NAAC ACCREDITED "B++" (CGPA 2.89)

Yogoda Satsanga Mahavidyalaya

Jagannathpur, Dhurwa, Ranchi-834004

www.ysmranchi.net

ysmranchi4@gmail.com

Course :Computer System Architecture

Class : Sem-1

Lesson :Number System Contd..

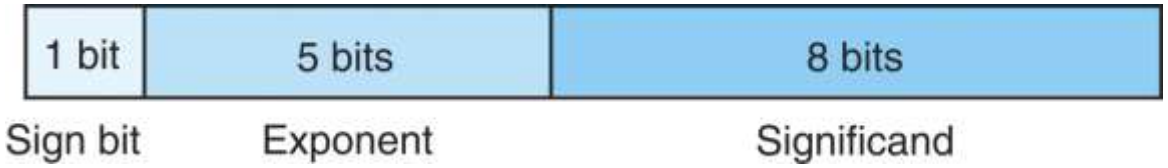
By :Goutam Sanyal

Floating Point Representation

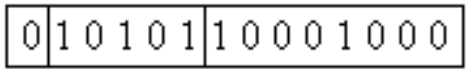
- Floating point numbers have a *floating* decimal point
  - Recall the fraction notation used a fixed decimal point
  - Floating point is based on scientific notation
    - $3518.76 = .351876 * 10^4$
    - We represent the floating point number using 2 integer values called the significand and the exponent, along with a sign bit
      - The integers are 351876 and 4 for our example above
  - For a binary version of floating point, we use base 2 instead of 10 as the radix for our exponent
  - We store the 2 integer values plus the sign bit all in binary
    - We *normalize* the floating point number so that the decimal is implied to be before the first 1 bit, and in shifting the decimal point, we determine the exponent
    - The exponent is stored in a bias representation to permit both positive and negative exponents
    - The significand is stored in unsigned magnitude

Examples

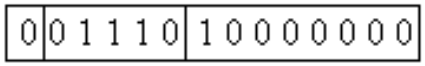
- Here, we use the following 14-bit representation:



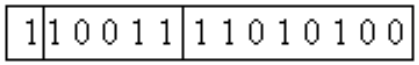
Exponents will be stored using excess-16



Sign bit = 0 (positive)
Exponent = 5 (10101 – 10000 = 5)
Significand = .10001000
We shift the decimal point 5 positions giving us 10001.0 = +17



Sign bit = 0 (positive)
Exponent = -2 (01110 – 10000 = -2)
Significand = .10000000
We shift the decimal point 2 positions to the left,
giving us 0.001 = +.125



Sign bit = 1 (negative)
Exponent = 3 (10011 – 10000 = 3)
Significand = .11010100
We shift the decimal point 3 positions to the right,
giving 110.101= -6.625

Floating Point Formats and Problems

- To provide a standard for all architectures, IEEE provides the following formats:
  - Single precision
    - 32-bits: 1-bit sign, 8-bit exponent using excess-127, 23-bit significand
  - Double precision
    - 64-bits: 1-bit sign, 11-bit exponent using excess-1023, 52-bit significand
  - IEEE also provides NAN for errors when a value is not a real number
    - NAN = *not a number*
- Problems
  - there are numerous ways to represent the same number (see page 58), but because we normalize the numbers, there will ultimately be a single representation for the number
  - Errors arise from
    - overflow (too great a positive number or too great a negative number) – overflowing the signficand
    - underflow (too small a fraction) – overflowing the exponent