In Practice: PSY



YouTube: "We never thought a video would be watched in numbers greater than a 32-bit integer (up to 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)!" [\[link\]](#)

"We saw this coming a couple months ago and updated our systems to prepare for it" [\[link\]](#)

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:

- [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](#)

Recap

- Bits and Bytes
- Hexadecimal
- Integer Representations
- Unsigned Integers
- Signed Integers
- Overflow

Lecture 3 takeaway: computers represent everything in binary. We must determine how to represent our numbers (e.g., base-10 numbers) in a binary format so a computer can manipulate them. Finite representations come with limitations.

Next Monday: How can we manipulate individual bits and bytes?



Extra Practice

Practice: Two's Complement

Fill in the below table:

It's easier to compute base-10 for positive numbers, so use two's complement first if negative.

	char x = ____;		char y = -x;	
	decimal	binary	decimal	binary
1.		0b1111 1100		
2.		0b0001 1000		
3.		0b0010 0100		
4.		0b1101 1111		



Practice: Two's Complement

Fill in the below table:

It's easier to compute base-10 for positive numbers, so use two's complement first if negative.

	char x = ____;		char y = -x;	
	decimal	binary	decimal	binary
1.	-4	0b1111 1100	4	0b0000 0100
2.		0b0001 1000		
3.		0b0010 0100		
4.		0b1101 1111		



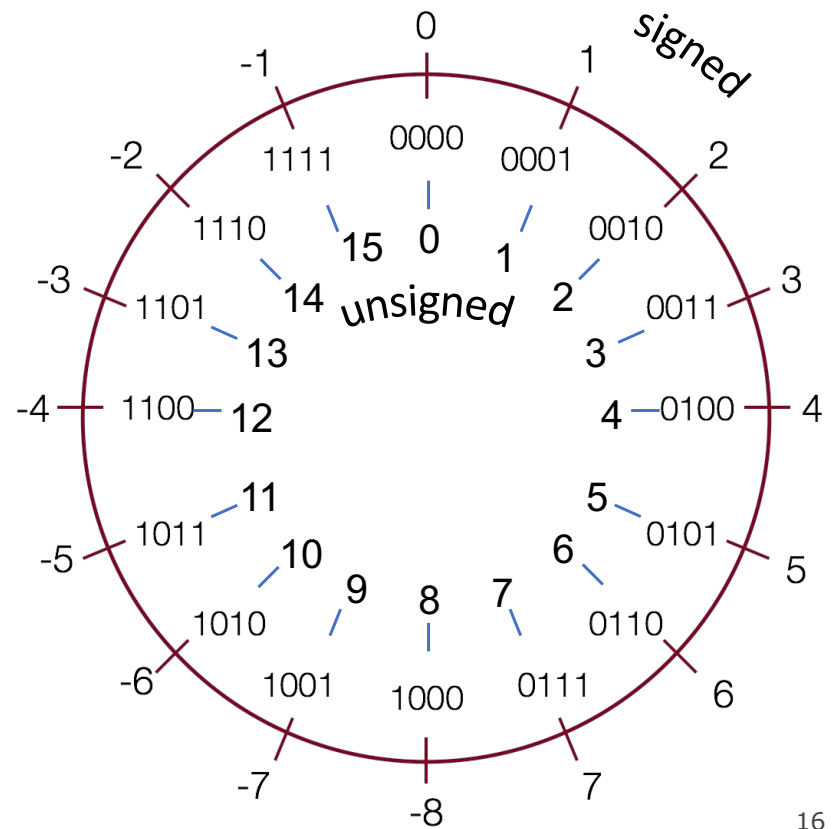
Practice: Two's Complement

Fill in the below table:

It's easier to compute base-10 for positive numbers, so use two's complement first if negative.

	char x = ____;		char y = -x;	
	decimal	binary	decimal	binary
1.	-4	0b1111 1100	4	0b0000 0100
2.	24	0b0001 1000	-24	0b1110 1000
3.	36	0b0010 0100	-36	0b1101 1100
4.	-33	0b1101 1111	33	0b0010 0001

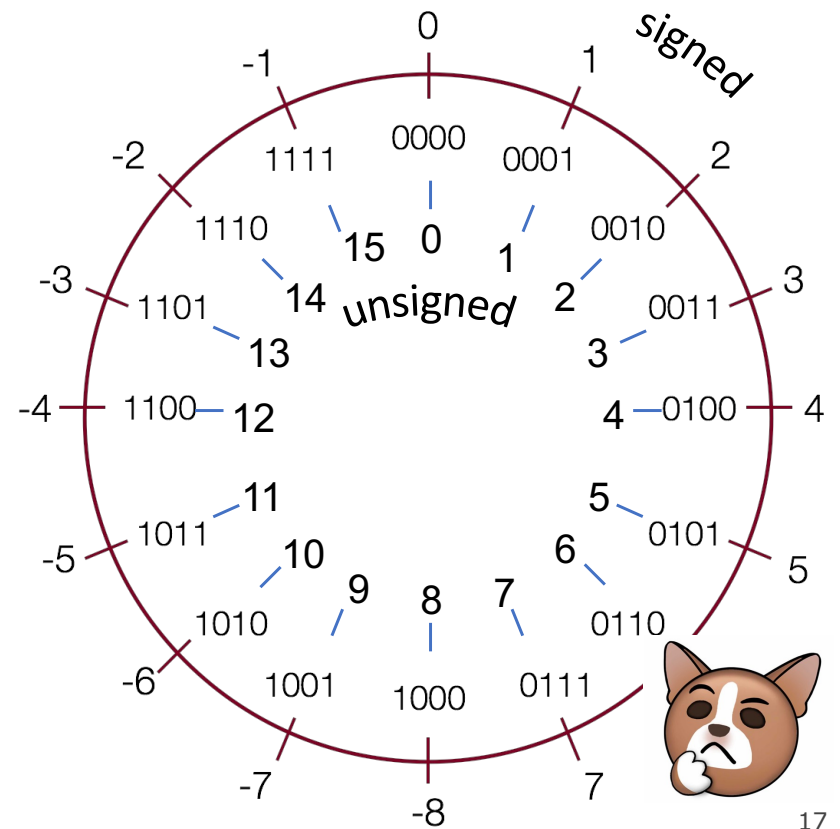
Signed vs. Unsigned Integers



Underspecified question

What is the following base-2 number in base-10?

0b1101



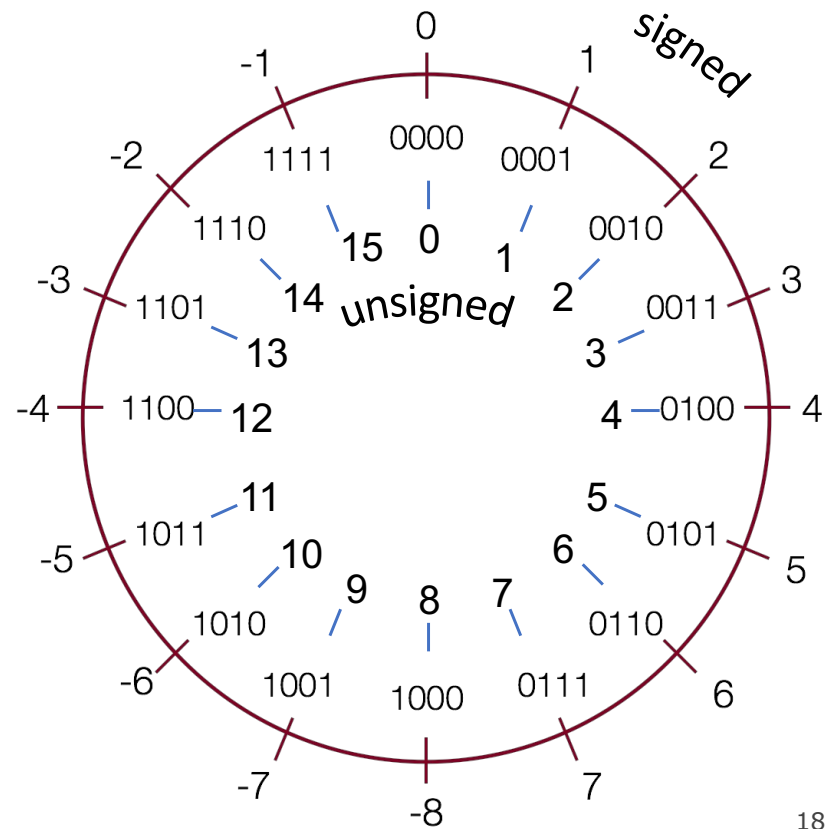
Underspecified question

What is the following base-2 number in base-10?

0b1101

- If 4-bit signed: **-3**
- If 4-bit unsigned: **13**
- If >4-bit signed or unsigned: **13**

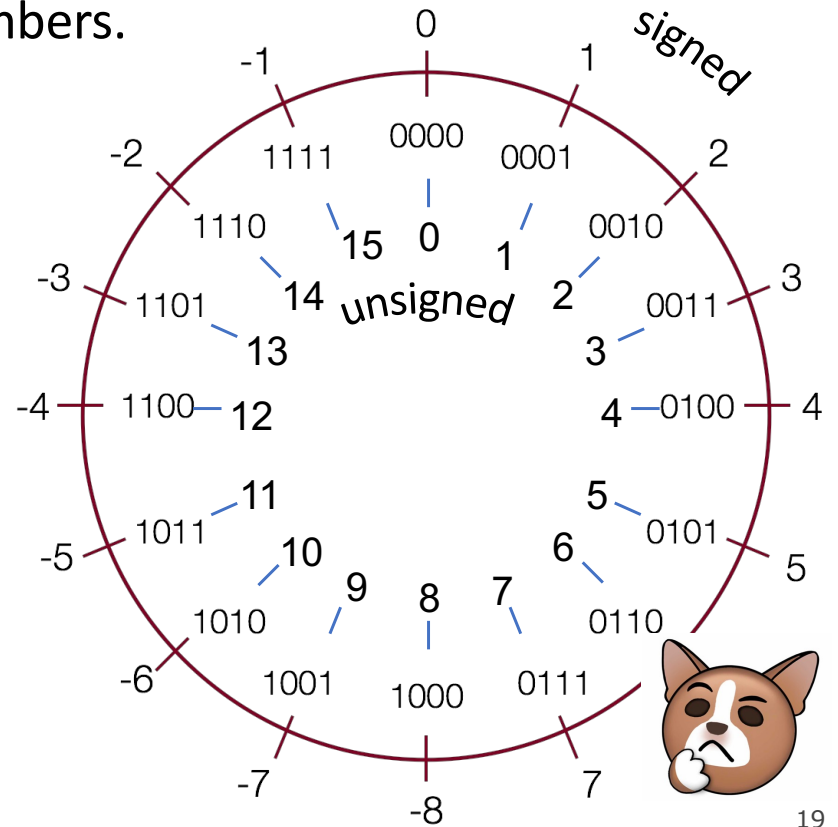
You need to know the type to determine the number! (Note by default, numeric constants in C are signed ints)



Overflow

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

0b1101
+ 0b0100



Overflow

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

$$\begin{array}{r} 0b1101 \\ + 0b0100 \\ \hline \end{array}$$

Signed

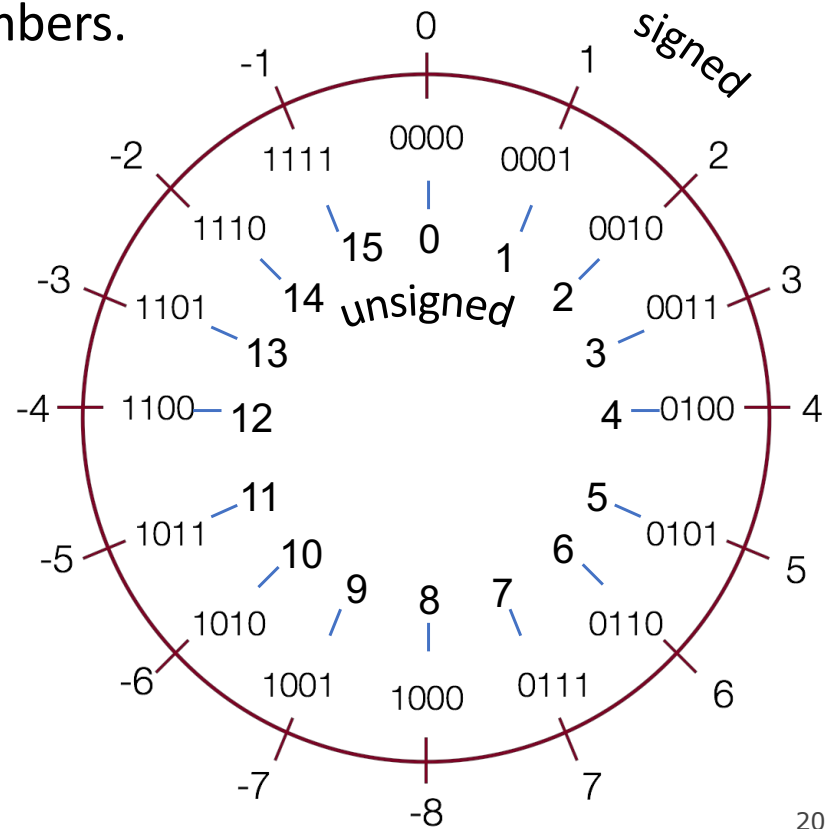
$$-3 + 4 = 1$$

No overflow

Unsigned

$$13 + 4 = 1$$

Overflow



Limits and Comparisons

1. What is the...

	Largest unsigned?	Largest signed?	Smallest signed?
char			
int			



Limits and Comparisons

1. What is the...

	Largest unsigned?	Largest signed?	Smallest signed?
char	$2^8 - 1 = 255$	$2^7 - 1 = 127$	$-2^7 = -128$
int	$2^{32} - 1 = 4294967296$	$2^{31} - 1 = 2147483647$	$-2^{31} = -2147483648$

These are available as UCHAR_MAX, INT_MIN, INT_MAX, etc. in the <limits.h> header.