Slide 1

Administrivia

- Reminder: Homework 1 due today. Send me e-mail with the file you wrote as an attachment. Simplest way to do this is from a browser on one of the lab/classroom machines. If you're working remotely, you can copy the file to your machine as described in my notes on remote access.
- Quiz 1 will be a week from today. More about it next time.
- If you're perplexed by the textbook's occasional use of the character ¿ — it's meant to be a $>$. (Tools glitch!)

Slide 2

Minute Essay From Last Lecture

- Everything apparently reasonably clear so far (!).
- One person asked about `vi` error messages produced when a session "crashes". My e-mail of earlier this week is meant to explain.

Slide 3

Programming-Language Terminology

- *token*: set of characters that means something in the language, often separated by whitespace.
- *literal*: token representing a value (e.g., 1).
- *statement*: set of tokens that give a complete action.
- *expression*: set of tokens that together give a value (e.g., 1 + 1).
- *type*: set of values together with operations on them (e.g., integer, (text) string). Every value has a type.

Slide 4

Numeric Literals and Expressions

- Numeric literals and expressions should be fairly familiar. Notice that Scala (like many programming languages) makes a distinction between integers and numbers that have (or might have) a fractional part.
- Use the interpreter's REPL to try out things. Some things may be surprising — integer division, large numbers, calculations using fractions. To understand some of these it helps to know how the computer represents numbers.

Slide 5

Binary Numbers

- We humans usually use the decimal (base 10) number system, but other (positive integer) bases work too. (Well, maybe not base 1.) Binary (base 2) is more widely used in computers because it makes the hardware simpler.
- In base 10, there are ten possible digits, with values 0 through 9.

  In base 2, there are 2 possible digits (*bits*), with values 0 and 1.
- In base 10, 1010 means what? What about in base 2?

Slide 6

Converting Between Bases

- Converting from another base to base 10 is easy if tedious (just use definition).
- Converting from base 10 to another base? Let's try to develop an algorithm (procedure) for that . . .

Slide 7

Decimal to Binary, Take 1

- One way is to first find the highest power of 2 smaller than or equal to the number, write that down, subtract it from the number, and continue:
  1. If $n = 0$, stop.
  2. Find largest p such that $2^p \leq n$.
  3. Write a 1 in the p-th output position.
  4. Subtract 2^p from n.
  5. Go back to first step.
- Is this okay? What's not quite right about it? (We don't say what to put in the positions that don't have ones in them.)
- (Example.)

Slide 8

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):
  1. If $n = 0$, stop.
  2. Divide n by 2, giving quotient q and remainder r.
  3. Write down r.
  4. Set n equal to q.
  5. 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.)

Slide 9

Octal and Hexadecimal Numbers

- Binary numbers are convenient for computer hardware, but cumbersome for humans to write. Octal (base 8) and hexadecimal (base 16) are more compact, and conversions between these bases and binary are straightforward.
- To convert binary to octal, group bits in groups of three (right to left), and convert each group to one octal digit using the same rules as for converting to decimal (base 10).
- Converting binary to hexadecimal is similar, but with groups of four bits. What to do with values greater than 9? represent using letters A through F (upper or lower case).
- (Examples.)

Slide 10

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).
- 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 doesn’t depend on their sign. So first let’s talk about addition . . .

Slide 11

Machine Arithmetic — Integer Addition and Negative Numbers

- Adding binary numbers works just like adding base-10 numbers — work from right to left, carry as needed. (Example.)
- Two's complement representation of negative numbers is chosen so that we easily get 0 when we add $-n$ and n.

  Computing $-n$ is easy with a simple trick: If m is the number of bits we're using, addition is in effect modulo 2^m. So $-n$ is equivalent to $2^m - n$, which we can compute as $((2^m - 1) - n) + 1)$.
- So now we can easily (?) do subtraction too — to compute $a - b$, compute $-b$ and add.

Slide 12

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 tricky, when adding more than two partial results involves carrying.)

Binary Fractions

- We talked about integer binary numbers. How would we represent fractions?
- With base-10 numbers, the digits after the decimal point represent negative powers of 10. Same idea works in binary.

Slide 13

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:

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

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.

Slide 14

Slide 15

Numbers in Math Versus Numbers in Programming

- 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!)

Slide 16

Minute Essay

- What is 110_2 in base 10?
- What's the (base 10) value of the largest number you can represent with 4 bits? (E.g., the largest number you can represent with 2 bits is 11_2, or 3_{10}.)

Slide 17

Minute Essay Answer

- 110_2 is 6_{10}.
- 15 (1111_2).