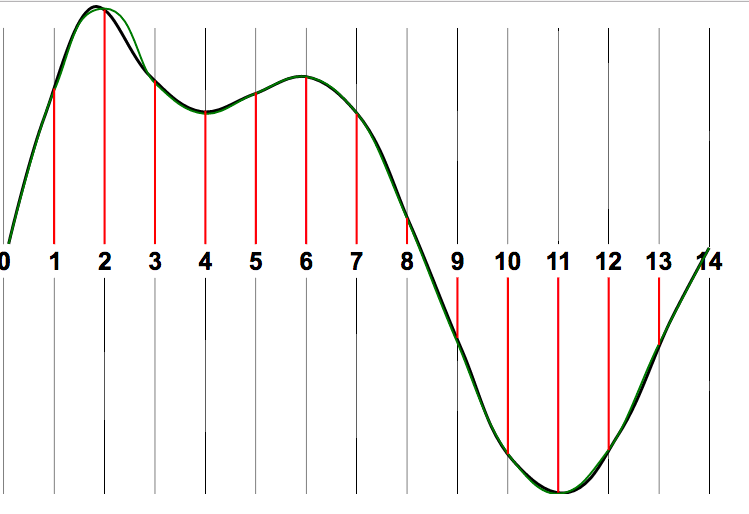
# Why?

In digital computers, data is ultimately stored as bits that take on the value 1 or 0. Later in this course, you'll see how the ability to manipulate data in this format enables us to more easily build complex, reliable computer systems. For now, we're concerned with understanding how bits can represent data.

# Model 1: Bits and binary numbers

Consider a device that samples a sound wave at regular intervals to convert it to a sequence of numbers, each stored digitally as bits. The resolution of the sampling is how many bits each sample gets. The plot below shows a wave. The vertical lines are points in time where the wave is *sampled*. When we sample the wave, we have to record its value as a sequence of bits. The number of tick marks on the y-axis the number of levels. Each level corresponds to a unique sequence of bits.

1. Fill in the following table. The 1st row is already finished and the 2nd row is started. Each bit can take on a value of 0 or 1.

|  |  |  |
| --- | --- | --- |
| **Resolution** | **The list of distinct levels at this resolution** | **Number of levels** |
| 1 bit | 0, 1 |  |
| 2 bits | 00,01,... |  |
| 3 bits |  |  |
| N bits | N/A |  |

1. How did you determine the last row?
2. Write an equation relating resolution (R) to number of levels (L).
3. How many more levels are available when you add 1 bit of resolution?
4. How many more levels are available when you *double* the number of bits of resolution?

# Extension Questions

1. Suppose we converted the sampled wave back to real audio using a speaker. Describe how a low-resolution wave might sound compared to a high-resolution wave.

# Model 2: Number bases and binary

**Base 10:**

places: Hundreds | tens | ones

19010 = 1 9 0

**Base 2:**

places: fours | twos | ones

1012 = 1 0 1

1. What numbers can be used in a single base 10 digit?
2. What numbers can be used in a single base 2 digit?
3. What would be the next place to the left of *hundreds*? To the left of *fours*?

# Read This!

An ordered sequence of ***bits*** can be used to represent integers. To represent an integer in ***base two (aka, binary)*** means to write its “digits" as bits indicating the powers of two that add up to the number.

1. List the numbers zero to eight in order, representing them in base two.
2. Translate the following base two numbers to our typical number base: base ten (aka, decimal).

11112 = \_\_\_\_10

101012 = \_\_\_10

1. Write a formula for translating a binary number xn-1 ... x3 x2 x1 to decimal. The xi means the ith digit of the binary number (either a 1 or 0). Your formula may contain decimal constants in it.
2. Translate the following base ten numbers to base two.

810 = \_\_\_\_\_\_\_\_\_\_2

2510 = \_\_\_\_\_\_\_\_\_\_2

10010 = \_\_\_\_\_\_\_\_\_\_2

1. Write an algorithm (as pseudocode or detailed steps is acceptable) to compute each bit in the binary representation of a decimal number.
2. Use what you've learned about base 10 and base 2 numbers to translate numbers between other bases.

2510 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3 Used the above algorithm but divide by 3 instead of 2

2116 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

# Read This!

Other than binary and decimal, another common base that we will use is 16 or *hexademical* (or *hex* for short). In hexadecimal, we represent digits larger than 9 as follows: A=10, B=11, C=12, D=13, E=14, F=15.

1. Convert the following numbers to the given base.

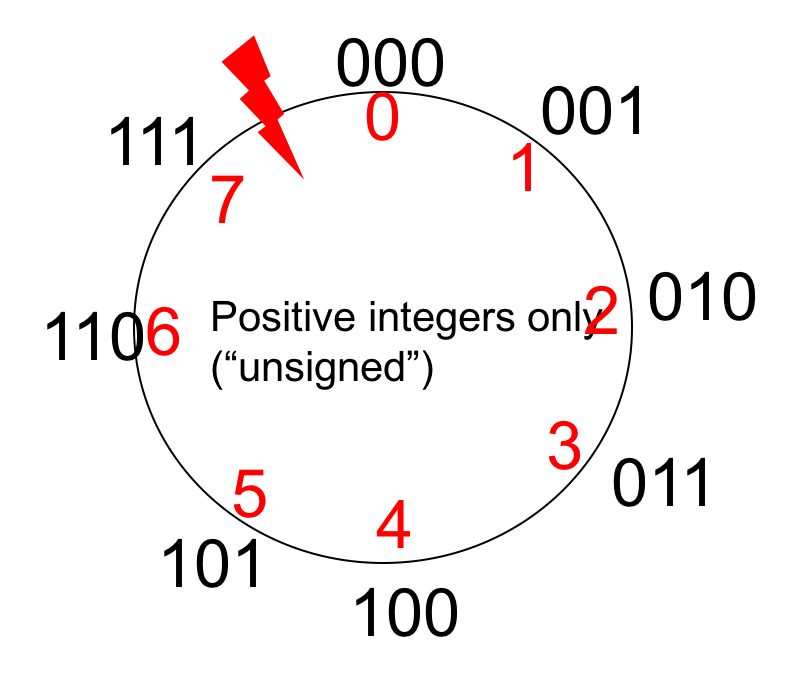
A116 = \_\_\_\_\_\_10

5610 = \_\_\_\_\_\_16

26710 = \_\_\_\_\_\_16

# Model 3: Including negative integers

On the left is a number wheel for positive integers that can be represented using 3 bits.



1. When we move clockwise on the number wheel what happens to the decimal numbers?

The numbers increase.

1. What happens at the lightning bolt?

It resets back to 0.

# Read This!

At the lightning bolt we ***overflow***, that is, jump between the smallest and largest integers.

1. On the following number line, write out 3-bit binary integers, using the following rules:

* the leftmost bit indicates sign (0 positive, 1 negative)
* the remaining bits indicates the absolute value

binary 111, 110, 101, 000, 001, 010, 011

decimal -3, -2, -1, 0, 1, 2, 3

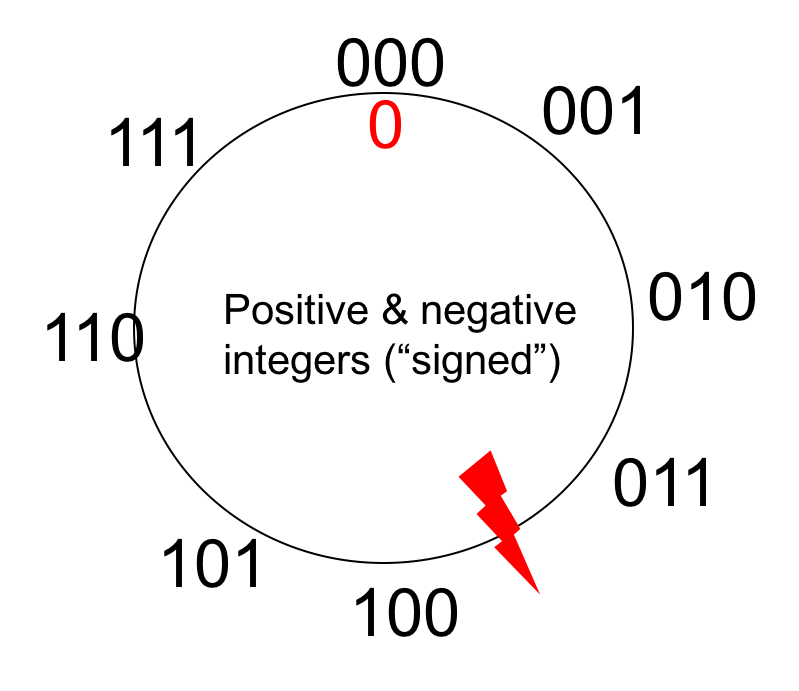
1. Did you use all available 3-bit numbers? Why or why not?

No, because 100 is -0, which doesn’t exist.

1. Turn your number line into a number wheel by attaching the endpoints. What do you observe that is interesting?

# Model 4: Including negative integers with Two’s Complement

Let's try a different approach to negative integers by starting with the 3-bit number wheel.





1. Complete the wheel above to include both positive and negative integers. The overflow point and 0 have been provided as hints. Use the following constraints:

* as before, clockwise must be +1 and counter-clockwise -1
* for all positive numbers and zero, the most significant (left-most) bit must be 0
* for all negative numbers, the most significant (left-most) bit must be 1
* unlike before, no integer may appear twice (i.e., no integer may have two different 3-bit representations)

# Read This!

The most common way to represent signed integers using bits is ***two's complement***. A correct number wheel above demonstrates 3-bit two's complement. In n-bit two's complement, the binary digits xn-1 ... x2 x1 x0 can be interpreted using the expression

1. Convert the following values from 4-bit two's complement to decimal.

1011 =-5

0111 =7

1. For an n-bit two's complement integer, what is the smallest value that can be represented? The largest? Why?

Smallest: 0000, largest: 1111

***Two's complement algorithm:***

To convert a negative integer, , to n-bit two's complement:

step 1: Convert to its n-bit binary representation

step 2: Complement each bit of ( ), that is, change 1's to 0's and 0's to 1's

step 3: Add one to

1. Convert the following signed decimal numbers into two's complement.

|  |  |  |
| --- | --- | --- |
| signed decimal | in 8-bit two’s complement | in 12-bit two’s complement |
| 59 | 00111011 | 0000 00111011 |
| -59 | 11000100 🡪 11000101 | 1111 11000100 🡪  1111 11000101 |
| 3 | 00000011 | 0000 00000011 |
| -3 | 11111101 | 1111 11111101 |
| 8 | 01000 | -8 = 10111 🡪 11000 |
| -8 + 3 = -5 | 11000 + 00011 = 11011 |  |

, which is the 2's complement algorithm

1. Compare the two’s complement numbers in 8-bit vs 12-bit. What pattern do you see?

If you want to extend the size in terms of bts, you just extend the number in the front to the number wanted. (00000 or 11111)

# Read This!

Converting a two’s complement number from a smaller bitwidth to a larger bitwidth can be done using an operation called ***sign extension***.

1. Using the pattern you recognized in #26, why is “sign extension” an appropriate name for this operation?

# Extension questions

-x in n-bits two’s complement

1. Solve the above equation for How does this new equation relate to the two's complement algorithm?

Derived from CS-POGIL ORG\_02\_TWOS

# Learning objectives

* Translate integers and decimals between different number bases
* Represent negative integers with two's complement
* Calculate the inverse of an integer in two's complement
* Identify the largest and smallest integers representable using N bits
* Sign-extend a two’s complement integer