Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Floating-Point Numbers

Schedule

- **Today**
  - **Homework #2 assigned**
  - Floating-Point Numbers
- **Friday**
  - Floating-Point Numbers
- **Monday**
  - Character representation
  - **Homework #2 due**
  - **Quiz #1**
    - Material from Homework #1 and #2
    - Material from Intro lecture

Recap

- **Take a few minutes with your neighbor to convert BAD_{16} to binary**
  - B= 11_{10} = 1011
  - A= 10_{10} = 1010
  - D= 13_{10} = 1101
  - **Soln = `1011 1010 1101`**

Recap

- **With your neighbor, write 23 in the following forms:**
  - **(1) Unsigned (2) Sign-magnitude**
    (3) One's complement (4) Two's complement
  - **What's the one thing I need to tell you first?**
    - **Let's say: 12 bit long representation**
- Convert |23| to binary: 10111 (i.e. 16+4+2+1)
- **Answers are all the same! 0000 0001 0111**
  - **Because number is positive**

Recap

- **With your neighbor, write -23 in the following forms using a 12-bit long representation:**
  - **(1) Unsigned (2) Sign-magnitude**
    (3) One's complement (4) Two's complement
- **Unsigned – No representation possible**
- **Sign-Magnitude: 1000 0001 0111**
- **One's complement: 1111 1110 1000**
  - ***(extend 23 to 12 bits, and then invert)***
- **Two's complement: 1111 1110 1001**
  - ***(one's complement plus 1)***

Range

- **What is the smallest and largest 8-bit two's complement number?**
  - $\mathtt{XXXXXXXX}_2$
  - Smallest (negative) # = $\mathtt{10000000}_2$ = **-128**
  - Largest (positive) # = $\mathtt{01111111}_2$ = **127**

Reminders

For positive numbers, the *signed-magnitude*, *one's complement*, and *two's complement* forms are all **the same**!

In *one's complement* / *two's complement* form, you only need to modify the number if it is **negative**!

Homework #1

- Solutions will be posted in Sakai (resources folder)
  - *Why Sakai? Only available to class members…*

Floating-Point Numbers

Why Floating-Point?

- Existing representations deal with integer values only
  - Signed magnitude
  - One's complement
  - Two's complement
- Adding in a fixed decimal point is awkward / inflexible
- Scientific and business applications need a standardized way to deal with real number values
  - **Floating-point numbers**

Floating-Point Representation

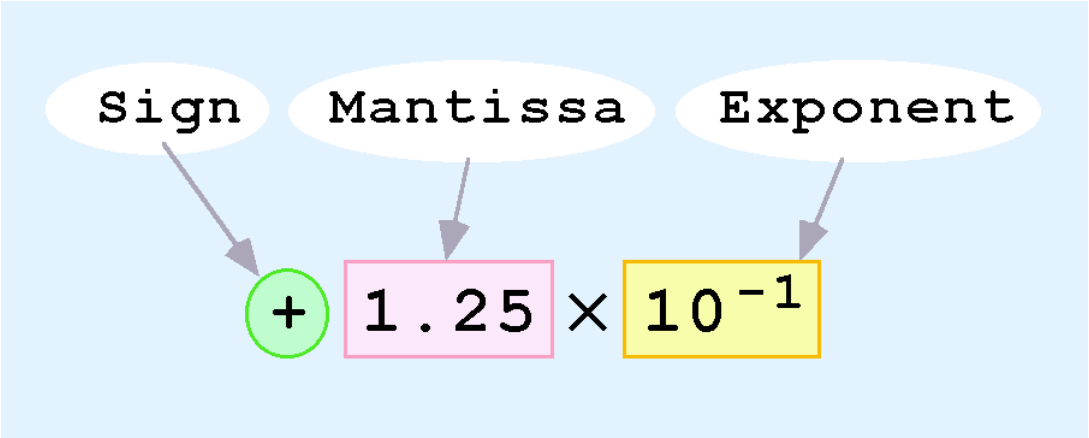
- Do we need hardware or software?
  - Clever programmers can do floating-point purely in software
  - Drawbacks: Complicated, slow
- Modern computers have **specialized hardware** that directly performs floating-point arithmetic

Floating-Point Representation

- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
  - For example: $0.5 \times 0.25 = 0.125$
- They are often expressed in **scientific notation**
  - For example:
    - $0.125 = 1.25 \times 10^{-1}$
    - $5{,}000{,}000 = 5.0 \times 10^{6}$

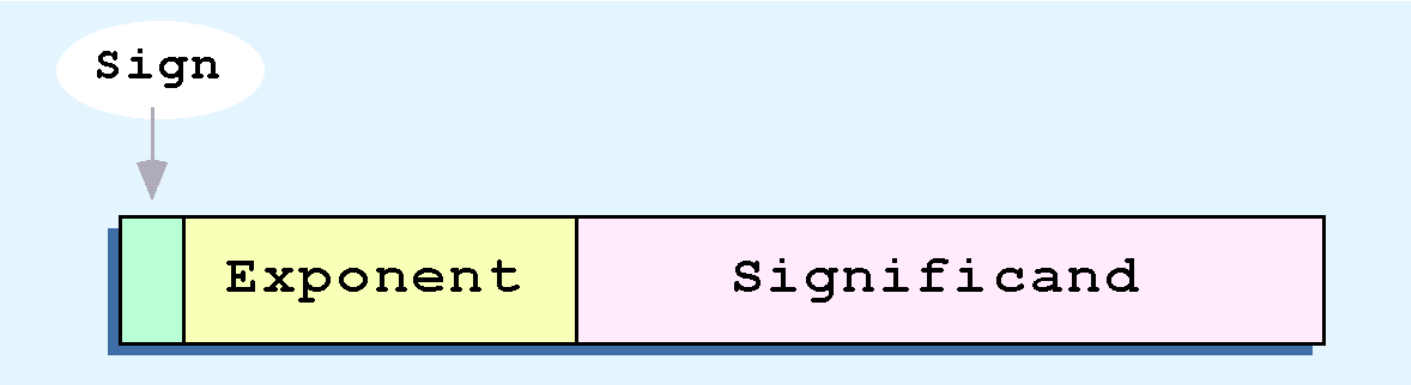
Floating-Point Representation

- Computers use a form of scientific notation for floating-point representation
- Numbers written in scientific notation have three components:



Floating-Point Representation

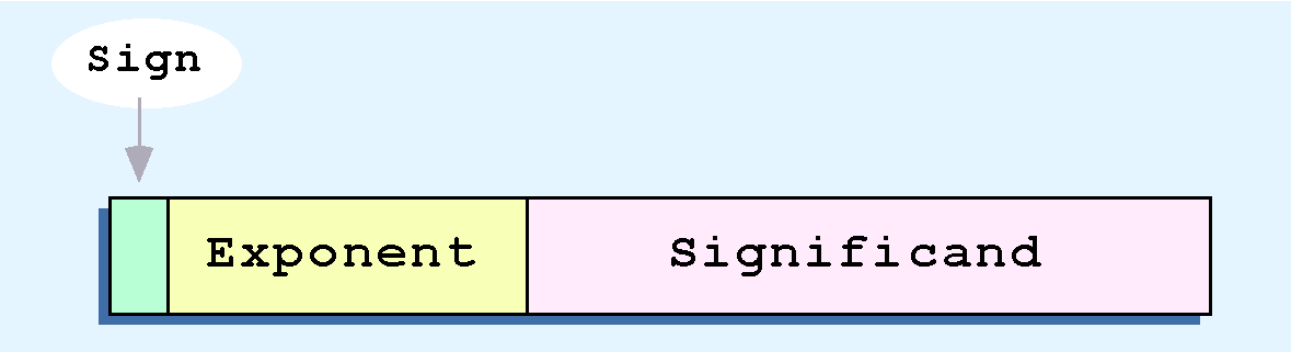
- Computer representation of a floating-point number consists of three fixed-size fields:
- This is the standard arrangement of these fields:



Note: Although "significand" and "mantissa" do not technically mean the same thing, many people use these terms interchangeably. We use the term "significand" to refer to the fractional part of a floating point number.

Floating-Point Representation

- The one bit *sign* field is the sign of the stored value.
- The size of the *exponent* field determines the **range** of values that can be represented
- The size of the *significand* determines the **precision** of the representation



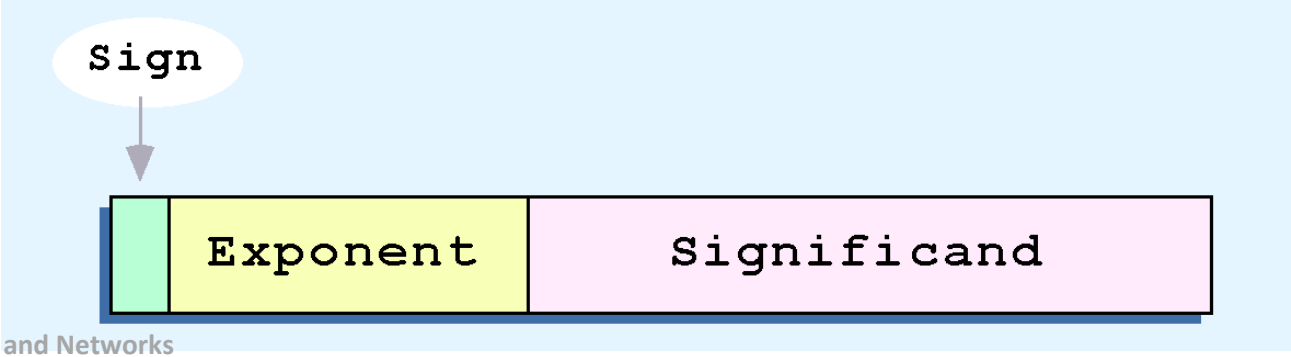
Floating-Point Errors

- When discussing floating-point numbers, it is important to understand the terms *range*, *precision*, and *accuracy*
- The **range** of a numeric integer format is the difference between the largest and smallest values that can be expressed
- **Accuracy** refers to how closely a numeric representation approximates a true value
- The **precision** of a number indicates how much information we have about a value

Simplified Floating-Point Model

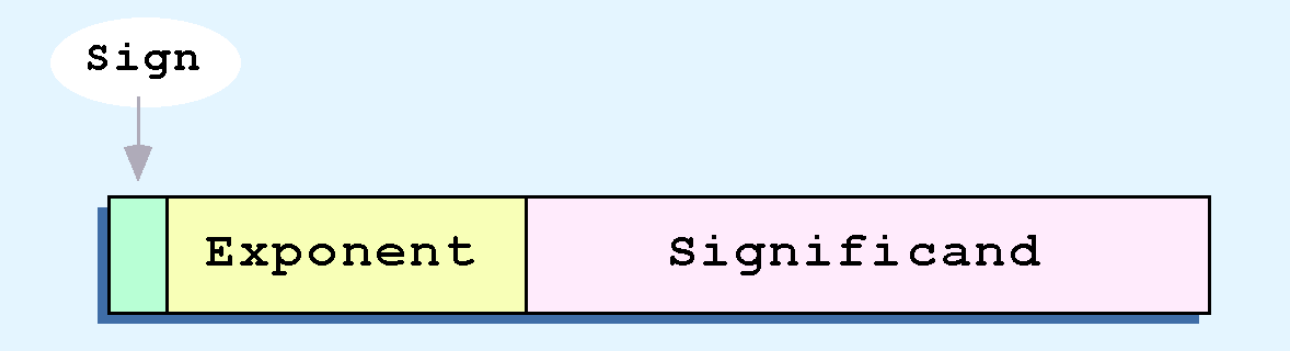
Floating-Point Representation

- We introduce a hypothetical "Simple Model" to explain the concepts with smaller numbers
  - *Later we'll discuss the real standard!*
- 14 bit long floating-point number:
  - The sign field is 1 bit
  - The exponent field is 5 bits
  - The significand field is 8 bits



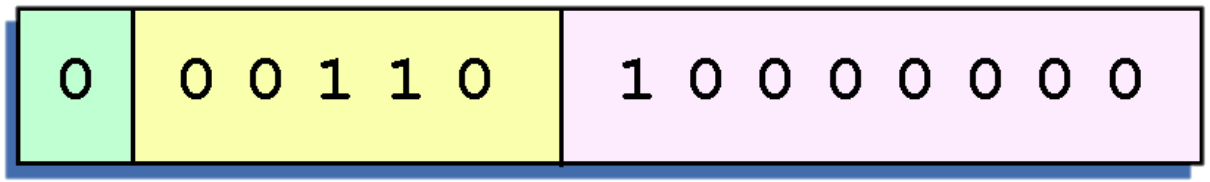
Floating-Point Representation

- The significand is always preceded by an implied binary point, i.e. `0.xxxxxxxxxx`
  - Thus it always contains a **fractional** binary value
- The exponent indicates the power of 2 by which the significand is multiplied



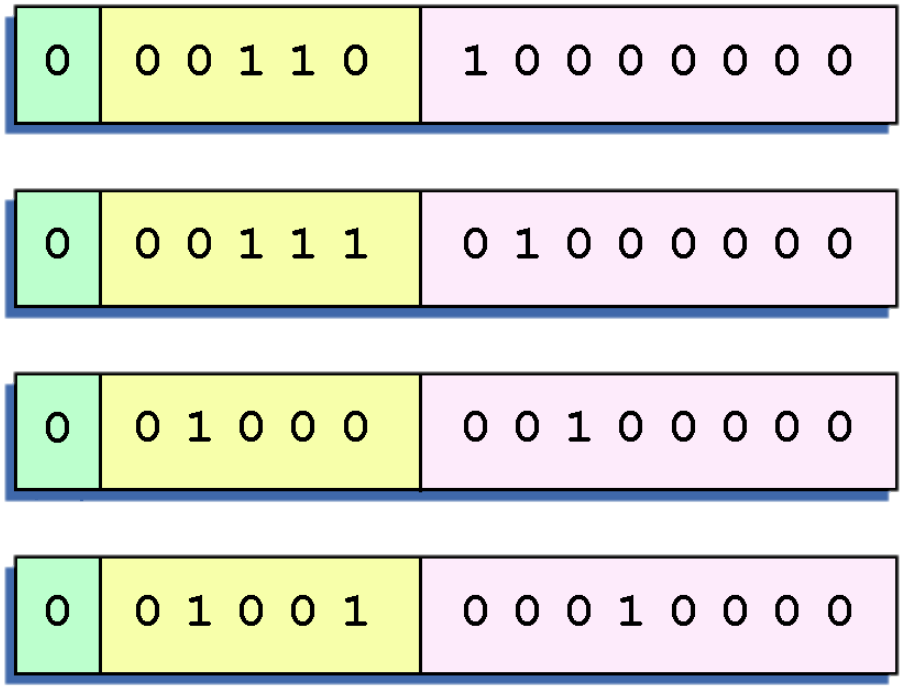
Floating-Point Representation

- Example: Express 32_{10} in the simplified 14-bit floating-point model
- We know that 32 is 2^5. So in (binary) scientific notation $32 = 1.0 \times 2^5 = 0.1 \times 2^6$
  - In a moment, we'll explain why we prefer the second notation versus the first
- Using this information, we put 110 (= 6_{10}) in the exponent field and 1 (padded with 0's) in the significand:



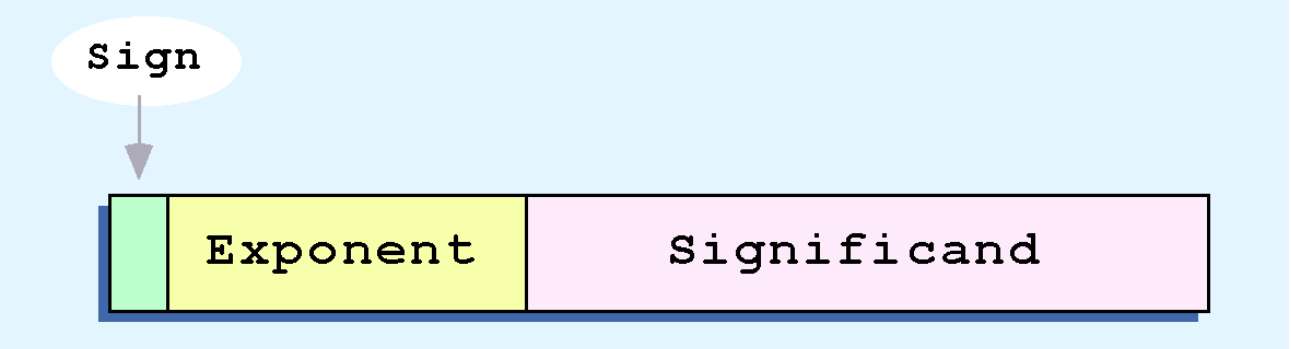
Floating-Point Representation

- **Problem 1:** We have many ways to represent the exact same number
  - Waste space
  - Cause confusion
  - Difficult to test for equality
- Figure: Multiple ways to represent 32 using our simplified model
  - By varying the exponent and shifting



Floating-Point Representation

- **Problem 2:** No way to express negative exponents
  - Can't store 0.5 ($=2^{-1}$)!
  - The exponent field is an **unsigned** value



All of these problems can be fixed with no changes to our basic model

Floating-Point Representation

- **New Rule #1**
  - The **first digit of the significand must be 1**
  - **In our simple model, all significands must have the form 0.1xxxxxxxx**
  - No ones to the left of the radix point
- This process is called **normalization**
  - Produces a unique pattern for each floating-point number
- Example: 4.5_{10}
  $= 100.1 \times 2^0$
  $= 1.001 \times 2^2$
  $= 0.1001 \times 2^3$.
  The last expression is correctly normalized.

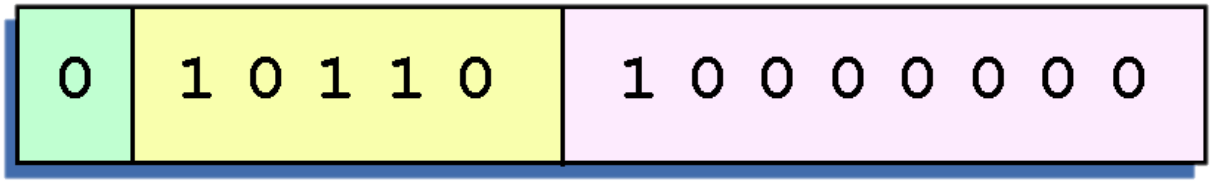
In our simple instructional model, we use no implied bits

Floating-Point Representation

- **New Rule #2:** To allow for negative exponents, we will use a *biased exponent*
- What is a *bias*?
  - A number that is approximately midway in the range of values expressible by the exponent
  - To determine the exponent to store, add the bias to your exponent
  - To decode a floating-point number, subtract the bias from the value in the exponent field
- Simplified model with 5-bit exponent
  - Use a bias of 16 (called an *excess-16* representation)
  - Exponent values less than 16 are negative, representing fractional numbers

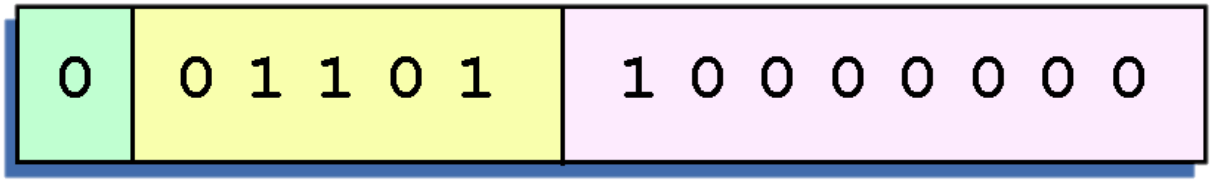
Floating-Point Representation

- Example: Express $\mathbf{32_{10}}$ in the revised 14-bit floating-point model
- We know that $32 = 1.0 \times 2^5 = 0.1 \times 2^6$
  - Significant is now normalized (0.1xxxxxx)
- Use excess 16 biased exponent
  - Add 16 to 6 = 22_{10} (=10110_2)
- Final value saved to memory:



Floating-Point Representation

- Example: Express **0.0625_{10}** in the revised 14-bit floating-point model
- 0.0625 is 2^{-4}. In (binary) scientific notation, $0.0625 = 1.0 \times 2^{-4} = 0.1 \times 2^{-3}$ *(normalized notation)*
- Use excess 16 biased exponent
  - $16 + -3 = 13_{10}$ $(=01101_2)$
- Final value saved to memory:



Floating-Point Representation

- Example: Express **-26.625_{10}** in the revised 14-bit floating-point model
- $26.625_{10} = 11010.101_2 \times 2^0$
  Normalize = 0.11010101×2^5.
- Use excess 16 biased exponent:
  - $16 + 5 = 21_{10}$ ($=10101_2$)
- Also need a 1 in the sign bit (negative number)
- Final value saved to memory: