

CSE 31

Computer Organization

Lecture 22 – Floating Point Numbers



Announcement

▶ Labs

- Lab 8 grace period ends this week and Lab 9 due this week with 1-week grace period
 - Demo is REQUIRED to receive full credit
- Lab 10 out this week
 - Due at 11:59pm on the same day of your next lab with 1-week grace period
 - You must demo your submission to your TA within 14 days

▶ Reading assignment

- Reading 07 (zyBooks 5.1 – 5.6) due **tonight**, 18-APR and Reading 08 (zyBooks 5.7 – 5.11) due on 2-MAY
 - Complete Participation Activities in each section to receive grade towards Participation
 - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcement

- ▶ Homework assignment
 - Homework 06 (zyBooks 5.1 - 5.11) due on 02-MAY
 - Complete *Challenge Activities* in each section to receive grade towards Homework
 - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses
- ▶ Midterm 2 details posted on CatCourses

Quote of the day

“95% of the folks out there are
completely clueless about
floating-point.”

James Gosling
Sun Fellow
Java Inventor
1998-02-28



Review of Numbers

- ▶ Computers are made to deal with numbers
- ▶ What can we represent in N bits?
 - 2^N things, and no more! They could be...

- Unsigned integers:

0 to $2^N - 1$

(for $N=32$, $2^N - 1 = 4,294,967,295$)

- Signed Integers (Two's Complement)

$-2^{(N-1)}$ to $2^{(N-1)} - 1$

(for $N=32$, $2^{(N-1)} = 2,147,483,648$)

What about other numbers?

1. Very large numbers? (seconds/millennium)
⇒ $31,556,926,000_{10}$ ($3.1556926_{10} \times 10^{10}$)
2. Very small numbers? (Bohr radius)
⇒ 0.0000000000529177_{10} m ($5.29177_{10} \times 10^{-11}$)
3. Numbers with both integer & fractional parts?
⇒ 1.5

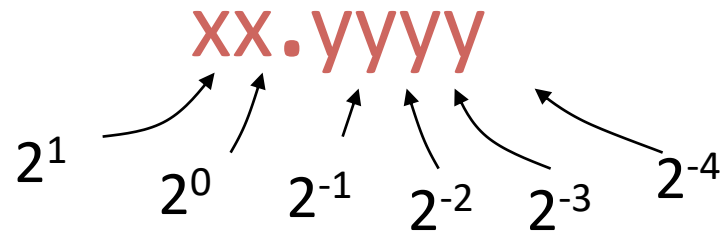
First consider #3.

...our solution will also help with 1 and 2.

Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:



$$10.1010_2 = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{10}$$

If we assume “**fixed binary point**”, with 6-bits, we can represent the following range of numbers using this format:

0 to 3.9375 (almost 4)

Fractional Powers of 2

i	2^{-i}	
0.	1.0	1
1.	0.5	1/2
2.	0.25	1/4
3.	0.125	1/8
4.	0.0625	1/16
5.	0.03125	1/32
6.	0.015625	
7.	0.0078125	
8.	0.00390625	
9.	0.001953125	
10.	0.0009765625	
11.	0.00048828125	
12.	0.000244140625	
13.	0.0001220703125	
14.	0.00006103515625	
15.	0.000030517578125	

Fractions with Fixed Points

What about addition and multiplication?

Addition is
straightforward:

$$\begin{array}{r} 01.100 \\ + 00.100 \\ \hline 10.000 \end{array}$$

1.5_{10}

0.5_{10}

2.0_{10}

$$\begin{array}{r} 01.100 \\ 00.100 \\ \hline \end{array}$$

1.5_{10}

0.5_{10}

Multiplication a bit more complex:

Where's the answer, 0.11 ?

(need to remember where the point is)

$$\begin{array}{r} 00\ 000 \\ 000\ 00 \\ 0110\ 0 \\ 00000 \\ 00000 \\ \hline 0000110000 \\ \underbrace{\hspace{1.5cm}}_{\text{HI}} \quad \underbrace{\hspace{1.5cm}}_{\text{LOW}} \end{array}$$

Representation of Fractions

So far, in our examples we used a “fixed” binary point

What we really want is to “**float**” the binary point. Why?

Floating binary point most effective use of our limited bits (and thus more accuracy in our number representation):

example: put 0.1640625 into binary. Represent in 5-bits
choosing where to put the binary point.

... 000000.001010100000...



Store these bits and keep track of the binary point 2
places to the left of the MSB

Any other solution would lose accuracy!

With floating point rep., each numeral carries an
exponent field recording the whereabouts of its
binary point.

The binary point **can be outside** the stored bits, so
very large and small numbers can be represented.

Scientific Notation (in Decimal)

mantissa exponent
 ↑
 decimal point
 radix (base)

$6.02_{10} \times 10^{23}$

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

Scientific Notation (in Binary)

A diagram illustrating the components of a floating-point number. The number is written as $1.0_{\text{two}} \times 2^{-1}$. Arrows point from labels to specific parts of the expression: 'mantissa' points to '1.0', 'exponent' points to '-1', 'radix (base)' points to '2', and '"binary point"' points to the decimal point in '1.0'.

- ▶ Computer arithmetic that supports it called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
 - Declare such variable in C as `float`

Floating Point Representation (1/2)

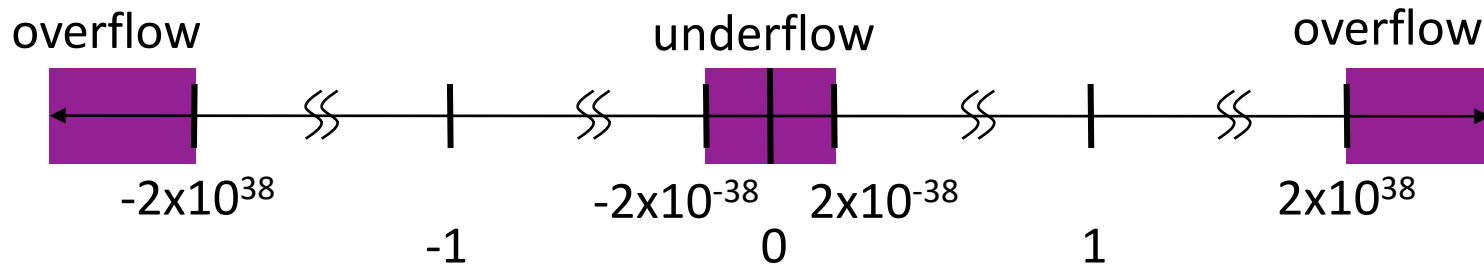
- ▶ Normal format: $+1.\text{xxx}\dots\text{x}_{\text{two}} * 2^{\text{yyy}\dots\text{y}_{\text{two}}}$
- ▶ Multiple of Word Size (32 bits)



- S represents Sign
- Exponent represents y's
- Significand represents x's
- Represent numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}

Floating Point Representation (2/2)

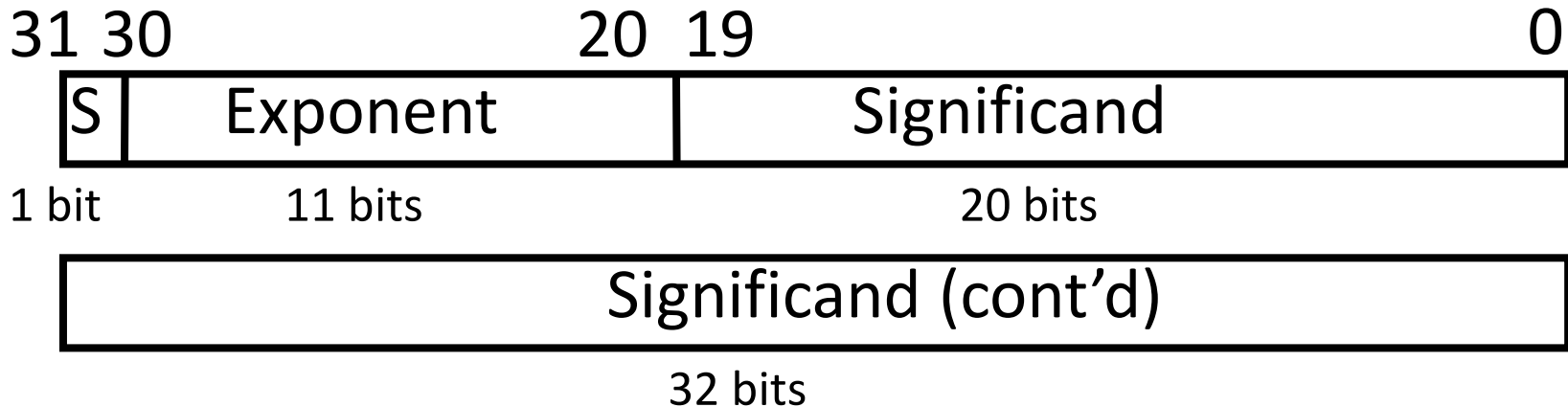
- ▶ What if result too large?
($> 2.0 \times 10^{38}$, $< -2.0 \times 10^{38}$)
 - Overflow! \Rightarrow Exponent larger than what can be represented in 8-bit Exponent field
- ▶ What if result too small?
($> 0 \text{ \& } < 2.0 \times 10^{-38}$, $> -2.0 \times 10^{-38} \text{ \& } < 0$)
 - Underflow! \Rightarrow Negative exponent larger than what can be represented in 8-bit Exponent field



- ▶ What would help reduce chances of overflow and/or underflow?

Double Precision Fl. Pt. Representation

- ▶ Next Multiple of Word Size (64 bits)



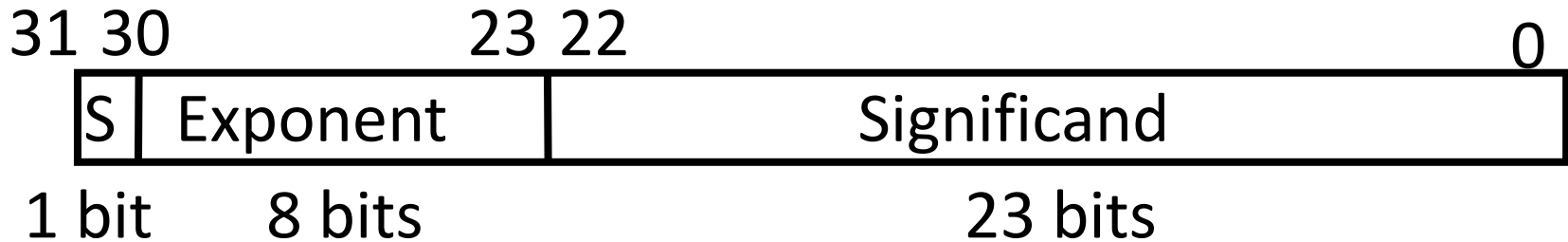
- Double Precision (vs. Single Precision)
 - C variable declared as `double`
 - Represent numbers almost as small as 2.0×10^{-308} to almost as large as 2.0×10^{308}
 - But primary advantage is greater accuracy due to larger significand

QUAD Precision Fl. Pt. Representation

- ▶ Next Multiple of Word Size (128 bits)
 - Unbelievable **range** of numbers
 - Unbelievable **precision** (accuracy)
- ▶ IEEE 754-2008 “binary128” standard
 - Has 15 exponent bits and 112 significand bits (can represent 113 precision bits)
- ▶ Oct-Precision?
 - Some have tried, no real traction so far
- ▶ Half-Precision?
 - Yep, “binary16”

IEEE 754 Floating Point Standard (1/3)

Single Precision (DP similar):



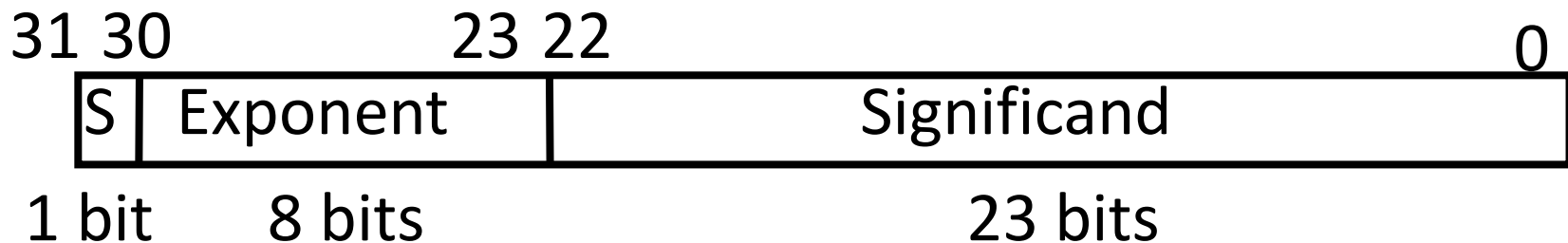
- ▶ Sign bit:
 - 1 means negative
 - 0 means positive
- ▶ Significand:
 - To pack more bits, leading 1 is implicit for normalized numbers
 - 1 + 23 bits single, 1 + 52 bits double
 - always true: $0 < \text{Significand} < 1$ (for normalized numbers)
- ▶ Note: 0 has no leading 1, so reserve exponent value 0 just for number 0

IEEE 754 Floating Point Standard (2/3)

- ▶ IEEE 754 uses “biased exponent” representation.
 - Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
 - Wanted bigger (integer) exponent field to represent bigger numbers.
 - 2's complement poses a problem (because negative numbers look bigger)
 - We're going to see that the numbers are ordered EXACTLY as in sign-magnitude
 - i.e., counting from binary odometer 00...00 up to 11...11 goes from 0 to +MAX to -0 to -MAX to 0

IEEE 754 Floating Point Standard (3/3)

- Called Biased Notation, where bias is number subtracted to get real number
 - IEEE 754 uses bias of 127 for single precision
 - Subtract 127 from Exponent field to get actual value for exponent
 - 1023 is bias for double precision
- ▶ Summary (single precision):



- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$
 - Double precision identical, except with exponent bias of 1023 (half, quad similar)

Understanding the Significand (1/2)

► Method 1 (Fractions):

- In decimal: $0.340_{10} \Rightarrow 340_{10}/1000_{10}$
 $\Rightarrow 34_{10}/100_{10}$
- In binary: $0.110_2 \Rightarrow 110_2/1000_2 = 6_{10}/8_{10}$
 $\Rightarrow 11_2/100_2 = 3_{10}/4_{10}$
- Advantage: less purely numerical, more thought oriented; this method usually helps people understand the meaning of the significand better

Understanding the Significand (2/2)

► Method 2 (Place Values):

- Convert from scientific notation
- In decimal:
 - $1.6732 = (1 \times 10^0) + (6 \times 10^{-1}) + (7 \times 10^{-2}) + (3 \times 10^{-3}) + (2 \times 10^{-4})$
- In binary:
 - $1.1001 = (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$
- Interpretation of value in each position extends beyond the decimal/binary point
- Advantage: good for quickly calculating significand value; use this method for translating FP numbers

Converting Binary FP to Decimal

0	0110 1000	101 0101 0100 0011 0100 0010
---	-----------	------------------------------

▶ Sign: 0 → positive

▶ Exponent:

- 0110 1000_{two} = 104_{ten}
- Bias adjustment: 104 - 127 = -23

▶ Significand:

$$\begin{aligned}
 &1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + \dots \\
 &= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-22} \\
 &= 1.0 + 0.666115
 \end{aligned}$$

- Represents: $1.666115_{\text{ten}} \times 2^{-23} \sim 1.986 \times 10^{-7}$
(about 2/10,000,000)

Converting Decimal to FP

-2.340625×10^1

1. Denormalize: -23.40625

2. Convert integer part:

$$23 = 16 + (7 = 4 + (3 = 2 + (1))) = 10111_2$$

3. Convert fractional part:

$$.40625 = .25 + (.15625 = .125 + (.03125)) = .01101_2$$

4. Put parts together and normalize:

$$10111.01101 = 1.011101101 \times 2^4$$

5. Convert exponent: $127 + 4 = 10000011_2$

1	1000 0011	011 1011 0100 0000 0000 0000
---	-----------	------------------------------

Quiz

1	1000 0001	111 0000 0000 0000 0000 0000
---	-----------	------------------------------

What is the decimal equivalent of the floating pt # above?

Note: $1000\ 0001_2 = 129_{10}$

$$2^{-1} = 0.5$$

$$2^{-2} = 0.25$$

$$2^{-3} = 0.125$$

- a) $-7 * 2^{129}$
- b) -3.5
- c) -3.75
- d) -7
- e) -7.5

Quiz Answer

What is the decimal equivalent of:

1	1000 0001	111 0000 0000 0000 0000 0000
---	-----------	------------------------------

S Exponent

Significand

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

$$(-1)^1 \times (1 + .111) \times 2^{(129-127)}$$

$$-1 \times (1.111) \times 2^{(2)}$$

$$-111.1$$

$$111.1 \Rightarrow 4 + 2 + 1 + .5 \Rightarrow 7.5$$

a) $-7 * 2^{129}$

b) -3.5

c) -3.75

d) -7

e) -7.5

Summary

- ▶ Floating Point lets us:
 - Represent numbers containing both integer and fractional parts; makes efficient use of available bits.
 - Store **approximate** values for very large and very small #s.
- ▶ **IEEE 754 Floating Point Standard** is most widely accepted attempt to standardize interpretation of such numbers (Every desktop or server computer sold since ~1997 follows these conventions)

single precision:



- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$
 - Double precision identical, except with exponent bias of 1023 (half, quad similar)

Converting Binary FP to Decimal (review)

0	0110 1000	101 0101 0100 0011 0100 0010
---	-----------	------------------------------

▶ Sign: 0 → positive

▶ Exponent:

- 0110 1000_{two} = 104_{ten}
- Bias adjustment: 104 - 127 = -23

▶ Significand:

$$\begin{aligned} & 1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + \dots \\ & = 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-22} \\ & = 1.0 + 0.666115 \end{aligned}$$

- Represents: $1.666115_{\text{ten}} \times 2^{-23} \sim 1.986 \times 10^{-7}$
(about 2/10,000,000)

Converting Decimal to FP (review)

-2.340625×10^1

1. Denormalize: -23.40625

2. Convert integer part:

$$23 = 16 + (7 = 4 + (3 = 2 + (1))) = 10111_2$$

3. Convert fractional part:

$$.40625 = .25 + (.15625 = .125 + (.03125)) = .01101_2$$

4. Put parts together and normalize:

$$10111.01101 = 1.011101101 \times 2^4$$

5. Convert exponent: $127 + 4 = 10000011_2$

1	1000 0011	011 1011 0100 0000 0000 0000
---	-----------	------------------------------

Example: Representing 1/3 in MIPS

▶ 1/3

$$= 0.33333..._{10}$$

$$= 0.25 + 0.0625 + 0.015625 + 0.00390625 + \dots$$

$$= 1/4 + 1/16 + 1/64 + 1/256 + \dots$$

$$= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + \dots$$

$$= 0.0101010101..._2 * 2^0$$

$$= \boxed{1}0101010101..._2 * 2^{-2}$$

- Sign: 0
- Exponent = $-2 + 127 = 125 = 01111101$
- Significand = 0101010101...

0	0111 1101	0101 0101 0101 0101 0101 010
---	-----------	------------------------------

Floating Point Fallacy

▶ FP add associative?

- $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, and $z = 1.0$
- $(x + y) + z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0$
 $= (0.0) + 1.0 = \underline{1.0}$
- $x + (y + z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0)$
 $= -1.5 \times 10^{38} + (1.5 \times 10^{38}) = \underline{0.0}$

▶ Therefore, Floating Point add is NOT associative!

- Why?
 - FP result approximates real result!
 - This example: 1.5×10^{38} is so much larger than 1.0 that $1.5 \times 10^{38} + 1.0$ in floating point representation is still 1.5×10^{38}

Precision and Accuracy

Don't confuse these two terms!

Precision is a count of the number bits in a computer word used to represent a value.

Accuracy is a measure of the difference between the actual value of a number and its computer representation.

High precision permits high accuracy but doesn't guarantee it. It is possible to have high precision but low accuracy.

Example: `float pi = 3.14;`
pi will be represented using all 24 bits of the significant (highly precise), but is only an approximation (not accurate).