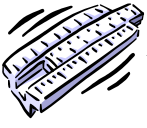



Floating Point Numbers



CS216
by Tom Eggers
used with permission


Fall, 2008
Charlie Shub - CS216 - Floating Point
Page 1



Alternatives for non integers

- String (or unpacked decimal)
 - 8 bits per decimal digit
 - Arithmetic ugly
- Packed Decimal
 - 4 bits per digit
 - Arithmetic ugly
- Fixed Point
 - Limited range
- Floating point
 - Good compromise


Fall, 2008
Charlie Shub - CS216 - Floating Point
Page 2



Topics

- Real numbers
- Scientific notation (decimal and binary)
- The number line (+ and – exponents, zero)
- Base conversions (F.P. decimal ↔ F.P. binary)
- IEEE Standard #754 (F.P. representation)
- Exceptions (overflow, NaN, underflow and zero)
- Extreme numbers
- Conversion exercises (F.P. decimal ↔ 754)

Fall, 2008
Charlie Shub - CS216 - Floating Point
Page 3



Real Numbers


Datatypes (in C, C++, Java, etc.): “float” or “double”
Used for fractions (non integers):

Rationals	6.0, 1/6, 2.5, -17.7
Irrationals	$\pi = 3.14159\dots$, $e = 2.71828\dots$

Used for the very large or small (scientific notation):

$-1.60217733 \times 10^{-19}$	(electron charge)
6.0221367×10^{23}	(Avogadro's number)
$\$5.7 \times 10^{12}$	(U.S. Federal debt in 2001)

Fall, 2008Fall, 2007
Charlie Shub - CS216 - Floating Point
Page 4



Scientific Notation (decimal)

$-3.60217733 \times 10^{-19}$

Leading sign
(Sign magnitude)

Exponent is
power of 10


Single leading digit
“Normalized” (not 0)
(Decimal point moved
and exponent adjusted)

Fraction digits

Decimal point

What is the normalized form of 123.456×10^6 ???

Fall, 2008
Charlie Shub - CS216 - Floating Point
Page 5



Scientific Notation (binary)

$-1.10111011 \times 2^{-5}$

Leading sign
(Sign magnitude)

Exponent is
power of 2

Single leading bit
“Normalized” (not 0)
(Binary point moved
and exponent adjusted)

Fraction bits

Binary point

What is the normalized form of $110.100110 \times 2^{0110}$???

Fall, 2008
Charlie Shub - CS216 - Floating Point
Page 6

UCCS The Number Line

Binary Floating Point using 8 bits:

1 sign bit: 0 or 1	4 exponent bits: +7 to -8	3 fraction bits: 1.000 to 1.111
-----------------------	------------------------------	------------------------------------

With exponent = 0: $N = 1.000 \times 2^0$ to 1.111×2^0

2 not 0 exp

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 7

UCCS The Number Line (cont.)

With exponent = 1: $N = 1.000 \times 2^1$ to 1.111×2^1

The representable numbers have spread out.

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 8

UCCS The Number Line (cont.)

With exponent = 2: $N = 1.000 \times 2^2$ to 1.111×2^2

Conclusions: Larger numbers are further apart.
Smaller numbers are closer together.
Relative error (LSB/N) \approx constant.

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 9

UCCS The Number Line (cont.)

Negative numbers & negative exponents:

There is no zero, and there is a "hole" $2 \times 2^{-6} = 2^{-5}$ wide, so ...

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 10

UCCS The Number Line (cont.)

Continue to the smallest exponent (-8):

Problems:

- (1) There is still no zero, and
- (2) The "hole" is now $2 \times 2^{-8} = 2^{-7}$ wide. The representable number spacing on either side is $2^{-8}/2^3 = 2^{-11}$, so the spacing jumps from 2^{-11} to 2^{-7} to 2^{-11} .

Goals:
Provide zero; fill hole with uniformly-spaced numbers.

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 11

UCCS The Number Line (cont.)

Special case:
When the exponent field is -8, do **not** insert the implied 1 to normalize the numbers.
 $.100 \times 2^{-8}$ changes to 0.100×2^{-7} , and the other numbers range from 0.000×2^{-7} to $\pm 0.111 \times 2^{-7}$.

Results:
There is a zero. (two)
The number spacing in the filled hole is $2^{-7}/2^3 = 2^{-10}$, and the spacing on either side is also 2^{-10} .

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 12

UCCS Base Conversions

Decimal	Binary	Normalized
1.0	1.0	1.0×2^0
6.5	110.1	1.101×2^2
2.25	10.01	1.001×2^1
-5.375	-101.011	-1.01011×2^2

Note: The normalized integer part is *always* exactly 1.

Step 0: Convert the exponent part (if any)
 Step 1: Convert the integer part
 Step 2: Convert the fraction part
 Rational numbers have a repeating digit group (e.g. 0...0) to ∞
 Irrational numbers (π , e , $\sqrt{2}$) do *not* repeat a digit group to ∞
 Step 3: Normalize

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 13

UCCS Base Conversions (cont.)

Example: $e = 2.71828..._{10}$

Step 1) Convert the integer: $2_{-10} \rightarrow 10_{-2}$
 Step 2) Convert the fraction: $0.71828_{10} \rightarrow 0.43656$
 a) Multiply *just the fraction* by 2: The left-most fraction bit is 0
 b) Multiply *just the fraction* by 2: The next right fraction bit is 0
 c) The next bits are: $e = 10.1011\ 0111\ 1110\ 0001\ 0101\ 0001...$
 Step 3) Normalize: $e = 1.0101\ 1011\ 1111\ 0000\ 1010\ 1000 \times 2^1$
 bit always = 1 \rightarrow not needed

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 14

UCCS Base Conversions (cont.)

Shortcut: do it in hex...

Same example: $e = 2.71828..._{10}$

Step 1) Convert the integer: $2_{-10} \rightarrow 2_{-16}$
 Step 2) Convert the fraction: $0.71828_{10} \rightarrow 0.43656_{16}$
 a) Multiply *just the fraction* by 16: The left-most hex digit is 11
 b) Multiply *just the fraction* by 16: The next right hex digit is 7
 c) The next hex digits are: $e = 2.b7e151..._{16}$
 d) Convert hex to binary: $e = 10.1011\ 0111\ 1110\ 0001\ 0101\ 0001...$
 Step 3) Normalize: $e = 1.0101\ 1011\ 1111\ 0000\ 1010\ 1000 \times 2^1$

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 15

UCCS Base Conversions (cont.)

Rational example: $n = 0.1_{10}$

Step 1) Convert integer (none)
 Step 2) Convert fraction \rightarrow Repeating digits $.00011$
 Shortcut using hex $\rightarrow 0.1999..._{16} = .0001\ 1001\ 1001\ 1001_2 \dots$
 Step 3) Normalize $\rightarrow n = 1.1001\ 1001\ 1001_2 \dots \times 2^{-4}$

Note: Rational numbers will *always* eventually repeat (the repeating group may be 0...0)

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 16

UCCS Base Conversions (cont.)

Example: $A = 6.0221367 \times 10^{23}$

Step 0) Convert the exponent (calculator required):
 $10^{23} = 2^x \rightarrow 23 = x \log_{10}(2) \rightarrow x = 76.404346$
 $\therefore 10^{23} = 2^{76.404346} = 2^{0.404346} \times 2^{76} = 1.323489 \times 2^{76}$
 $\therefore A = 6.0221367 \times (1.323489 \times 2^{76}) = 7.970231 \times 2^{76}$

Step 1) Convert the integer $_{10}$ to: 7_{-16}
 Step 2) Convert the fraction $_{10}$ to: $0.f8611a_{16}$
 $\therefore A = 7.f8611a_{16} \times 2^{76} = 111.1111\ 1000\ 0110\ 0001\ 0001\ 1010_2 \times 2^{76}$

Step 3) Normalize: $A = 1.1111\ 1110\ 0001\ 1000\ 0100\ 0110_2 \times 2^{78}$
 This bit is *always* = 1 \rightarrow not needed

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 17

UCCS IEEE Standard 754

The F.P. number, in binary scientific notation, must be converted to the "IEEE machine representation."

$A = +1.1111\ 1110\ 0001\ 1000\ 0100\ 0110_2 \times 2^{78}$

Discarded! 1

sign	8-bit exponent	23-bit fraction
0	1100 1101 ₂	1111 1110 0001 1000 0100 0111

Sign Magnitude $205_{10} \leftarrow 127_{10} + 78_{10}$ Bias on Exponent

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 18

UCCS IEEE 754 (cont.)

Q: Why the magic 127_{10} bias?

A: So *integer* compare instructions will work on F.P. numbers: an architecture goal

Consider two F.P. numbers (in 8-bits; 2's comp exponent):
Which is larger (as F.P.? as integer?)

1.0×2^1 :	0	0001	000
1.0×2^{-1} :	0	1111	000

Problem: exponent-sign bit makes number smaller as F.P. but larger as integer!

Solution: add **bias=7** (or 127) to exponent.

1.0×2^1 :	0	1000	000
1.0×2^{-1} :	0	0110	000

Larger F.P. number is larger as integer!

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 19

UCCS IEEE 754 (cont.)

Convert IEEE to decimal:
Example: $n = 0\ 1100\ 1101\ 1111\ 1110\ 0001\ 1000\ 0100\ 0111$

Step 1) Convert the exponent: IEEE_{exp} = $127_{10} \rightarrow$ decimal
 $205_{10} - 127_{10} = 78_{10} \rightarrow 2^{78}$

Step 2) Convert the fraction to hex and re-insert "discarded bit"
IEEE_{fraction} \rightarrow 1.hex
1.fe1846₁₆

Step 3) Convert the hex to decimal:
 $1.fe1846_{16} = 1 + \frac{15}{16^1} + \frac{14}{16^2} + \frac{1}{16^3} + \frac{8}{16^4} + \frac{4}{16^5} + \frac{6}{16^6} = 1.992557883_{10}$

Calculator shortcut:
 $1.fe1846_{16} = 1 + \frac{1}{16} \left(15 + \frac{1}{16} \left(14 + \frac{1}{16} \left(1 + \frac{1}{16} \left(8 + \frac{1}{16} \left(4 + \frac{6}{16} \right) \right) \right) \right) \right) \right)$

Step 4) Multiply results of steps 1 & 3 and set the sign:
 $n = 1.992557883 \times 2^{78} = +6.022136 \times 10^{23} = A$

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 20

UCCS IEEE 754 (cont.)

Convert a repeating fraction to decimal:
Example: the number extracted from the IEEE format is
 $n = 1.1001\ 1001\ 1001 \dots_2 \times 2^{-3}$

Identify the repeating group's size: group=1001; its **size = 4** bits
Multiply n by 2^{size} (shift left **size** bits), then subtract n :

$$n \times 2^4 = 16n = 1\ 1001.1001\ 1001\ 1001 \dots_2 \times 2^{-3}$$

$$n = 1.1001\ 1001\ 1001 \dots_2 \times 2^{-3}$$

$$n \times 2^4 - n = 16n - n = 15n = 1\ 1000.0_2 \times 2^{-3} = 24_{10} \times 2^{-3} = 24/8$$

$$\therefore n = (24/8)/15 = 24/(8 \times 15) = 0.2_{10} \text{ exactly!}$$

Repeating fractions are rational numbers.
The repeating groups can be "subtracted away" to exactly recover the rational number.

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 21

UCCS IEEE 754 (cont.)

Reserved exponent (all ones): 1111 1111 = 255_{10}

- $\pm\infty$ ("overflow") when all fraction bits = 0
- NaN ("Not a Number") when fraction bits \neq 0

Reserved exponent (all zeros): 0000 0000 = 0

- The integer 1 bit is **not** discarded
- The number is **not** normalized (allows "gradual underflow")
- The exponent is 2^{-126} (**not** 2^{-127})
- **Zero (0.0)** is

 all zero bits !!!

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 22

UCCS IEEE 754 (cont.)

"Gradual Underflow" and Zero

-2^{-124} -2^{-125} -2^{-126} **-gap+** $+2^{-126}$ $+2^{-125}$ $+2^{-124}$

Problems: the "gap" still remains; no way to represent zero
Solution: when $|n| < 2^{-126}$, keep the integer 1 bit and **UN**normalize

Results:

0	0000 0001	...	$\rightarrow \{ 1.11\dots11 \text{ to } 1.00\dots00 \} \times 2^{-126}$
0	0000 0000	...	$\rightarrow \{ 0.11\dots11 \text{ to } 0.00\dots00 \} \times 2^{-125}$

There are 2×2^{23} evenly-spaced numbers to fill the \pm gap, and ...

= **Zero**

Fall, 2008 Charlie Shub - CS216 - Floating Point Page 23

UCCS IEEE 754 (cont.)

Extreme numbers


Largest (**non-reserved**) number is

Exp = $254_{10} - 127_{10} = 127_{10}$
Significant bits = $1.111\dots1 \approx 2.0$
 $\therefore n = 2.0 \times 2^{127} = 3.4 \times 10^{38}$

Smallest (**non-reserved**) number is

Exp = $1 - 127_{10} = -126_{10}$
Significant bits = $1.000\dots0 = 1.0$
 $\therefore n = 1.0 \times 2^{-126} = 1.18 \times 10^{-38}$


Fall, 2008 Charlie Shub - CS216 - Floating Point Page 24



Conversion Exercises

0	0000 0000	0		= 0
0	0111 1111	0		= +1.0
0	1000 0000	1001		= +3.125
1	1000 0001	1011		= -6.75
0	0000 0001	1010 101		= $+1.956096 \times 10^{-38}$
1	1111 1111	0		= $-\infty$
0	1000 1001	1011 11		= +1776
1	1000 0100	011		= (Et tu Brutè)
0	1000 1001	1110 0101 01		= (Infamy)
0	0111 1111	0101 0101 0101 0101 0101 010		= 4/3
D e a d B e e f ₁₆				= -6.259853×10^{18}


Fall, 2008
Page 25
Charlie Shub - CS216 - Floating Point



Floating Point Arithmetic Operations

- **Multiply and Divide**
 - by a power of 2 – just change the exponent
 - otherwise
 - multiply/divide fractions
 - add/subtract exponents (careful with bias)
 - renormalize
- **Add and Subtract**
 - align (set exponents the same)
 - add/subtract fractions
 - renormalize

Fall, 2008
Page 26
Charlie Shub - CS216 - Floating Point



Arithmetic Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
 - two's complement
 - IEEE 754 floating point
- Computer instructions determine "meaning" of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines (i.e., algorithms and implementation).
- We are ready to move on (and implement the processor)

Fall, 2008
Page 27
Charlie Shub - CS216 - Floating Point