CS1101: Lecture 31
IEEE Floating-Point Standard 754

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie

Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101

Department of Computer Science, University College Cork

Lecture Outline

- IEEE Floating-Point Standard 754
- The Significand
- An Example Conversion
- Another Example Conversion
- Error Handling
- IEEE numerical types
- Denormalized Numbers
- Zero
- Overflow
- **Reading**: Tanenbaum, Appendix B.

IEEE Floating-Point Standard 754

- The standard defines three formats:
  - single precision (32 bits),
  - double precision (64 bits), and
  - extended precision (80 bits).
- Both the single- and double precision formats use **radix 2 for fractions** and **excess notation for exponents**.
- Both formats start with a sign bit, 0 being positive and 1 being negative.
- The exponent is defined using excess 127 for single precision and excess 1023 for double precision.
- The minimum (0) and maximum (255 and 2047) exponents are not used for normalized numbers – they have special uses.
- The fractions have 23 and 52 bits, respectively.

IEEE floating-point formats

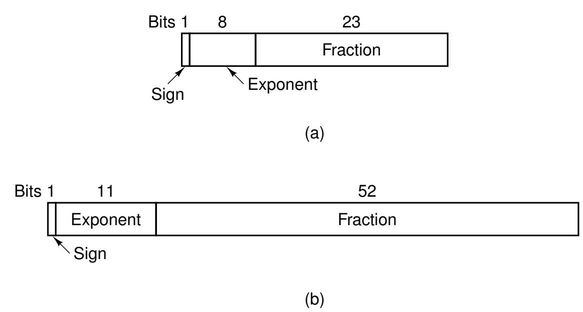

Figure B-4. IEEE floating-point formats. (a) Single precision. (b) Double precision.

The Significand

- A normalized fraction begins with a 1 bit, followed by a binary point, and then the rest of the fraction.
- The leading 1 bit in the fraction does not have to be stored, since it can just be assumed to be present.
- Consequently, the standard defines the fraction in a slightly different way than usual.
- It consists of an implied 1 bit, an implied binary point, and then either 23 or 52 arbitrary bits.
- To avoid confusion with a conventional fraction, the combination of the implied 1, the implied binary point, and the 23 or 52 explicit bits is called a **significand** instead of a fraction or mantissa.
- All normalized numbers have a significand, s, in the range $1 \leq s < 2$.

An Example Conversion

- **Example:** Show the IEEE 754 binary representatoin of the number 0.5_{10} in single precision.
- This is equivalent to 1.0×2^{-1} in normalised binary scientific notation
- Thus, the fraction is $00000\ldots000$ (i.e. we ignore the "1." in the significand)
- The sign is positive, which is 0
- The exponent is

$$-1 + 127 = 126_{10} = 01111110_2$$

- We can now put it all together

An Example Conversion

- Thus the IEEE floating-point formatted number for 0.5_{10} is

$$00111111000000000000000000000000$$

which, formatted differently, is

$$0011\ \ 1111\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

- We can also express this as

$$3F000000_{16}$$

- Also, 0.5, 1.0 and 1.5 are represented in hexadecimal as 3F000000, 3F800000, 3FC00000, respectively.

Another Example Conversion

- **Example:** Convert the IEEE single-precision floating-point number $3FC00000_{16}$ from hex to decimal.
- In binary this is:

$$0011\ \ 1111\ \ 1100\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

- The sign is 0 - it's a positive number
- The exponent is

$$01111111 = 127_{10} = 127_{10} - 127_{10} = 0$$

Another Example Conversion

- The fraction is

$$10000000000000000000000$$

giving a significand of 0.1.

- Thus, the number is

$$(1 + fraction) \times 2^{exponent}$$

giving

$$(1 + 0.1) \times 2^0 = 1.1 \times 2^0 = 1.5$$

Error Handling

- One of the traditional problems with floating-point numbers is how to deal with underflow, overflow, and uninitialized numbers.
- In addition to normalized numbers, the standard has four other numerical types:
  - Normalized
  - Denormalized
  - Zero
  - Infinity
  - Not a number

IEEE numerical types

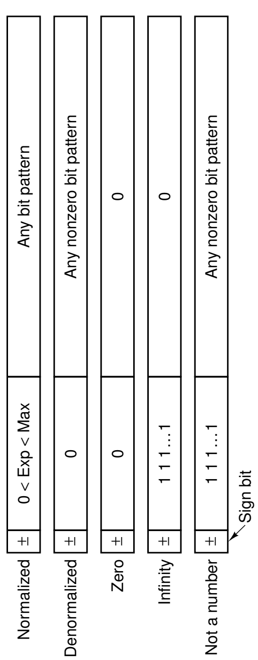

Figure B-6. IEEE numerical types.

Denormalized Numbers

- A problem arises when the result of a calculation has a magnitude smaller than the smallest normalized floating-point number that can be represented in this system.
- To handle this sort of situation the IEEE invented **denormalized numbers**.
- These numbers have an exponent of 0.
- Normalized numbers are not permitted to have an exponent of 0.

Denormalized Numbers

- The smallest nonzero denormalized number consists of a 1 in the rightmost bit, with the rest being 0.
- The exponent represents 2^{-127} and the fraction represents 2^{-23} so the value is 2^{-150}.
- This scheme provides for a graceful underflow by giving up significance instead of jumping to 0 when the result cannot be expressed as a normalized number.

Zero

- Two zeros are present in this scheme, positive and negative, determined by the sign bit.
- Both zeros have an exponent of 0 and a fraction of 0.
- Here too, the bit to the left of the binary point is implicitly 0 rather than 1.

Overflow

- Overflow cannot be handled gracefully.
- There are no bit combinations left.
- Infinity is represented by an exponent with all 1s (not allowed for normalized numbers), and a fraction of 0.
- This number can be used as an operand and behaves according to the usual mathematical rules for infinity.
- For example infinity plus anything is infinity, and any finite number divided by infinity is zero.

Overflow

- Similarly, any finite number divided by zero yields infinity.
- Infinity divided by infinity is undefined.
- To handle this case, another special format is provided, called **NaN (Not a Number)**.
- NaN can be used as an operand with predictable results.