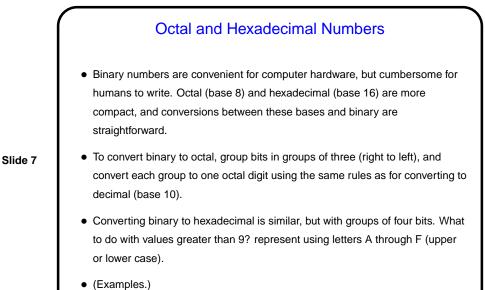


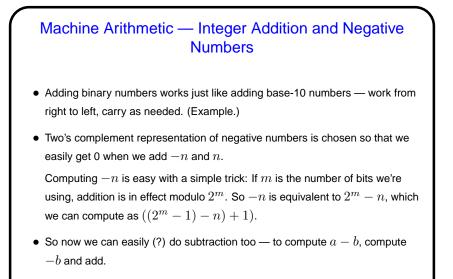
Decimal to Binary, Take 2 Another way produces the answer from right to left rather than left to right, repeatedly dividing by 2 (again n will be the number we want to convert): If n = 0, stop. Divide n by 2, giving quotient q and remainder r. Write down r. Set n equal to q. Go back to first step. Is this okay? What's not quite right about it? (We don't say to write down the remainders from right to left.) (Example.)



(Examples.)

Computer Representation of Integers

- Computers represent everything in terms of ones and zeros. For non-negative integers, you can probably guess how this works — number in binary. Fixed size (so we can only represent a limited range).
- Slide 8
- How about negative numbers, though? No way to directly represent plus/minus. Various schemes are possible. The one most used now is "two's complement": Motivated by the idea that it would be nice if the way we add numbers didn't depend on their sign. So first let's talk about addition ...



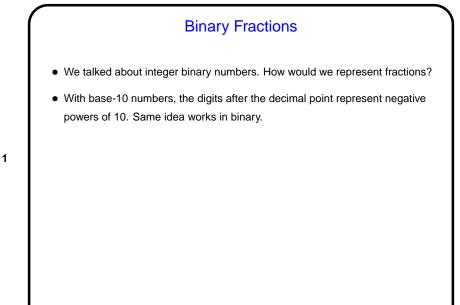
Machine Arithmetic — Integer Multiplication

Multiplying binary numbers also works just like multiplying base-10 numbers
 — for each digit of the second operand, compute a partial result, and add them.

• (This can get slightly tricky, when adding more than two partial results involves carrying, but basic idea is straightforward extrapolation from how it

Slide 10

works in base 10.)



Computer Representation of Real Numbers

- How are non-integer numbers represented? usually as floating point.
- Idea is similar to scientific notation represent number as a binary fraction multiplied by a power of 2:

Slide 12

$$x = (-1)^{sign} \times (1 + frac) \times 2^{bias + exp}$$

and then store $sign \ frac$, and exp. Sign is one bit; number of bits for the other two fields varies — e.g., for usual single-precision, 8 bits for exponent and 23 for fraction. Bias is chosen to allow roughly equal numbers of positive and negative exponents.



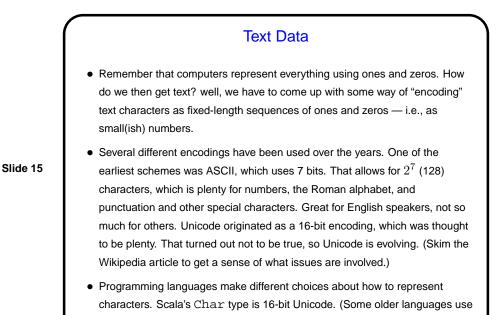
- The integers and real numbers of the idealized world of math have some properties not (completely) shared by their computer representations.
- Math integers can be any size; computer integers can't.
- Math real numbers can be any size and precision; floating-point numbers can't. Also, some quantities that can be represented easily in decimal can't be represented exactly in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)

Scala and Representing Numbers — Review/Recap

• Computer hardware typically represents integers as a fixed number of binary digits, using "two's complement" idea to allow for representing negative numbers. Scala, like many (but not all!) programming languages bases its notion of integer data on this, but also has a notion of different types with different sizes (e.g., Int versus Long).

Slide 14

 Hardware also typically supports "floating-point" numbers, with a representation based on a base-2 version of scientific notation. This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Notice that only fractions that can be written with a denominator that's a power of two can be represented exactly. Again Scala goes along with this and provides two different "sizes" (Float and Double).



ASCII instead.) Single-character literals use single quotes.

For the probability of the probability of

Objects and Methods

• Text strings don't really correspond to anything the hardware can work with as directly as it works with integer and floating-point numbers. So how to represent them is left somewhat more to the discretion of the programming language. They're a simple example of a kind of thing we might want to be able to work with that's somewhat more complicated than what the hardware provides.

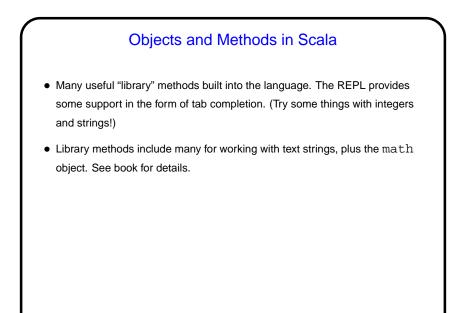
Slide 17

- To make working with things other than simple numbers easy, Scala, again like many (but not all!) programming languages has a notion of *objects* (i.e., it is an *object-oriented* (OO) language).
- Remember that we defined a type as a set of values together with some operations on them? In OO-speak, an *object* is something with a value of a particular type, and its *methods* are operations that the type says can be done on it (e.g., arithmetic operations on integers).

Objects and Methods in Scala

• In Scala (unlike some other popular programming languages), everything is an object. This makes some things very convenient (though it puts a certain distance between the language and the hardware, which *may* have negative effects on performance).

- Some operations on objects just do something, without any need for more information (e.g., toInt converts a Double to an Int). Others require *parameters* (e.g., integer addition).
- Basic syntax for invoking an object's methods requires a period, the name of the method, parentheses, and any parameters. Scala allows many of these to be omitted if it can figure out what you mean. (Indeed, some methods that take no parameters must *not* be followed by parentheses.)



Slide 20

Variables We know enough — more than enough — at this point to use the Scala REPL as a calculator. But that's not really programming, since if we want to do the same calculation for different sets of values we'd have to retype everything. To do almost anything interesting, we need some way to save values and give them names, so we can reference them again. So Scala, like most programming languages, has a notion of *variables*, similar (but not identical!) to variables in math. (The biggest difference is that some Scala variables can take on different values as a calculation proceeds.) (To be continued.)

