Unsigned Integers are just what you’d expect: traditional translation from binary to decimal

- add with carry and borrow, just like in decimal
- for n bits, range from 0 to $2^n-1$

Signed Integers: Sign and Magnitude

- treat the most significant bit (MSB) as a sign bit (1 indicates negatives) and remaining bits as magnitude
- arithmetic doesn’t work out right
- two representations of zero
- for n bits, range from $-2^{n-1}-1$ to $2^{n-1}-1$

Signed Integers: Two’s Complement

- treat the most significant bit as having $(-1)*2^{n-1}$ weight, for a word length of n
- makes -1 equal to 0xFFFF (all ones)
- $-x = \sim x + 1$
- math works out right (assuming we drop the carry), just wraps around once we’re out of range: A+B in two’s complement is equal to (A+B) % range modular arithmetic
  - overflow (when the number gets too big and wraps around to negatives)
  - underflow (when the number gets too small and wraps around to high positives)
- for n bits, range from $-2^{n-1}$ to $2^{n-1}-1$

Casting between signed and unsigned: doesn’t change the bits, just the way we interpret those bits

- Explicit casting by saying (int) ux or (unsigned) of -3
- Implicit casting by assigning: converts to the type of the destination
- Implicit casting by comparing: converts to unsigned if any of the members is unsigned

- Casting to a larger type (e.g. from short to int) does sign extension to fill in the extra bits

- Shifting moves binary representation right or left n bits (equivalent to multiply or divide by $2^n$)
  - arithmetic extends the sign, logical extends with zeros

- Multiplication and Division with signed and unsigned
  - equivalent to computing the correct “math” result (no matter how many bits) and truncating down to n bits, or computing “math” result % range