Numeric Data Representation & Computation Accuracy

WE ARE ILLINOIS TECH

ILLINOIS INSTITUTE OF TECHNOLOGY

Number Bases

- Base 10 – 1354 is $1*10^3+3*10^2+5*10^1+4*10^0$ or “one thousands, three hundreds, five tens and four ones”
- Base 2 – 10110 is $1*2^4+0*2^3+1*2^2+1*2^1+0*2^0$ or “one sixteens, zero eights, one fours, one twos, and zero ones” (22 in base 10)

- How do we convert from base 10 to base 2?
  - Repeated integer division by two until nothing is left, keep track of the remainder (0 or 1), write down the remainders right to left

- How do we convert from base 2 to base 10?
  - See above

POLL Convert 47 to binary

a)111101

b)101111

c)101110

d)111011

Why do S/W Engineers care?

Why do S/W Engineers care?

- Accuracy of the abstraction
  - Will integer calculations in my programs be accurate?
  - If I get unexpected results when testing, what could the cause be?

Some new questions

- Does every base 10 integer have a unique, exact equivalent in base 2? And vice-versa?
- How are negative integers represented?

3 Methods for Representing Integers

Goals

- Tell when a number is negative or positive
- Add two numbers together regardless of sign
- Compare a pair of numbers to quickly tell which is larger (and by how much)

Depending upon the application one or the other of these goals becomes more important.

1. Signed Binary
2. Two's Complement
3. Bias

Two's Complement

Decimal value	Binary (two's-complement representation)
0	000
1	001
2	010
3	011
−4	100
−3	101
−2	110
−1	111

What is the result when you add 1 to 3? Or add -1 to -4?

POLL What is -23 base 10 in 8-bit two's-complement binary?

a) 11101000

b) 11101001

c) 10010110

d) 10010111

3 Methods for Representing Integers

1. Signed Binary - easy to tell when a number is positive or negative, but isn't particularly useful for addition or for a simply described comparison algorithm

2. Two's Complement - easy to tell when a number is positive or negative, and easy comparison (via subtraction and checking the result). But also has the advantage that simple binary addition (and subtraction, and multiplication) still works.

3. Bias - If a quick comparison between two numbers is more important than addition, storing integers using bias is more convenient.

Binary Fractions

- How do we represent base 10 non-integers? Is there a parallel in binary?
- Normalized form of binary fractions is given as 1.xxxxxxxx * 2^power
  This format ensures that we retain as many significant bits as possible
- Why do S/W Engineers care?

Why do S/W Engineers care?

- Accuracy of the abstraction
  - rounding
  - finite

Floating Point Representation

(IEEE-754 floating point standard)

$$(-1)^S * 1.F * 2^E$$

- S represents the number's Sign: 0 for positive and 1 for negative numbers.
- F represents the float's Fractional part.
- E represents the float's Exponent

Single-precision floats use 32 bits (float data type in Java)
-|--------|----------------------
1 bit is reserved for S
8 bits are reserved for E which is stored with a +127 bias
remaining 23 bits are used to store F

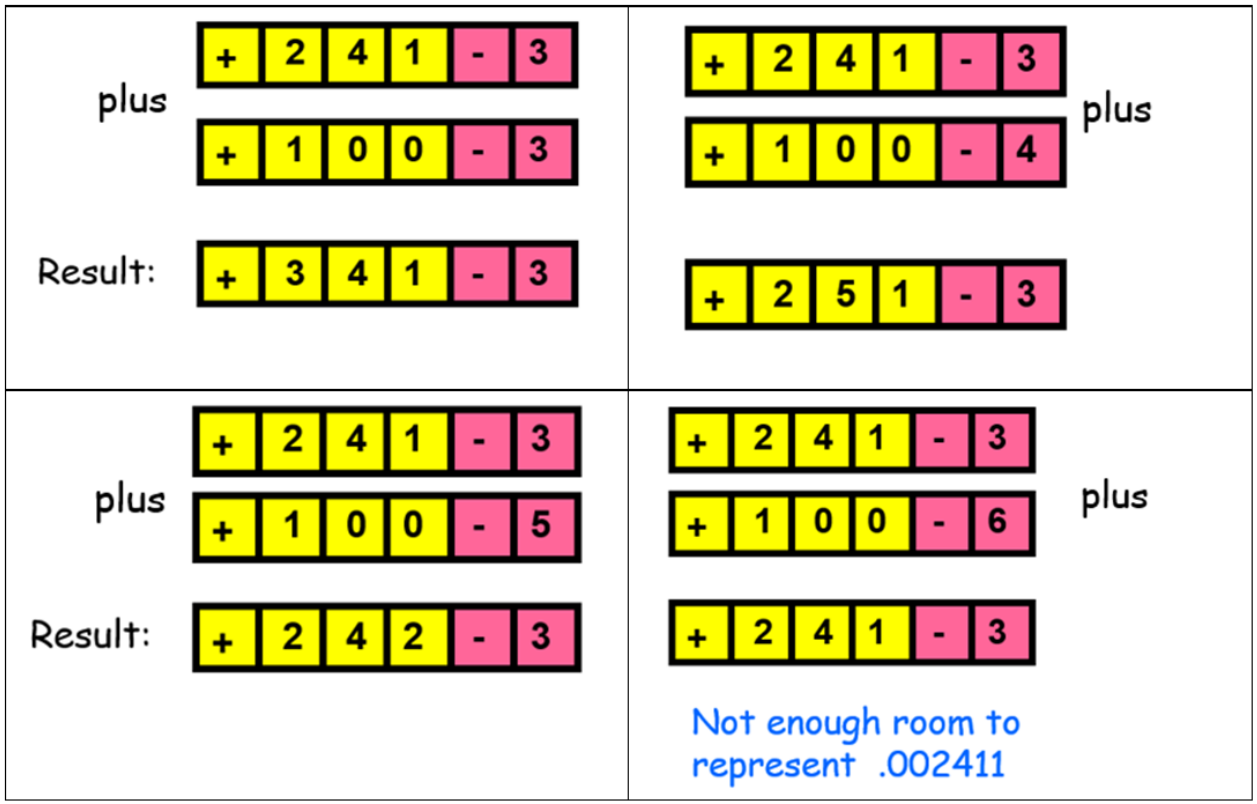
Double-precision floats use 64 bits (double data type in Java)
1 bit is reserved for S
11 bits are reserved for E which is stored with a +1023 bias
remaining 52 bits are used to store F

Why do S/W Engineers care?

Why do S/W Engineers care?

- Finite Limitation (not in binary example)

Representing 2.41×10^{-3}



Why do S/W Engineers care?

- Arithmetic not associative in extremes

We are told from a rather early age that addition is associative. This means that given 3 (or more) numbers a, b and c we can write a+b+c and that it doesn't matter how we determine the sum because (a+b)+c equals a+(b+c). However, imagine a situation where b and c are relatively small in comparison to a.

What might happen?

Java Data Type Ranges

Type	Size in Bytes	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
char	2 byte	0 to 65,536 (unsigned)