EECS 150 - Components and Design Techniques for Digital Systems

Lec 19 – Fixed Point & Floating Point Arithmetic

10/23/2007

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://inst.eecs.berkeley.edu/~cs150

Outline

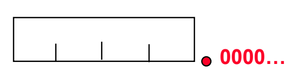
- Review of Integer Arithmetic
- Fixed Point
- IEEE Floating Point Specification
- Implementing FP Arithmetic (interactive)

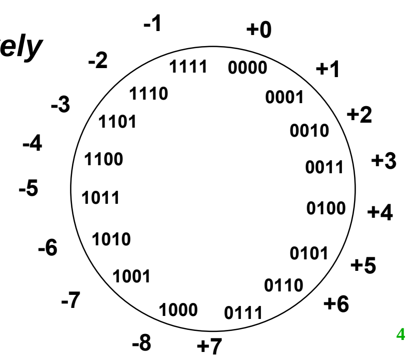
Representing Numbers

- What can be represented in N bits?
  - 2^N distinct symbols => values
  - Unsigned 0 to $2^N - 1$
  - 2s Complement $-2^{(N-1)}$ to $2^{(N-1)} - 1$
  - ASCII $-10^{(N/8-2)} - 1$ to $10^{(N/8-1)} - 1$
- But, what about?
  - Very large numbers? (seconds/century)
    3,155,760,000$_{ten}$ (3.15576$_{ten}$ x 10^9)
  - Very small numbers? (secs/ nanosecond)
    0.000000001$_{ten}$ (1.0$_{ten}$ x 10^{-9})
  - Bohr radius
    ⇒ 0.0000000000529177$_{10}$m (5.29177$_{10}$ x 10^{-11})
  - Rationals 2/3 (0.666666666. . .)
  - Irrationals $2^{1/2}$ (1.414213562373. . .)
  - Transcendentals e (2.718...), π (3.141...)

Recall: Digital Number Systems

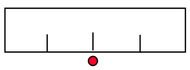
- Positional notation
  - $D_{n-1}\,D_{n-2}\ldots D_0$ represents $D_{n-1}B^{n-1} + D_{n-2}B^{n-2} + \ldots + D_0\,B^0$
    where $D_i \in \{0, \ldots, B-1\}$
- 2s Complement
  - $D_{n-1}\,D_{n-2}\ldots D_0$ represents: $- D_{n-1}2^{n-1} + D_{n-2}2^{n-2} + \ldots + D_0\,2^0$
  - MSB has *negative weight*
- *Binary Point is effectively at the far right of the word*

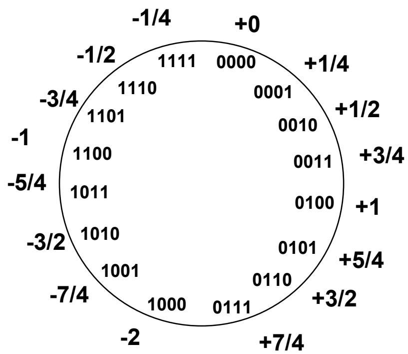




Representing Fractional Numbers

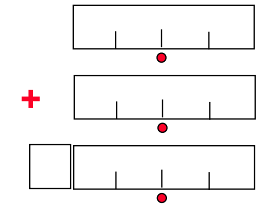
- **Fixed-Point Positional notation**
  - $D_{n-k-1}\, D_{n-k-2} \ldots D_{0 \ldots} D_{-k}$ represents $D_{n-k-1}B^{n-k-1} + D_{n-2}B^{n-2} + \ \ldots + D_{-k}\, B^{-k}$ where $D_i \in \{ 0, \ldots, B-1 \}$
- **2s Complement**
  - $D_{n-k-1}\, D_{n-2} \ldots D_{-k}$ represents: $- D_{n-k-1}2^{n-k-1} + D_{n-2}2^{n-2} + \ \ldots + D_{-k}\, 2^{-k}$

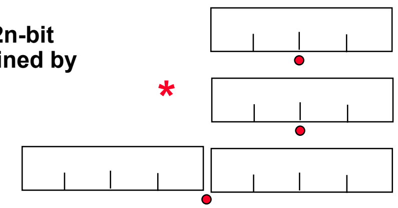




Circuits for Fixed-Point Arithmetic

- **Adders?**
  - identical circuit
  - Position of the binary point is entirely in the interpretation
  - Be sure the interpretations match
    - » i.e. binary points line up
- **Subtractors?**
- **Multipliers?**
  - Position of the binary point just as you learned by hand
  - Mult two n-bit numbers yields 2n-bit result with binary point determined by binary point of the inputs
  - $2^{-k} * 2^{-m} = 2^{-k-m}$

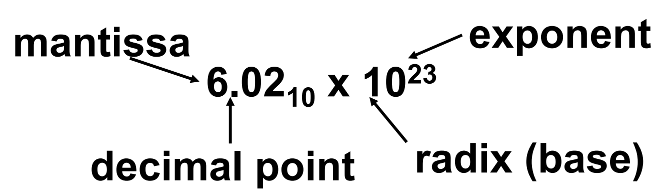




How do you represent…

- **Very big numbers - with a few characters?**
- **Very small numbers – with a few characters?**

Scientific Notation



- **Normalized form: no leadings 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)

mantissa → $1.0_{two} \times 2^{-1}$ ← exponent

"binary point"   radix (base)

- Computer arithmetic that directly supports this kind of representation called floating point, because it represents numbers where the binary point is not in a fixed position, but "floats".
  - Declared in C as `float`
- Floats are more like "reals" than integers, but they are not. They have a finite representation.

UCB's "Father" of IEEE Floating point

IEEE Standard 754 for Binary Floating-Point Arithmetic.

1989
ACM Turing
Award Winner!

Prof. Kahan

`www.cs.berkeley.edu/~wkahan/`
`…/ieee754status/754story.html`

IEEE Floating Point Representation

- Normal format: $+1.\text{xxxxxxxxxx}_{two} * 2^{\text{yyyy}_{two}}$
- Multiple of Word Size (32 bits)

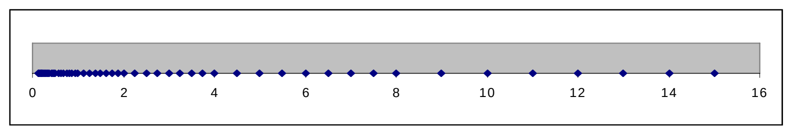
31	30 … 23	22 … 0
S	Exponent	Significand
1 bit	8 bits	23 bits

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

Hidden 1   excess 127

- Single precision represents numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}

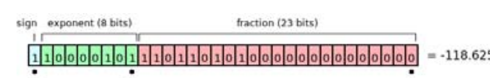
Which 2^N numbers can you represent?



- 8 million equally spaced values, between …
  - 1 and 2
  - -1.0 and -0.5 (-2^0 and -2^{-1})
  - 2^{-125} and 2^{-124}
  - 2^{124} and 2^{125}

Each successive power of two

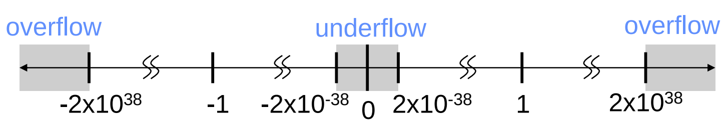
- Which integers are represented exactly?
- Which are not?
- Which fractions?
- Where is there a gap?



Floating Point Representation

- What if result too large (in magnitude)?
  (> 2.0×10^{38} , < -2.0×10^{38})
  - Overflow! ⇒ Exponent larger than represented in 8-bit Exponent field
- What if result too small (in magnitude)?
  (>0 & < 2.0×10^{-38} , <0 & > -2.0×10^{-38})
  - Underflow! ⇒ Negative exponent larger than represented in 8-bit Exponent field

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

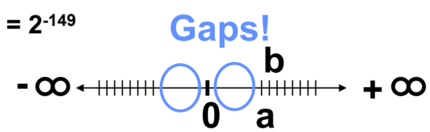


Denorms

- Problem: if A ≠ B then is A-B ≠ 0?
- gap among representable FP numbers around 0
  - Smallest representable pos num:
    $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - Second smallest representable pos num:
    $b = 1.000\ldots\ldots1_2 * 2^{-126}$
    $= (1 + 0.00\ldots1_2) * 2^{-126}$
    $= (1 + 2^{-23}) * 2^{-126}$
    $= 2^{-126} + 2^{-149}$

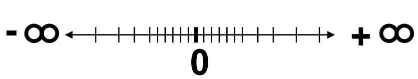
  $a - 0 = 2^{-126}$

  $b - a = 2^{-149}$



Denorms

- **Solution:**
  - Denormalized number: no (implied) leading 1, implicit exponent = -126.
  - Exponent = 0, Significand nonzero
  - Smallest representable pos num:
    $a = 2^{-149}$
  - Second smallest representable pos num:
    $b = 2^{-148}$
- What do you give up for A ≠ B => A-B ≠ 0 ?
  - Multiplicative inverse: If A exists 1/A exists



Announcements

- Readings: http://en.wikipedia.org/wiki/IEEE_754
- Labs
  - Free week inserted now, remove one check point, back off the options at the end
  - Design review will stay on schedule
    - » More time between review and implementation
    - » Take the prep for design review seriously
- Discuss Thurs discussion
- Party Problem
- Lab 5 code walk through on Friday
- Mid term II on 11/1, review 10/30 at 8 pm

Special IEEE 754 Symbols: Infinity

- Overflow is not same as divide by zero
- IEEE 754 represents +/- infinity
  - OK to do further computations with infinity e.g., X/0 > Y may be a valid comparison
  - Most positive exponent reserved for infinity

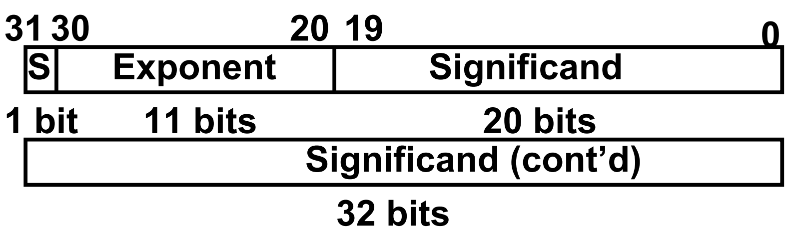
Exponent	*Significand*		*Object*
0	0	=>	0
0	nonzero	=>	denorm
1-254	anything	=>	+/- fl. pt. #
255	0	=>	+/- ∞
255	nonzero	=>	NaN

Examples

Type	Exponent	Significand	Value
Zero	0000 0000	000 0000 0000 0000 0000 0000	0.0
One	0111 1111	000 0000 0000 0000 0000 0000	1.0
Small denormalized number	0000 0000	000 0000 0000 0000 0000 0001	1.4×10^{-45}
Large denormalized number	0000 0000	111 1111 1111 1111 1111 1111	1.18×10^{-38}
Large normalized number	1111 1110	111 1111 1111 1111 1111 1111	3.4×10^{38}
Small normalized number	0000 0001	000 0000 0000 0000 0000 0000	1.18×10^{-38}
Infinity	1111 1111	000 0000 0000 0000 0000 0000	Infinity
NaN	1111 1111	non zero	NaN

Double Precision FP Representation

- Next Multiple of Word Size (64 bits)

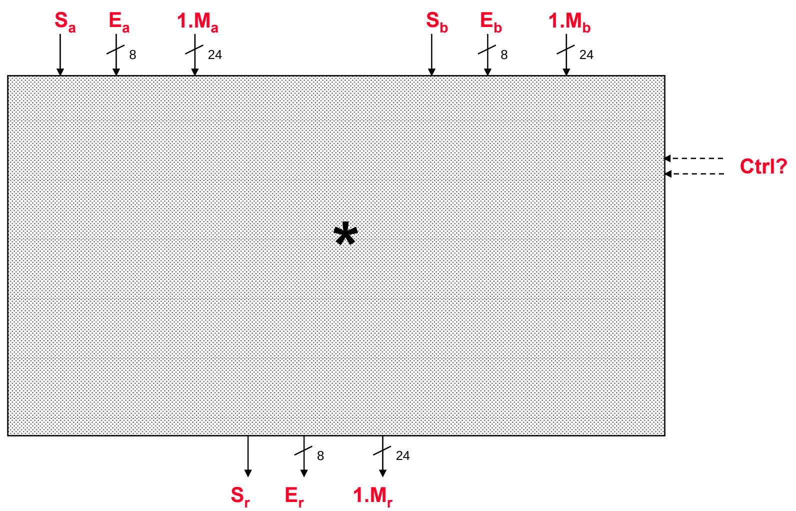


- Double Precision (vs. Single Precision)
  - C variable declared as `double`
  - Represent numbers almost as small as 2.0×10^{-308} to almost as large as 2.0×10^{308}
  - But primary advantage is greater accuracy due to larger significand

How do we do arithmetic on FP?

- Just like with scientific notation
- Addition
  - Eg. $9.45 \times 10^3 + 6.93 \times 10^2$
  - Shift mantissa so that have common exponent (unnormalize)
  - $9.45 \times 10^3 + 0.693 \times 10^3$
  - Add mantissas: 10.143×10^3
  - Renormalize: 1.0143×10^4
  - Round: 1.01×10^4
- IEEE rounding – as if had carried full precision and rounded at the last step
- Multiplication?

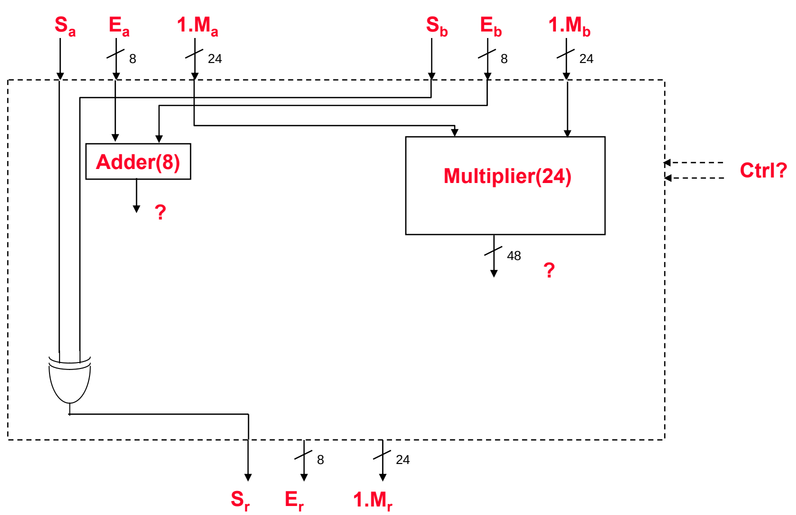
Let's build an FP function unit: mult



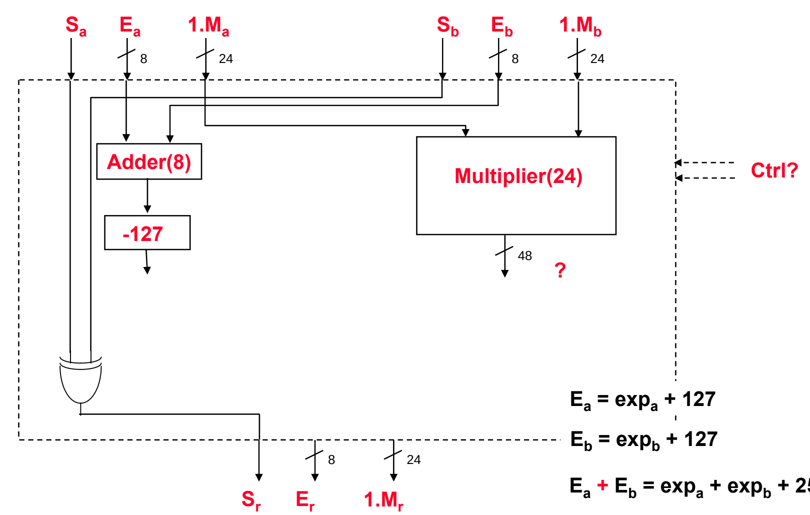
What is the multiplication algorithm?

- 9.45×10^3 * 6.93×10^2

Let's build an FP function unit: mult



Let's build a FP function unit: mult

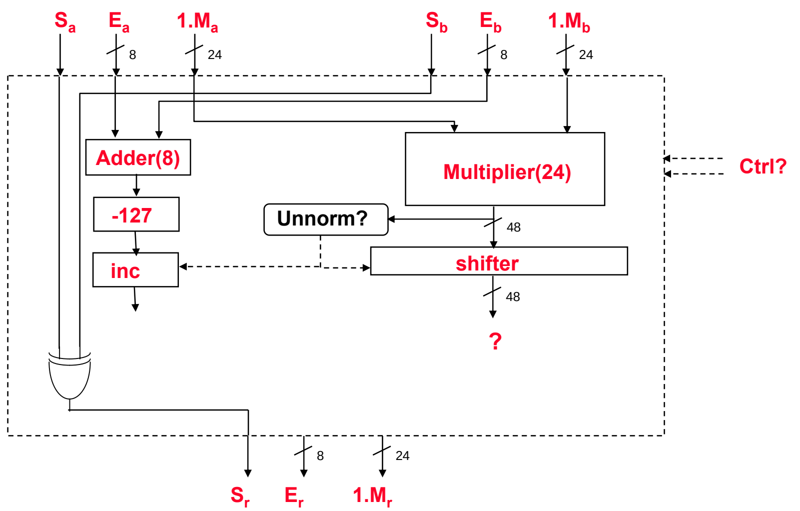


$E_a = exp_a + 127$

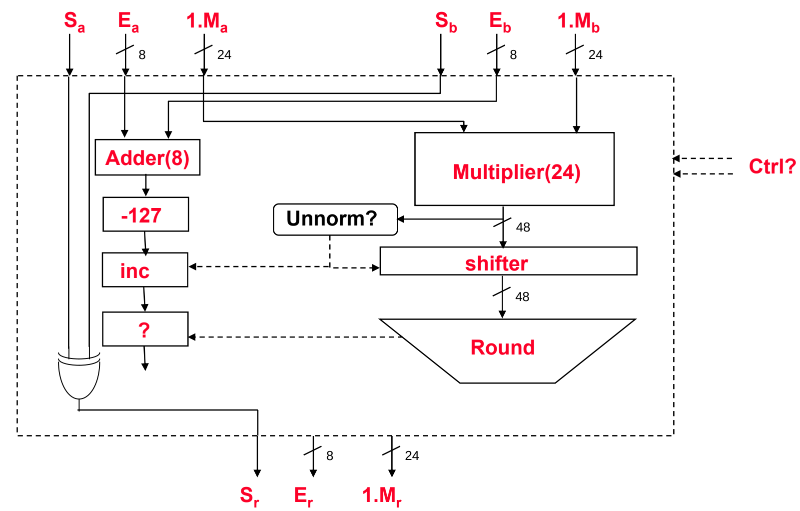
$E_b = exp_b + 127$

$E_a + E_b = exp_a + exp_b + 254$!

What is the range of mantissas?

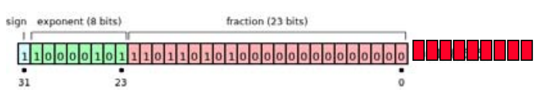


What is the range of mantissas?



Rounding

- Real numbers have "inifinite precision", FPs don't.
- When we perform arithmetic on FP numbers, we must round to fit the result in the significand field.



- IEEE FP behaves as if all internal operations were performed to full precision and then rounded at the end.
  - Actually only carries 3 extra bits along the way
    - » Guard bit | Round bit | Sticky bit

IEEE FP Rounding Modes

- Round towards +∞
  - Decimal: 1.1 → 1, 1.9 → 2, 1.5 → 2, -1.1 → -1, -1.9 → -2, -1.5 → -1,
  - Binary: 1.01 → 1, 1.11 → 10, 1.1 → 10, -1.01 → -1, -1.11 → -10, -1.1 → -1,
  - What is the accumulated bias with a large number of operations?
- Round towards - ∞
  - Decimal: 1.1 → 1, 1.9 → 2, 1.5 → 1, -1.1 → -1, -1.9 → -2, -1.5 → -2,
  - Binary: 1.01 → 1, 1.11 → 10, 1.1 → 1, -1.01 → -1, -1.11 → -10, -1.1 → -10,
  - What is the accumulated bias with a large number of operations?
- Round Towards Zero - Truncate
  - Decimal: 1.1 → 1, 1.9 → 2, 1.5 → 1, -1.1 → -1, -1.9 → -2, -1.5 → -1,
  - Binary: 1.01 → 1, 1.11 → 10, 1.1 → 1, -1.01 → -1, -1.11 → -10, -1.1 → -1,
  - What is the accumulated bias with a large number of operations?
- Round to even - Unbiased (default mode).
  - Decimal: 1.1 → 1, 1.9 → 2, 1.5 → 2, -1.1 → -1, -1.9 → -2, -1.5 → -2, 2.5 → 2, -2.5 → -2
  - Binary: 1.01 → 1, 1.11 → 10, 1.1 → 10, -1.01 → -1, -1.11 → -10, -1.1 → -1, 10.1 → 10, -10.1 → -10
  - if the value is right on the borderline, we round to the nearest EVEN number
  - This way, half the time we round up on tie, the other half time we round down.

Basic FP Addition Algorithm

For addition (or subtraction) of X to Y (assuming X<Y):

(1) Compute D = Exp_Y - Exp_X (align binary point)

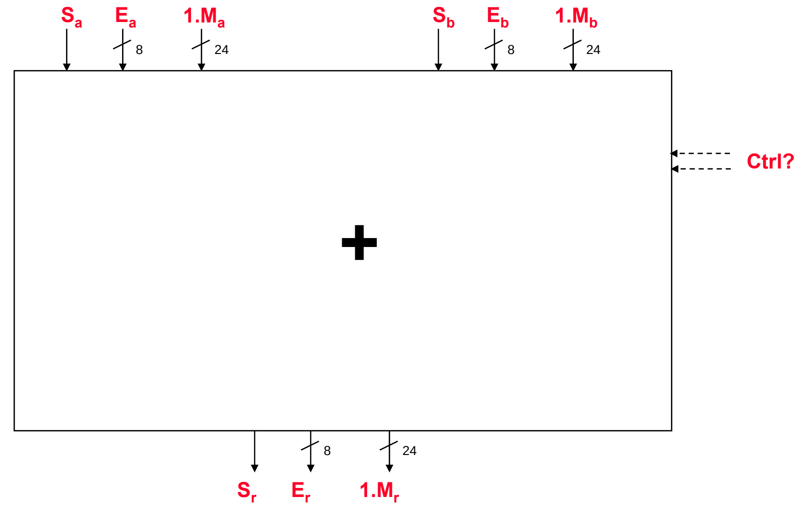
(2) Right shift ($1+Sig_X$) D bits => $(1+Sig_X)*2^{(ExpX-ExpY)}$

(3) Compute $(1+Sig_X)*2^{(ExpX - ExpY)} + (1+Sig_Y)$

Normalize if necessary; continue until MS bit is 1
(4) Too small (e.g., 0.001xx...)
left shift result, decrement result exponent
(4') Too big (e.g., 101.1xx…)
right shift result, increment result exponent

(5) If result significand is 0, set exponent to 0

Let's build an FP function unit: add



Floating Point Fallacies: Add Associativity?

- x = – 1.5 x 10^{38}, y = 1.5 x 10^{38}, and z = 1.0
- x + (y + z) = –1.5x10^{38} + (1.5x10^{38} + 1.0)
  = –1.5x10^{38} + (1.5x10^{38}) = 0.0
- (x + y) + z = (–1.5x10^{38} + 1.5x10^{38}) + 1.0
  = (0.0) + 1.0 = 1.0
- Therefore, Floating Point add not associative!
  – 1.5 x 10^{38} is so much larger than 1.0 that 1.5 x 10^{38} + 1.0 is still 1.5 x 10^{38}
  – Fl. Pt. result approximation of real result!

Floating Point Fallacy: Accuracy optional?

- July 1994: Intel discovers bug in Pentium
  – Occasionally affects bits 12-52 of D.P. divide
- Sept: Math Prof. discovers, put on WWW
- Nov: Front page trade paper, then NYTimes
  – Intel: "several dozen people that this would affect. So far, we've only heard from one."
  – Intel claims customers see 1 error/27000 years
  – IBM claims 1 error/month, stops shipping
- Dec: Intel apologizes, replace chips $300M
- Reputation? What responsibility to society?

Arithmetic Representation

- **Position of binary point represents a trade-off of range vs precision**
  - **Many digital designs operate in fixed point**
    - » **Very efficient, but need to know the behavior of the intended algorithms**
    - » **True for many software algorithms too**
  - **General purpose numerical computing generally done in floating point**
    - » **Essentially scientific notation**
    - » **Fixed sized field to represent the fractional part and fixed number of bits to represent the exponent**
    - » $\boxed{\pm}$ `1.`$\boxed{\texttt{fraction}}$ `x 2^` $\boxed{\texttt{exp}}$
  - **Some DSP algorithms used block floating point**
    - » **Fixed point, but for each block of numbers an additional value specifies the exponent.**