Week 4 Lecture

B&O Chapter 2 (continued)

## Bitwise Shift Operations

* Shifting bit patterns to the left ( operator)
  + Let have a bit representation and let be an integer.
  + The expression , shifts bits to the left, dropping off the most significant bits and filling the right end with zeros.
  + So the result of has a bit representation
  + Note: Shift operations associate left to right , so is equivalent to .
* Shifting bit patterns to the right ( operator)
  + Corresponds to left shift, except machines generally support two forms of the right shift operation:
    1. **Logical**
       - The logical right shift fills the left end with k zeros, giving a result .
    2. **Arithmetic**
       - The arithmetic right shift fills the left end with repetitions of the most significant bit of , which gives the result .
* Example: Applying the different shift operations to two different values of an 8-bit argument .

|  |  |  |  |
| --- | --- | --- | --- |
| Value of (binary) |  | (logical) | (arithmetic) |
| 01100011 | 0011*0000* | *0000*0110 | *0000*0110 |
| 10010101 | 0101*0000* | *0000*1001 | *1111*1001 |

* Practice: (problem 2.16, B&O pg. 58)

## Floating-Point Representation

* What is a floating-point number?
  + It is the encoding of rational numbers
  + Useful for performing computations involving very large numbers
  + Standard is IEEE floating-point format

### Fractional binary numbers

* Binary numbers having fractional values
* Analogous to decimal notation (just in base 2 rather than base 10)
* The symbol ’.’ now becomes a binary point instead of a decimal point
  + bits on the left are weighted by nonnegative powers of 2
  + bits on the right weighted by negative powers of two
* Example: 101.112
  + Answer: Represents the number
* Shifting the binary point one position to the left has the effect of dividing the number by 2, and similarly, shifting the binary point one position to the right has the effect of multiplying the number by 2.

### IEEE Floating-Point Format

* This format is defined in the **IEEE Standard for Floating-Point Arithmetic (IEEE 754)** (
  + The first version of this standard was published by IEEE in 1985 (IEEE 754-1985).
  + A revised and updated version was later published in 2008 (IEEE 754-2008).
* This standard represents a number in the form .
  + represents the sign ( for negative and for positive)
    - Note: Interpretation of the sign bit for numeric value 0 is handled by a special case.
  + represents the significand (also called the mantissa)
    - This is a fractional binary number that ranges either between and or between and
  + represents the exponent, which weights the value by a possibly negative power of 2.
* The bit representation of this format is divided into three fields to encode these three values:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Sign  (1 bit) | Exponent  ( bits) | Significand (i.e. Mantissa)  ( bits) |

* + 1-bit field for sign, directly encodes the sign
    - So again, for negative and for positive
  + -bit exponent field to encode the exponent
  + -bit fractional field to encode the significand (or mantissa)
* Exponent bias
  + Reason for using a bias
    - Exponents have to be signed values in order to be able to represent both tiny and huge values.
    - **Problem**: Simply using a signed value (two's complement is the usual representation for signed values) would make comparison harder.
    - **Solution**: Store the exponent as an unsigned value suitable for comparison, and when being interpreted it is converted to an exponent in a signed range by subtracting the bias.
  + For an arbitrarily sized floating point number, the bias , where is the number of bits in the exponent
* Formats: Several are defined in the standard but we will only discuss the 3 basic binary formats.
  + Basic Binary Formats:
    - Single Precision (binary32, occupies 32 bits in memory)
      * **Sign bit**: 1 bit
      * **Exponent width**: 8 bits
        + **Exponent bias**:

since exponent width is 8 bits

The exponent is stored in the range 1 .. 254 (0 and 255 have special meanings), and is biased by subtracting 127 to get an exponent value in the range −126 .. +127

* + - * **Significand field width**: 23 bits
      * **Bits precision**: 24 bits
    - Double Precision (binary64, occupies 64 bits in memory)
      * **Sign bit**: 1 bit
      * **Exponent width**: 11 bits
        + **Exponent bias**:

since exponent width is 8 bits

The exponent is stored in the range 1 .. 2046 (0 and 2047 have special meanings), and is biased by subtracting 1023 to get an exponent value in the range −1022 .. +1023

* + - * **Significand field width**: 52 bits
      * **Bits precision**: 53 bits
    - Quadruple Precision (binary128, occupies 128 bits in memory)
      * Same idea