COMP3221: Microprocessors and Embedded Systems

Lecture 14: Floating Point Numbers
http://www.cse.unsw.edu.au/~cs3221
Lecturer: Hui Wu
Session 1, 2005

Overview

- IEEE Floating Point Number Representation
- Floating Point Number Operations

Scientific Notation

- Normalized form: no leadings 0 (exactly one non-zero digit to the left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: $1.0 \times 10^{-9}$
  - Not normalized: $0.1 \times 10^{-8}, 10.0 \times 10^{-10}$

How to represent 0 in Normalized form?

Scientific Notation for Binary Numbers

- Computer arithmetic that supports it is called floating point, because it represents numbers where binary point is not fixed, as it is for integers
  - Declare such variables in C as float (single precision floating point number) or double (double precision floating point number).
Floating Point Representation

Normal form: \( +(-) \ 1.x \times 2^y \)

- How many bits for significand (mantissa) \( x \)?
- How many bits for exponent \( y \)?
- Is \( y \) stored in its original value or in transformed value?
- How to represent \( +\text{infinity} \) and \( -\text{infinity} \)?
- How to represent 0?

Overflow and Underflow

- What if result is too large?
  - Overflow!
  - Overflow \( \Rightarrow \) Positive exponent larger than the value that can be represented in exponent field
- What if result too small?
  - Underflow!
  - Underflow \( \Rightarrow \) Negative exponent smaller than the value that can be represented in Exponent field
- How to reduce the chance of overflow or underflow?

IEEE 754 FP Standard—Single Precision

Sign bit 
Biased Exponent 
Significand

\( S \ EEEEEEE FFFFFFFFFFFFFFF \\
31 \ 30 \ 23 \ 22 \ 1 \ 0 \)

- Bit 31 for sign
  - \( S=1 \) for negative numbers, 0 for positive numbers
- Bits 23-30 for biased exponent
  - The real exponent = \( E - 127 \)
  - 127 is called bias.
- Bits 0-22 for significand

IEEE 754 FP Standard—Single Precision (Cont.)

The value \( V \) of a single precision FP number is determined as follows:

- If \( 0 < E < 255 \) then \( V = (-1)^S \times 2^{E-127} \times 1.F \) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If \( E = 255 \) and \( F \) is nonzero, then \( V = \text{NaN} \) ("Not a number")
- If \( E = 255 \) and \( F \) is zero and \( S \) is 1, then \( V = -\text{Infinity} \)
- If \( E = 255 \) and \( F \) is zero and \( S \) is 0, then \( V = \text{Infinity} \)
- If \( E = 0 \) and \( F \) is nonzero, then \( V = (-1)^S \times 2^{-126} \times 0.F \). These are unnormalized numbers or subnormal numbers.
- If \( E = 0 \) and \( F \) is 0 and \( S \) is 1, then \( V = 0 \)
- If \( E = 0 \) and \( F \) is 0 and \( S \) is 0, then \( V = 0 \)
IEEE 754 FP Standard—Single Precision (Cont.)

Subnormal numbers reduce the chance of underflow.

• Without subnormal numbers, the smallest positive number is $2^{-127}$
• With subnormal numbers, the smallest positive number is $0.0000000000000000001 \times 2^{-126} = 2^{-(126+23)} = 2^{-149}$

IEEE 754 FP Standard—Double Precision

Sign bit Biased Exponent Significand
S EEEEEEEEEEE FFFFFFFFFF...FFFFFFFFFFFF

Bits 63 62 52 51 1 0

• Bit 63 for sign
  - S = 1 for negative numbers, 0 for positive numbers
• Bits 52-62 for biased exponent
  - The real exponent = $E - 1023$
  - 1023 is called bias.
• Bits 0-51 for significand

Hardware Support for FP Numbers

• Typically a coprocessor implements FP.
  - Works under the processor’s supervision
  - Has its own set of registers and instructions
  - The hardware for FP is quite complicated.
• Most low end microprocessors microcontrollers such as AVR do not support FP numbers in hardware.
  - Need to use software to implement FP if necessary.
Implementing FP Addition by Software

How to implement \( x + y \) where \( x \) and \( y \) are two single precision FP numbers?

Step 1: Convert \( x \) and \( y \) into IEEE format

Step 2: Align two significands if two exponents are different.

- Let \( e_1 \) and \( e_2 \) are the exponents of \( x \) and \( y \), respectively, and assume \( e_1 > e_2 \). Shift the significant (including the implicit 1) of \( y \) right \( e_1 - e_2 \) bits to compensate for the change in exponent.

Step 3: Add two (adjusted) significands.

Step 4: Normalize the result.

An Example

How to implement \( x + y \) where \( x = 2.625 \) and \( y = -4.75 \)?

Step 1: Convert \( x \) and \( y \) into IEEE format

\[
\begin{align*}
x &= 2.625 &\rightarrow 10.101 \text{ (Binary)} \\
&\rightarrow 1.0101 \times 2^1 \text{ (Normal form)} \\
&\rightarrow 1.0101 \times 2^{128} \text{ (IEEE format)} \\
&\rightarrow 0 10000000 01010000000000000000000000
\end{align*}
\]

Comments: The fractional part can be converted by multiplication. (This is the inverse of the division method for integers.)

\[
\begin{align*}
0.625 \times 2 &= 1.25 &\text{(the most significant bit in fraction)} \\
0.25 \times 2 &= 0.5 &\text{(the least significant bit in fraction)} \\
0.5 \times 2 &= 1.0 &\text{(the least significant bit in fraction)}
\end{align*}
\]

An Example (Cont.)

\[
y &= -4.75 &\rightarrow -100.11 \text{ (Binary)} \\
&\rightarrow -1.0011 \times 2^2 \text{ (Normal form)} \\
&\rightarrow -1.0011 \times 2^{128} \text{ (IEEE format)} \\
&\rightarrow 1 10000001 00110000000000000000000000000
\]

Step 2: Align two significands.

The significand of \( x = 1.0101 \rightarrow 0.10101 \) (After shift right 1 bit)

Comments: \( x = 0.10101 \times 2^{129} \) and \( y = -1.0011 \times 2^{129} \) after the alignment.

An Example (Cont.)

Step 3: Add two (adjusted) significands.

\[
\begin{align*}
0.10101 \rightarrow &\text{ The adjusted significand of } x \\
-1.00110 \rightarrow &\text{ The significand of } y \\
-0.10001 \rightarrow &\text{ The significand of } x+y
\end{align*}
\]

Step 4: Normalize the result.

\[
\begin{align*}
\text{Result} &= -0.10001 \times 2^{129} \rightarrow -1.0001 \times 2^{128} \\
&\rightarrow 1 10000000 000100000000000000000000000000000000
\end{align*}
\]

(Normal form)
Reading