Review of Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
  - Unsigned integers:
    \[ 0 \text{ to } 2^N - 1 \]
  - Signed Integers (Two’s Complement)
    \[ -2^{(N-1)} \text{ to } 2^{(N-1)} - 1 \]

Other Numbers

- What about other numbers?
  - Very large numbers? (seconds/century)
    \[ 3,155,760,000 \text{ (3.15576 \times 10^8) } \]
  - Very small numbers? (atomic diameter)
    \[ 0.00000001 \text{ (1.0 \times 10^{-9}) } \]
  - Rationals (repeating pattern)
    \[ \frac{2}{3} \text{ (0.666666666...)} \]
  - Irrationals
    \[ \frac{\sqrt{2}}{2} \text{ (1.414213562373...)} \]
  - Transcendentals
    \[ e \text{ (2.718...), } \pi \text{ (3.141...)} \]

Scientific Notation (in Decimal)

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

Scientific Notation (in Binary)

- 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: \[ +1.xxxxxxxx_{two} \times 2^{y_{two}} \]
- Multiple of Word Size (32 bits)

\[
\begin{array}{ccc}
31 & 30 & 22 \\
S & \text{Exponent} & \text{Significand} \\
1 \text{ bit} & 8 \text{ bits} & 23 \text{ bits}
\end{array}
\]

- \( S \) represents Sign
- \( \text{Exponent} \) represents y’s
- \( \text{Significand} \) represents x’s
- Represent numbers as small as \[ 2.0 \times 10^{-38} \text{ to as large as } 2.0 \times 10^{38} \]
Floating Point Representation (2/2)

- What if result too large? (> $2.0 \times 10^{38}$)
  - **Overflow!**
    - Overflow ⇒ Exponent larger than represented in 8-bit Exponent field
- What if result too small? (>0, < $2.0 \times 10^{-38}$)
  - **Underflow!**
    - Underflow ⇒ Negative exponent larger than represented in 8-bit Exponent field
- How to reduce chances of overflow or underflow?

Double Precision Fl. Pt. Representation

- Next Multiple of Word Size (64 bits)
  
<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 20 19 0</td>
<td>1 bit 11 bits 20 bits</td>
</tr>
</tbody>
</table>

Significand (cont'd)

32 bits

- **Double Precision** (vs. Single Precision)
  - C variable declared as double
  - Represent numbers almost as small as $2.0 \times 10^{-308}$ to almost as large as $2.0 \times 10^{308}$
  - 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)
  - This is currently being worked on
  - The current version has 15 bits for the exponent and 112 bits for the significand
  - Oct-Precision? That’s just silly! It’s been implemented before...

IEEE 754 Floating Point Standard (1/4)

- Single Precision, DP similar
  - Sign bit:
    - 1 means negative
    - 0 means positive
  - Significand:
    - To pack more bits, leading 1 implicit for normalized numbers
    - $1 + 23$ bits single, $1 + 52$ bits double
    - always true: 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 (2/4)

- Kahan wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
  - Could break FP number into 3 parts: compare signs, then compare exponents, then compare significands
  - Wanted it to be faster, single compare if possible, especially if positive numbers
  - Then want order:
    - Highest order bit is sign (negative < positive)
    - Exponent next, so big exponent ⇒ bigger #
    - Significand last: exponents same ⇒ bigger #

IEEE 754 Floating Point Standard (3/4)

- Negative Exponent?
  - $2$’s comp? $1.0 \times 2^{-1}$ v. $1.0 \times 2^{+1}$ (1/2 v. 2)
  - This notation using integer compare of 1/2 v. 2 makes 1/2 > 2!
  - Instead, pick notation 0000 0001 is most negative, and 1111 1111 is most positive
  - $1.0 \times 2^{-1}$ v. $1.0 \times 2^{+1}$ (1/2 v. 2)
IEEE 754 Floating Point Standard (4/4)

- Called Biased Notation, where bias is number subtract to get real number
- IEEE 754 uses bias of 127 for single prec.
- Subtract 127 from Exponent field to get actual value for exponent
- 1023 is bias for double precision

**Summary (single precision):**

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
<tr>
<td>((-1)^{x} \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Double precision identical, except with exponent bias of 1023

**"Father" of the Floating point standard**

IEEE Standard 754 for Binary Floating-Point Arithmetic.

1989 ACM Turing Award Winner!

Prof. Kahan

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

Administrivia...Midterm in 2 weeks!

- Midterm 1 LeConte Mon 2004-03-07 @ 7-10pm
  - Conflicts/DSP? Email Head TA Andy, cc Dan
- How should we study for the midterm?
  - Form study groups – don’t prepare in isolation!
  - Attend the review session (2004-03-06 @ 2pm in 10 Evans)
  - Look over HW, Labs, Projects
  - Write up your 1-page study sheet-handwritten
  - Go over old exams – HKN office has put them online (link from 61C home page)

Upcoming Calendar

<table>
<thead>
<tr>
<th>Week #</th>
<th>Mon</th>
<th>Wed</th>
<th>Thurs Lab</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6</td>
<td>Holiday</td>
<td>Floating Pt I</td>
<td>Floating Pt</td>
<td>Floating Pt II</td>
</tr>
<tr>
<td>#7</td>
<td>Next week</td>
<td>MIPS inst. Format III</td>
<td>Running Program</td>
<td>Running Program</td>
</tr>
<tr>
<td>#8</td>
<td>Midterm week</td>
<td>Digital Systems</td>
<td>State Elements</td>
<td>Comb. Logic</td>
</tr>
</tbody>
</table>

Understanding the Significand (1/2)

- Method 1 (Fractions):
  - In decimal: \(0.340 = 340_{0}/1000_{0}\)
    \(\Rightarrow 340_{10}/100_{10}\)
  - In binary: \(0.110_{2} = 110_{2}/1000_{2}\)
    \(\Rightarrow 11_{2}/100_{2} = 3_{10}/4_{10}\)
  - 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: \(1.6732 = (1x10^0) + (6x10^{-1}) + (7x10^{-2}) + (3x10^{-3}) + (2x10^{-4})\)
  - In binary: \(1.1001 = (1x2^0) + (1x2^{-1}) + (0x2^{-2}) + (0x2^{-3}) + (1x2^{-4})\)
  - 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
Example: Converting Binary FP to Decimal

0110 1000 101 0101 0100 0011 0100 0010

• Sign: 0 ⇒ positive
• Exponent:
  - 0110 1000two = 104ten
  - Bias adjustment: 104 - 127 = -23
• Significand:
  - 1 + 1x2^-1 + 0x2^-2 + 1x2^-3 + 0x2^-4 + 1x2^-5 +...
  - = 1.666115ten * 2^-23
• Represents: 1.666115ten * 2^-23 ~ 1.986*10^-7
  (about 2/10,000,000)

Converting Decimal to FP (1/3)

• Simple Case: If denominator is an exponent of 2 (2, 4, 8, 16, etc.), then it’s easy.
• Show MIPS representation of -0.75
  - -0.75 = -3/4
  - -11two/100two = -0.11two
  - Normalized to -1.1two x 2^-1
  - (-1)^0 x (1 + Significand) x 2^(Exponent-127)
  - (-1)^0 x (1 + .100 0000 ... 0000) x 2^(126-127)

10111 1110 100 0000 0000 0000 0000

Converting Decimal to FP (2/3)

• Not So Simple Case: If denominator is not an exponent of 2.
  - Then we can’t represent number precisely, but that’s why we have so many bits in significand: for precision
  - Once we have significand, normalizing a number to get the exponent is easy.
  - So how do we get the significand of a neverending number?

Converting Decimal to FP (3/3)

• Fact: All rational numbers have a repeating pattern when written out in decimal.
• Fact: This still applies in binary.
• To finish conversion:
  - Write out binary number with repeating pattern.
  - Cut it off after correct number of bits (different for single v. double precision).
  - Derive Sign, Exponent and Significand fields.

Peer Instruction

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

1: -1.75
2: -3.5
3: -3.75
4: -7
5: -7.5
6: -15
7: -7.5
8: -129
9: -129 + 2^7

“And in conclusion...”

• Floating Point numbers approximate values that we want to use.
• 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
• Summary (single precision):

<table>
<thead>
<tr>
<th></th>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 30</td>
<td>23 22</td>
</tr>
</tbody>
</table>

1 bit 8 bits 23 bits
(-1)^8 x (1 + Significand) x 2^(Exponent-127)

• Double precision identical, bias of 1023