IEEE754浮点数

chaishushan@gmail.com

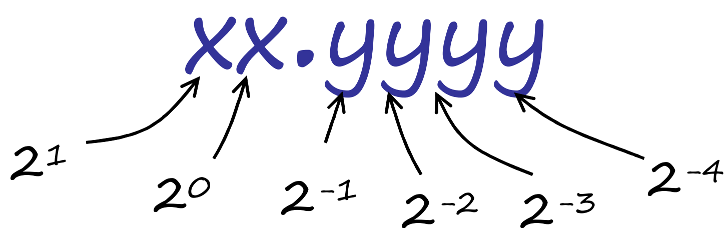
if(x+1 == x): x = ?

N个bit可以表示什么?

- **2^N**个东西, 不会更多...
  - uint: 0 ~ 2^N-1
  - int: -2^(N-1) ~ 2^(N-1) - 1
  - 2^32 == 4,294,967,295
- 其他的数呢**?**
  - 很大的数, 例如: 6.02*10^23
  - 很小的数, 例如: 9.10938188*10^(-31)
  - 有整数又有小数的数, 例如: 1.5

二进制小数

和十进制一样, 小数点指定整数和小数边界.
例如, 6个bit位:



10.10102 = 1x21 + 1x2-1 + 1x2-3 = 2.62510

假设小数点固定, **6bit**可以表示范围:
0 ~ 3.9375 (约等于4)

2^(-i)对应当值

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	

小数点固定

加法的结果:

01.100	1.5_{10}
00.100	0.5_{10}
10.000	2.0_{10}

乘法会更复杂一些:

01.100	1.5_{10}
00.100	0.5_{10}
00 000	
000 00	
0110 0	
00000	
00000	

$$\underbrace{00001}_{\mathrm{HI}}\underbrace{10000}_{\mathrm{LOW}}$$

答案是, 0.11? (需要记住小数点的位置)

小数点浮动

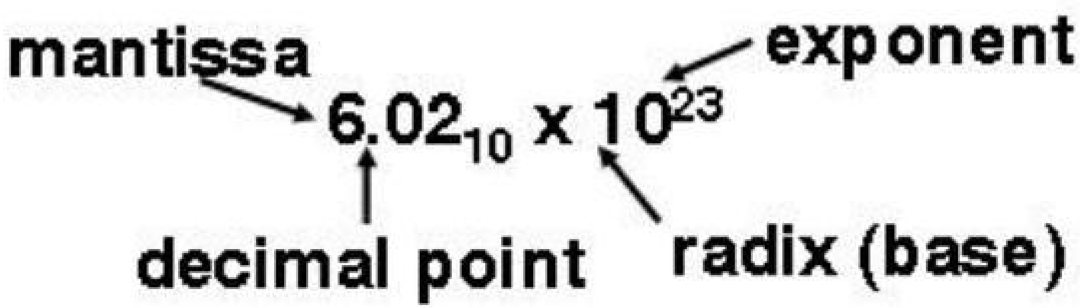
原因: **浮点数能高效利用有限的二进制位,**
从而有更高的精度.

例如: 将 0.1640625 表为二进制.

5-位, 定小数点: 000000**.00101**0100000

浮点表示中, 每个数都有指数域来记录小数点的位置.
小数可以在存储位之外, 因此可表示很大或很小的数.

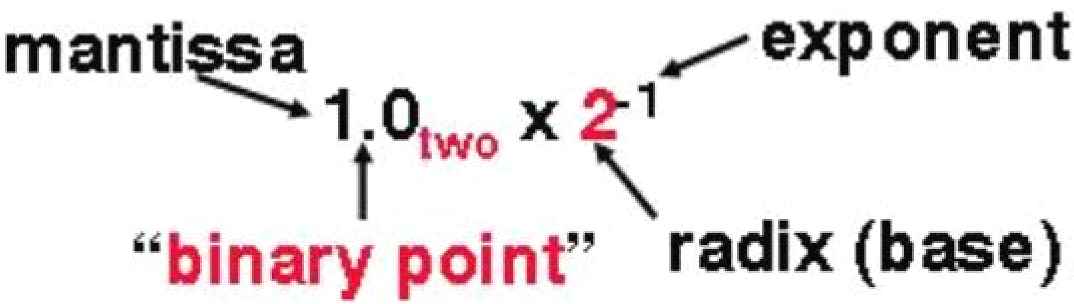
科学计数法(十进制)



规范化形式:

无前导**0**(小数点左边仅有1位非0数字)

科学计数法(二进制)

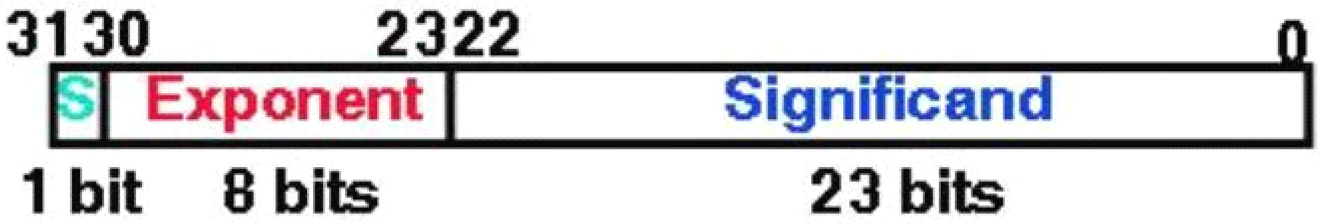


- 计算机中有专门的运算指令, 称为 **floating point**, 表示小数点位置是不固定的.
- C语言中对于**float**类型.

浮点表示: 普通形式

$+x.xxxxxxxxxx_{two} * 2^{yyyy_{two}}$

浮点表示: 规范化形式



- S - 为符号位
- **E** - 为指数位
- **Significand** - 有效数
- 表示范围: $2.0 \times 10^{-38} \sim 2.0 \times 10^{38}$

浮点数表示: 太 大~小？

- 如果结果太大，怎么办?

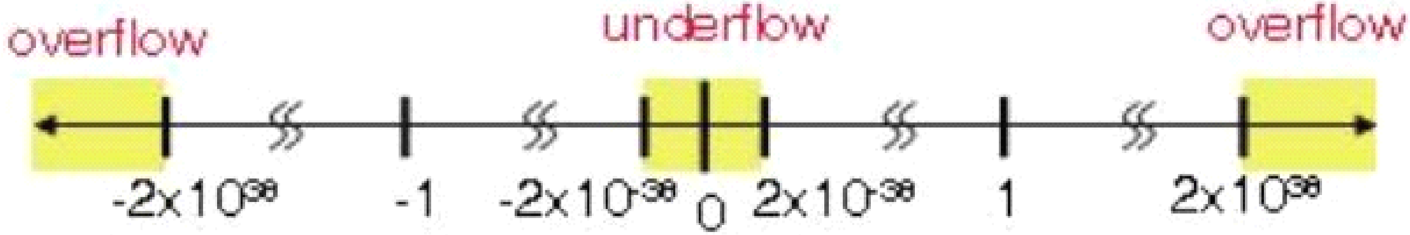
  ($>2.0\times10^{38}$, $< -2.0\times10^{38}$)

  - Overflow上溢! ⇒ 正指数大于8位指数字段可表示范围

- 如果结果太小，怎么办?

  (>0 & $<2.0\times10^{-38}$, <0 & $> -2.0\times10^{-38}$)

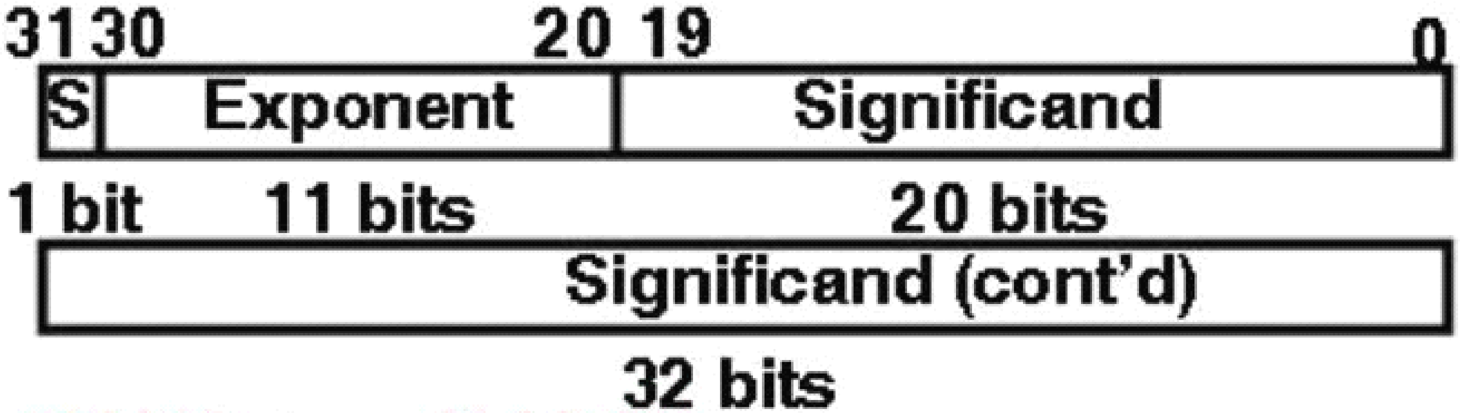
  - Underflow!下溢 ⇒ 负指数大于8位指数字段可表示范围



- 如何减少上下溢出的机会?

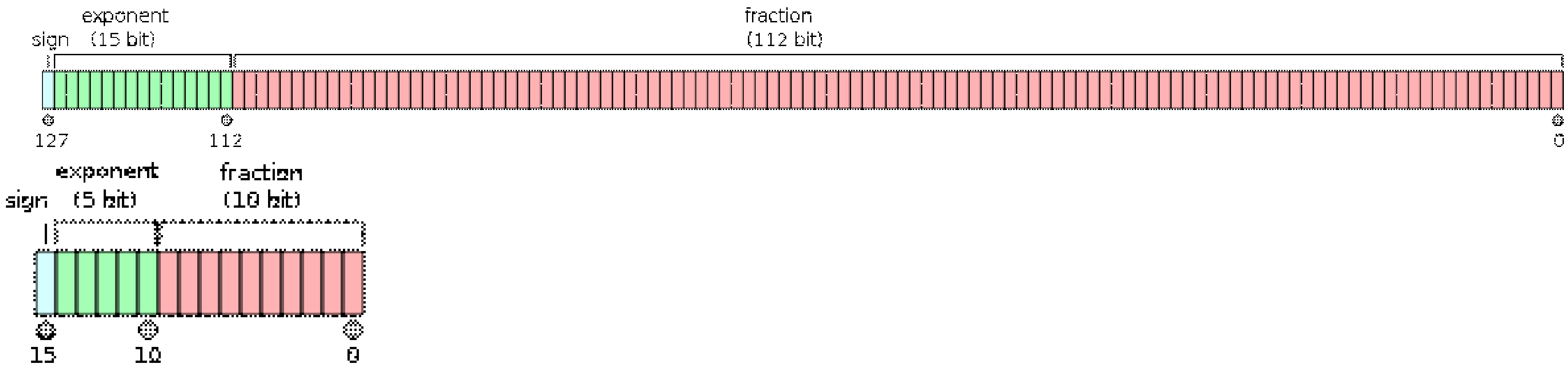
浮点数表示: 双精度

- 字的二倍长 (64位)



- 对精度 (vs. 单精度)
  - C变量，用double声明
  - 可表示的最小数为2.0 x 10^{-308}， 最大数为2.0 x 10^{308}
  - 但，根本的好处是有效位更多，从而精度更高

四精度/半精度



- http://en.wikipedia.org/wiki/Quad_precision
- http://en.wikipedia.org/wiki/Half_precision

IEEE754标准 - 01

单精度 (双精度类似): $+1.xxxxxxxxxx_{two} * 2^{yyyy}{}_{two}$

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

- Sign bit: 1 表示负数   0 表示正数
- 有效位Significand:
  - 想多一些有效位, 对规范化数, 前导1缺省
  - 单精度有1 +23位, 双精度有1 +52位
  - 对于规范化数恒有: 0 < Significand < 1
- 有问题吗?
  - 0怎么表示呢?
  - 指数的符号如何表示? 补码吗?
  - 符号位为什么不与有效位联在一起, 为什么不使用补码?

IEEE754标准 - 02

- 指数部分使用偏移表示Biased Notation，意即从指数中减去一常量得到真实的数
  - IEEE 754 对单精度数使用偏移值为127.
  - 即真实的指数为Exponent字段减去127得到
  - 对双精度其偏移为1023

- 小结 (单精度数):

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

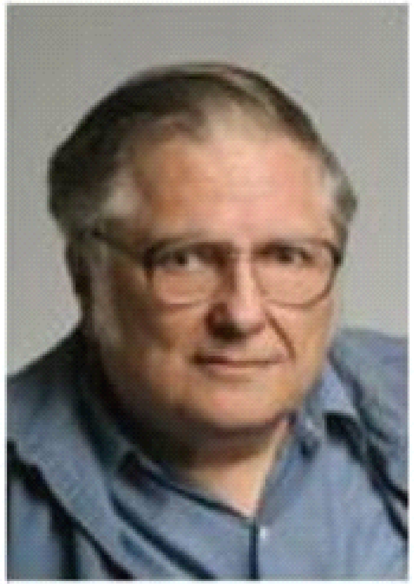
- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$
  - 双精度是一样的，只是指数偏移为1023 (半精度，四精度类似)

IEEE754标准 - 03

- IEEE 754使用“偏移指数biased exponent”表示的原因.
  - 与整数（序）兼容：就同样的位模式来说，大的浮点数对应于大的整数，从而即使没有浮点硬件，也可以使用整数运算，来比较两个浮点数，实现排序
  - 大指数字段表示大数.
    - 011000000ssssss 和 001000000ssssss谁大
  - 如果用补码就会出问题 (因为负数看来更大一些)
  - 后面，我们将看到浮点数的演化顺序和整数是完全一样的
    - 即., 二进制数从 00…00 到 11…11，而浮点数从 0 到 +MAX 到 -0 到 -MAX 到 0

浮点数标准 之 "父"

IEEE 754 二制制浮点数算术标准.

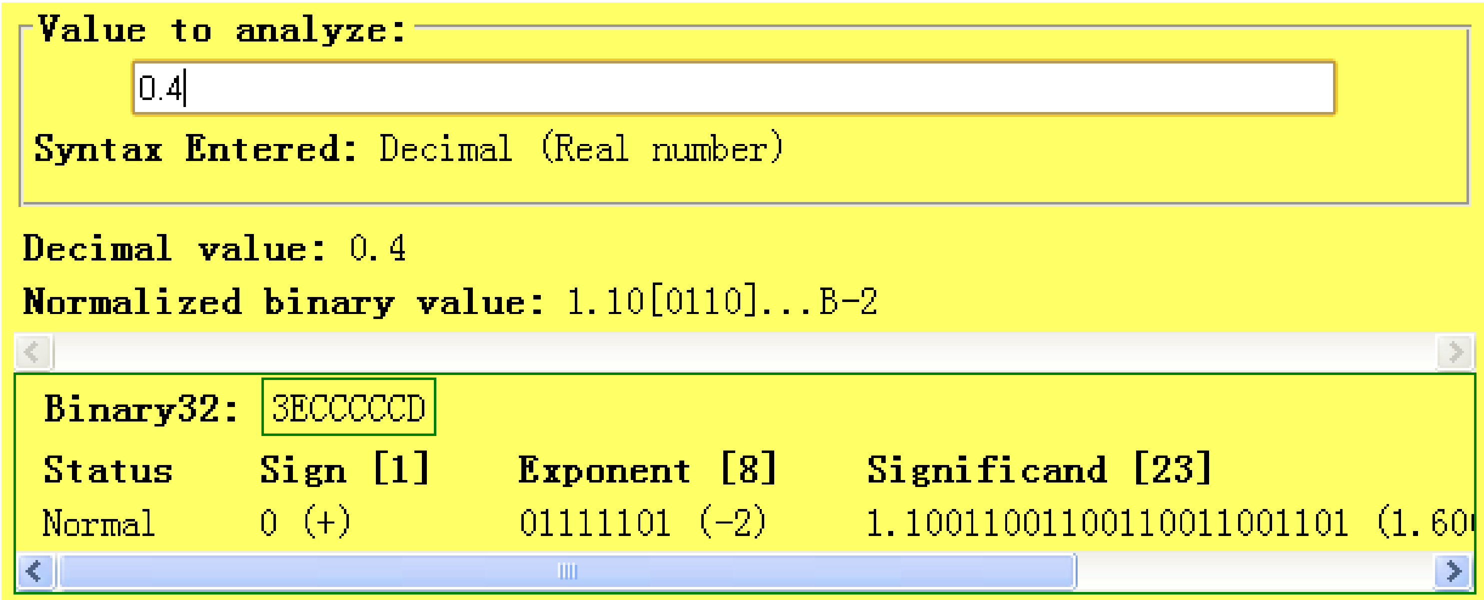


Prof. Kahan

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

浮点数分析

- http://babbage.cs.qc.cuny.edu/IEEE-754/
- **0.4** = 1.60000002384185791015625 * 2^(-2)
- **0.4** = 0.40000000596046447753906250



IEEE754: 表示0？

- 如何表示0?
  - **exponent全为0**
  - **Significand全0**
  - 0 00000000 00000000000000000000000
  - 这是$1.0*2^0=1$吗?
  - **What about sign? Both cases valid.**

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

IEEE754: 正负无穷

- 在浮点数中，除0应该得到 ± ∞, 而非溢出.
- Why?
  - 这样就能用∞做进一步的计算，如 X/0 > Y 可能是一个有意义的比较
  - Ask math majors
- IEEE 754 可表示 ± ∞
  - 最大的正指数保留用于表示 ∞
  - 有效位全为0

IEEE754: 特殊的数

- 目前所定义的 (单精度)

Exponent	Significand	Object
0	0	0
0	nonzero	???
1-254	anything	+/- fl. pt. #
255	0	+/- ∞
255	nonzero	???

- Professor Kahan had clever ideas; "Waste not, want not"（要舍得！）
  - 马上变Exp=0,255 且 Sig!=0

IEEE754: Not a Number

- 下面算式的结果是什么？
  `sqrt(-4.0)` or `0/0`?
  - 如果∞不出错，这些也不该错
  - 结果为 Not a Number (NaN)
  - Exponent = 255, Significand nonzero
- 这有什么用呢?
  - 希望NaN能有助于调试?
  - They contaminate: op(NaN, X) = NaN

非规格化数原因 - 01

- 问题：在0附近的浮点数，有一个宏沟
  - 浮点数可表示的最小正整数：

    $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - 浮点数可表示的次小的正整数：

    $b = 1.000\ldots\ldots1_2 * 2^{-126} = 2^{-126} + 2^{-149}$

  $a - 0 = 2^{-126}$ （$*2^{149} = 2^{23} = 8{,}000{,}000$）

  $b - a = 2^{-149}$ （$*2^{149} = 2^{0} = 1$）

问题出在规范化及缺省的前导1!

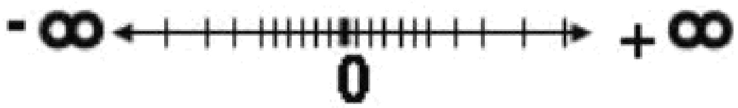
非规格化数原因 - 02

- 解决方案:
  - 还未使用 Exponent = 0, Significand 非
  - 非规范化数: 无 (缺省) 前导1, 指定expo
    = -126.
  - 可表示的最小正数:

    $a = 2^{-149} = 2^{-126} * 2^{-23}$ (0.000…000 001)
  - 可表示的次小正数:

    $b = 2^{-148} = 2^{-126} * 2^{-22}$ (0.000…000 010)

    $b-a = 2^{-148} - 2^{-149} = 2^{-149}$



非规格化数

- Reserve exponents, significands:

Exponent	Significand	Object
0	0	0
0	nonzero	Denorm
1-254	anything	+/- fl. pt. #
255	0	+/- ∞
255	nonzero	NaN

4种近似模式

为何不是五舍六入？

- 近似到 + ∞
  - 总是向上"up"近似：2.001 → 3, -2.001 → -2
- 近似到 - ∞
  - 总是向下 "down"近似：1.999 → 1, -1.999 → -2
- 截断
  - 去掉最后一位 (近似到0)
- 无偏的 (缺省模式). 中值？近似到偶数
  - 几乎总和通常近似一致：2.4 → 2, 2.6 → 3, 2.5 → 2, 3.5 → 4
  - 和在学校学的近似一样 (最近的整数)
  - 例外情况，值在边界上，此时近似到最近的偶数
  - 保证计算的公正性
  - 这样，一半情况会向下近似，一半情况会向上近似，从而平衡非精确性。

浮点数打印

```
float e = 2.718281828
float zero = 0.0;

printf("%.0f\n",e);  // 3
printf("%.1f\n",e);  // 2.7
printf("%.2f\n",e);  // 2.72
printf("%.6f\n",e);  // 2.718282
printf("%f\n",e);    // 2.718282
printf("%.7f\n",e);  // 2.7182818

printf("%f\n", (+1.0/zero));  // 1.#INF00
printf("%f\n", (-1.0/zero));  // -1.#INF00
printf("%f\n", (-0.0/zero));  // -1.#IND00
```

浮点数加法

加法不满足结合律!
无穷大+(N个无穷小+)

- 比整数更复杂
- 不能仅加有效位significands
- 如何做?
  - 匹配指数，使其一致（去规范化）
  - 有效位相加
  - 指数不变
  - 规范化 (可能需要改变指数)
- 注意: 如果符号不同, 则做减法.

浮点数比较

- 浮点数当作有符号整数时, 非**NaN**值可正确排序.
- IEEE754规定, 有一个是NaN比较时, 不可排序.
- 假设 -0.0 和 +0.0 不等(如相等代码会复杂一些).

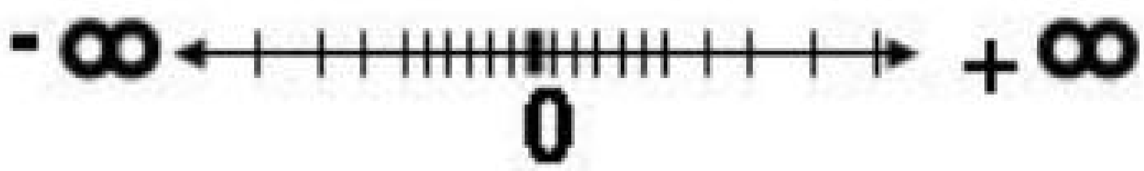
$$a \overset{f}{=} b \approx (a = b)$$

$$a \overset{f}{<} b \approx (a \geq 0 \,\&\, a < b) \mid (a < 0 \,\&\, a \overset{u}{>} b)$$

$$a \overset{f}{\leq} b \approx (a \geq 0 \,\&\, a \leq b) \mid (a < 0 \,\&\, a \overset{u}{\geq} b)$$

浮点数分布

- 定点数在数轴上分布均匀
- 浮点数分布不均匀
  - 越靠近原点越密
  - 同一阶码分布均匀



if(x+1 == x): x = ?

float的有效位23bit, 加上前导1共**24**个有效位.
当超过24个有效位时, 后面数会被丢掉.

因此, 当**x**大于**2^(23+1)**时, 加**1.0**会被丢失.
2^24对应整数 **16777216** .

1677721**7** ==> 4B800001
1677721**8** ==> 4B800001

满足条件的x有很多: **16777217, ...**

IEEE754资源

- http://babbage.cs.qc.cuny.edu/IEEE-754/
- http://en.wikipedia.org/wiki/IEEE_754-2008
- http://en.wikipedia.org/wiki/Single_precision
- http://www.h-schmidt.net/FloatApplet/IEEE754.html
- http://babbage.cs.qc.edu/IEEE-754/
- http://www.scs.stanford.edu/histar/src/pkg/uclibc/include/ieee754.h

Thanks!