信息的表示和处理(4)

谢旻晖

2025.9

Floating Point

Outline

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in *C*
- Summary

Fractional binary numbers

- What is 1011.101_2?

填空题 1分

设置

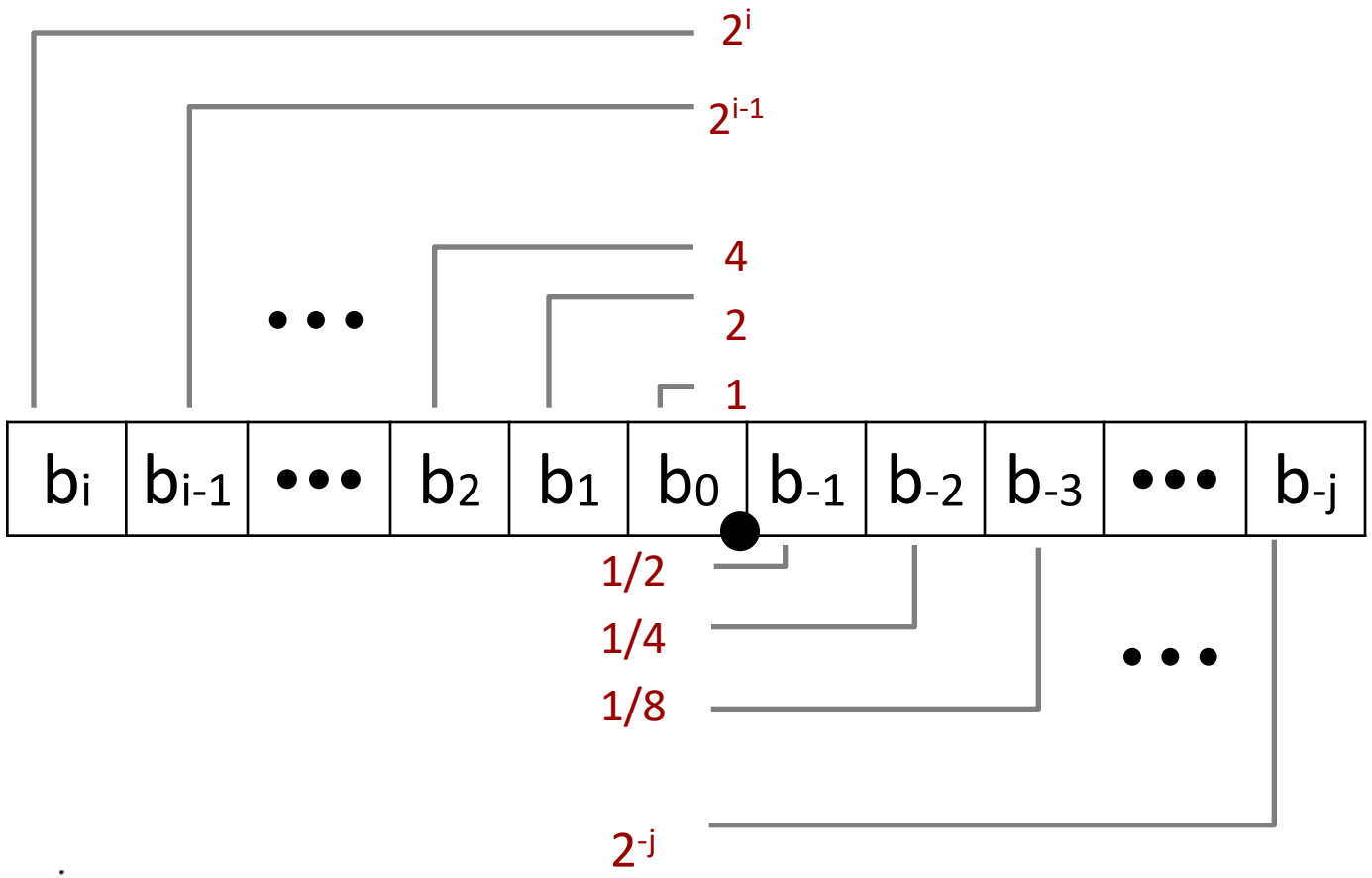
定点数计算

What is 1011.101_2?

[填空1]

作答

Fractional Binary Numbers



$$\sum_{k=-j}^{i} b_k \times 2^k$$

Fractional Binary Numbers: Examples

- **Value** **Representation**
  - **5 3/4** 101.11_2
  - **2 7/8** 10.111_2
  - **1 7/16** 1.0111_2

- 观察
  - 通过左移小数点可以除以2（无符号数）
  - 通过右移小数点可以乘以2
  - 数字$0.111111\ldots_2$刚刚比1.0小一点
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$

Representable Numbers

- Limitation #1
  - 只能精确表示$x/2^k$形式的数字（x为整数）
    - 其他数字会是无限小数（循环小数）形式
  - Value Representation
    - 1/3 0.0101010101[01]…2
    - 1/5 0.001100110011[0011]…2
    - 1/10 0.0001100110011[0011]…2

- Limitation #2
  - 位数有效（w bits），表达的数字大小范围有限
    - Limited range of numbers (very small values? very large?)

课堂练习

爱国者导弹系统中有一个内置的时钟，其实现类似一个计数器，每0.1秒就加1。为了以秒为单位来确定时间，程序将用一个24位的近似于1/10的二进制小数值来乘以这个计数器的值。特别地，1/10的二进制表达式是一个无穷序列$0.000110011[0011]\cdots_2$，其中，方括号里的部分是无限循环的。程序用值x近似地表示0.1，x只考虑这个序列的二进制小数点右边的前23位：x = 0.00011001100110011001100。（参考练习题2.51，里面有关于如何更精确地近似表示0.1的讨论。）

A. 0.1 − x 的二进制表示是什么？

B. 0.1 − x 的近似的十进制值是多少？

C. 当系统初始启动时，时钟从0开始，并且一直保持计数。在这个例子中，系统已经运行了大约100个小时。程序计算出的时间和实际的时间之差为多少？

D. 系统根据一枚来袭导弹的速率和它最后被雷达侦测到的时间，来预测它将在哪里出现。假定飞毛腿导弹的速率大约是2000米每秒，对它的预测偏差了多少？

通过一次读取时钟得到的绝对时间中的一个轻微错误，通常不会影响跟踪的计算。相反，它应该依赖于两次连续的读取之间的相对时间。问题是爱国者导弹的软件已经升级，可以使用更精确的函数来读取时间，但不是所有的函数调用都用新的代码替换。结果就是，跟踪软件一次读取用的是精确的时间，而另一次读取用的是不精确的时间[100]。

答案

A. 0.1的二进制表示是 0.000110011[0011],程序中的x是 0.00011001100110011001100。将两者对齐之后互相减一下：

```
0.00011001100110011001100110011001100110011.....
0.00011001100110011001100
```

得到结果是 0.0000000000000000000000[1100]

B. 近似的10进制值是2的24次幂分之一加上2的25次幂分之一。

C. 100个小时之后，系统计数的次数运行了100*3600*10 = 3600000次。误差大概是0.343秒。

D. 2000*343 = 686米

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in C
- Summary

定点数的问题

- 无法高效表示非常大的数字，例如$5 * 2^{100}$需要101后面跟100个0的位模式

- 能否仅用<5, 100> 两个数代表形如 $x * 2^{y}$的数

Floating Point Representation

- 浮点表示法
  - 举例：$(-1101.0101)_2$,m=9, n=3
  - $-0.11010101*2^4$

S	Es	E	M
1位	1位	3位	9位
1	0	100	110101010

  - $-0.011010101*2^5$

S	Es	E	M
1位	1位	3位	9位
1	0	101	011010101

IEEE Floating Point

- IEEE Standard 754
  - 在1985年建立，作为浮点数运算的统一标准
    - 在此之前，有很多不同（古怪）的设计
  - 所有主流CPU都支持
- William (Velvel) Morton Kahan 威廉·卡亨
  - 1933 -
  - UC Berkeley
  - 1989年图灵奖，因为数值计算方面的贡献
  - Kahan是浮点运算IEE标准IEEE 754，IEEE 854的主要设计师

IEEE Floating Point Representation

- Numerical Form:

$$(-1)^s \; M \;\; 2^E$$

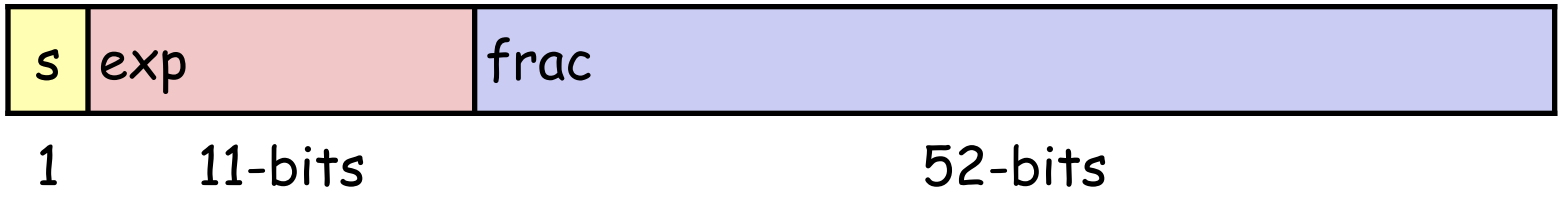
  - Sign bit s determines whether number is negative or positive
  - 尾数Significand M normally a fractional value in range [1.0,2.0).
  - 阶码Exponent E weights value by power of two

- Encoding
  - MSB s is sign bit s
  - exp field encodes E (but **is not equal** to E)
  - frac field encodes M (but **is not equal** to M)

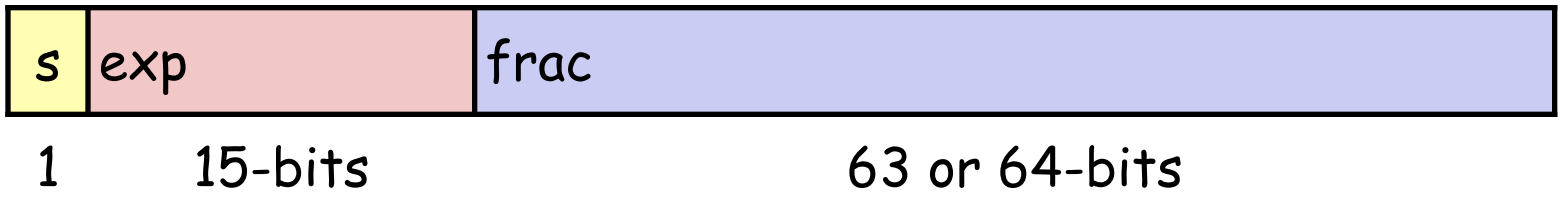
Precision options

- Single precision: 32 bits (float)

- Double precision: 64 bits (double)



- Extended precision: 80 bits (Intel only, long double)



浮点数编码

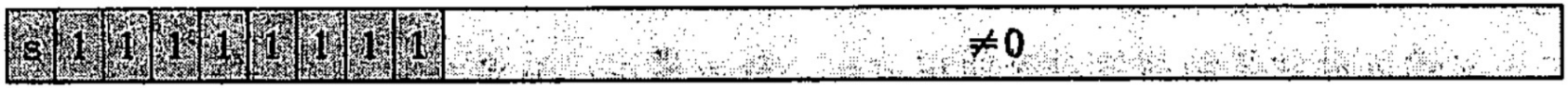
- 三种情况

1. 规格化的

2. 非规格化的

3a. 无穷大

3b. NaN



"Normalized" Values(规格化)

$v = (-1)^s M 2^E$

- When: exp ≠ 000…0 and exp ≠ 111…1
- E coded as a biased value: E = Exp – Bias (移码/偏置)
  - Exp: unsigned value of exp field
  - Bias = $2^{k-1} - 1$, where k is number of exponent bits
    - Single precision: 127 (Exp: 1…254, E: -126…127)
    - Double precision: 1023 (Exp: 1…2046, E: -1022…1023)
- M coded with implied leading 1: M = $1.xxx…x_2$
  - xxx…x: bits of frac field
  - Minimum when frac=000…0 (M = 1.0)
  - Maximum when frac=111…1 (M = 2.0 – ε)
  - Get extra leading bit for "free"

Normalized Encoding Example

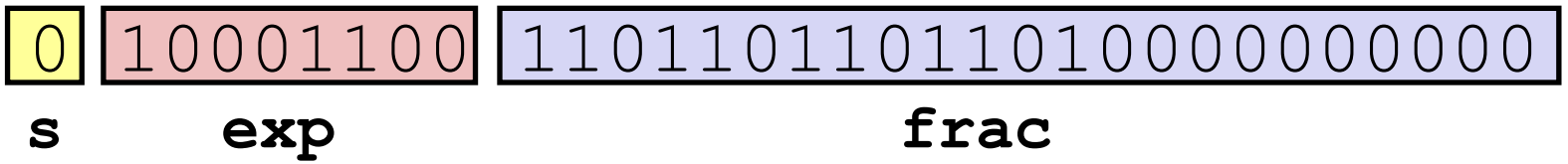
$v = (-1)^s M 2^E$
$E = Exp - Bias$

- Value: `float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$

- Significand
  $M = 1.\underline{1101101101101}_2$
  $\mathbf{frac} = \underline{1101101101101}0000000000_2$

- Exponent
  $E = 13$
  $Bias = 127$
  $Exp = 140 = \mathbf{10001100}_2$

- Result:



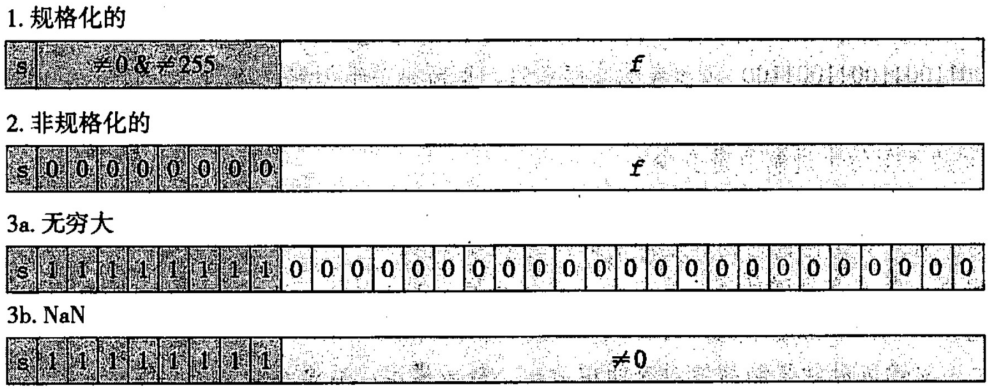
Denormalized Values (非规格化)

$v = (-1)^s M 2^E$
$E = 1 - Bias$

- Condition: exp = 000…0

- 阶码Exponent value: E = 1 – Bias (instead of E = 0 – Bias)
  - 为了平滑过度到规格化浮点数（$1.xxx * 2^{-126}$）
  - 非规格化：$0.xxx * 2^{-126}$
- M没有隐藏的1，方便表示0及接近0的数字: M = $0.xxx…x_2$
- Cases
  - **exp** = 000…0, **frac** = 000…0
    - 表示0
    - 但是根据S为0或1，可以表示 +0 and -0
  - **exp** = 000…0, **frac** ≠ 000…0
    - 用于表示接近于 0.0的数字

Special Values

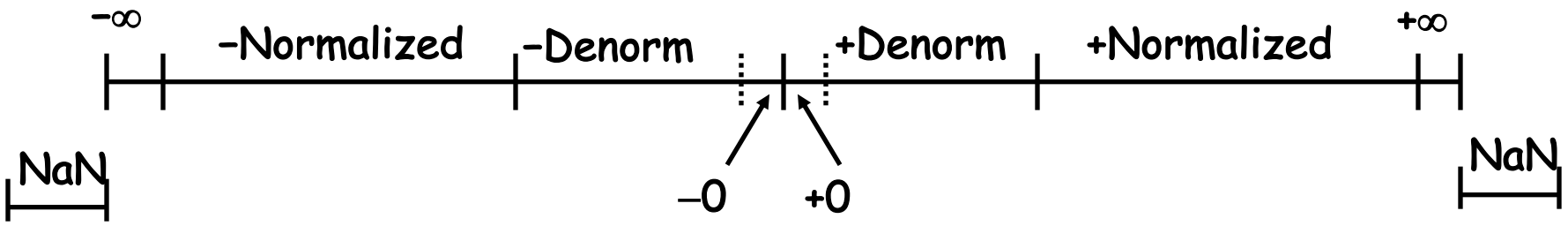


- Condition: **exp = 111…1**

- Case: **exp = 111…1, frac = 000…0**
  - Represents value ∞
  - 用来表示overflow
  - 包括+∞和−∞
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞

- Case: **exp = 111…1, frac ≠ 000…0**
  - Not-a-Number (NaN)
  - 表示无法得到一个数值（出错/未定义）
  - E.g., 0/0, sqrt(-1), ∞ − ∞, ∞ × 0

Visualization: Floating Point Encodings

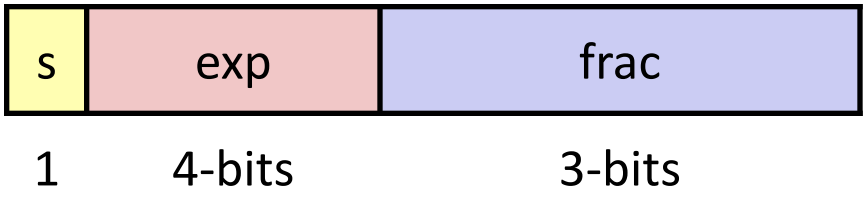


有限范围内的一些采样点，与实数完全不同

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in C
- Summary

Tiny Floating Point Example



- 8-bit Floating Point Representation
  - 符号为S是最左边的1位
  - 下面4位是阶码，偏置bias=2^3-1=7
  - 最后三位是尾数

- 其他规则采用IEEE 754标准
  - 规格化、非规划化
  - 0, NaN, 无穷大的表示

Dynamic Range (Positive Only)

$v = (-1)^s M 2^E$

n: E = Exp – Bias

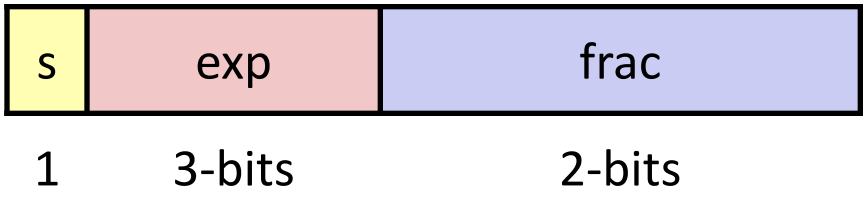
d: E = 1 – Bias

Bias = 2^3-1 = 7

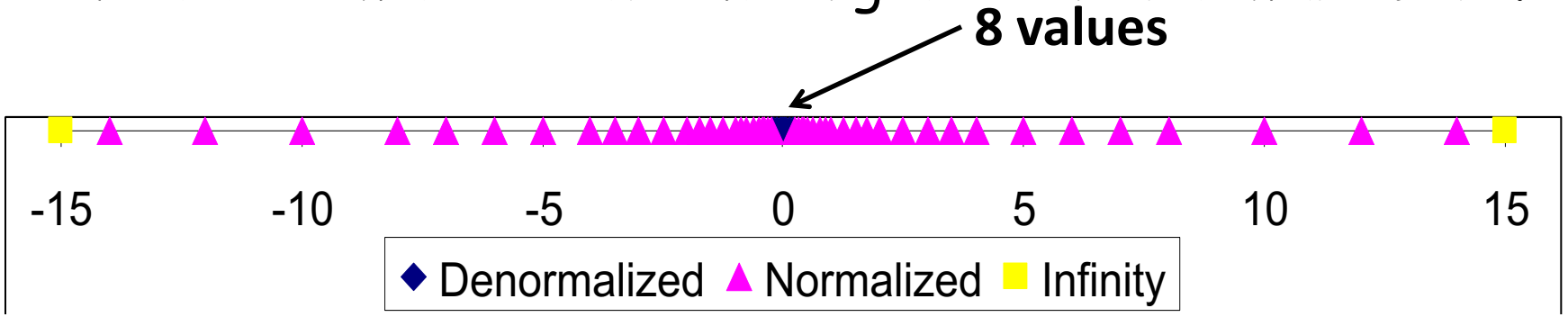
	s	exp (k位)	frac (m位)	E	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	closest to zero $2^{-m}*2$
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	largest denorm $(1-2^{-m}$
Normalized numbers	0	0001	000	-6	8/8*1/64 = 8/512	smallest norm $1*2^{2-}$
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	closest to 1 below
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	closest to 1 above
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	largest norm $(2-2^{-m}$
	0	1111	000	n/a	inf	

Distribution of Values

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is $2^{3-1}-1 = 3$

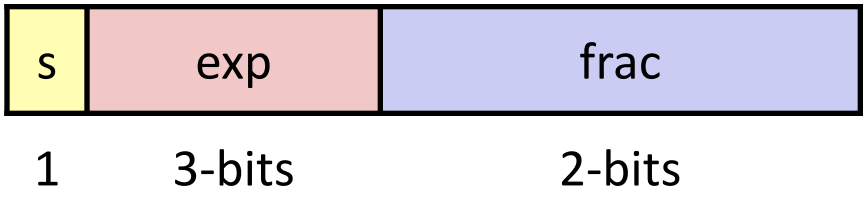


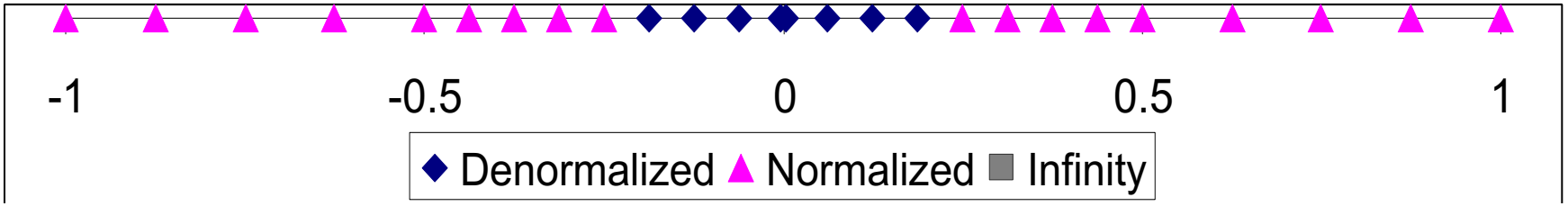
- Notice how the distribution gets denser toward zero.



Distribution of Values (close-up view)

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3



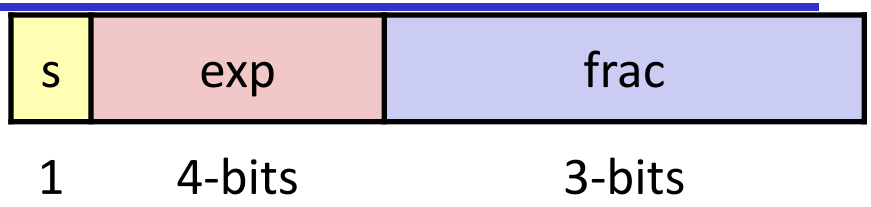


Special Properties of the IEEE Encoding

- 浮点数0和整型0在二进制形式上一样
  - 所有bits都是0（+0）

- 几乎总可以使用unsigned integer的比较大小方式
  - 必须首先比较符号位
  - 必须考虑-0 = 0
  - NaNs的问题
    - 比任何其他值都大
    - NaNs有多个值
  - 其他情况OK
    - 非规格化< 规格化
    - 规格化 < 无穷大

Creating Floating Point Number



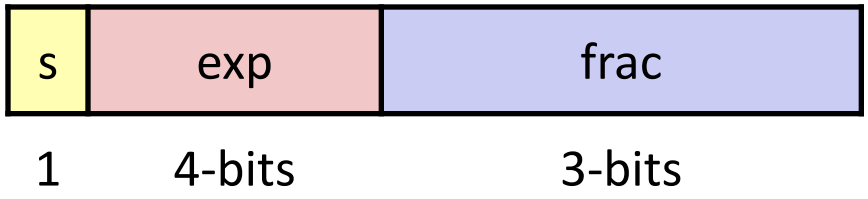
- Steps
  - 规格化，首位为 1
  - 尾数部分进行round
  - 后规格化(Postnormalize)处理round带来的问题

- Case Study
  - Convert 8-bit unsigned numbers to tiny floating point format

Example Numbers

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010
63	00111111

Normalize



- Requirement
  - Set binary point so that numbers of form 1.xxxxx
  - Adjust all to have leading one
    - Decrement exponent as shift left

Value	Binary	Fraction	Exponent
128	10000000	1.0000000	7
15	00001101	1.1010000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

Rounding

- 向偶数舍入

Value	Fraction	Rounded
128	1.0000000	1.000
15	1.1010000	1.101
17	1.0001000	1.000
19	1.0011000	1.010
138	1.0001010	1.001
63	1.1111100	10.000

方 式	1.40	1.60	1.50	2.50	−1.50
向偶数舍入	1	2	2	2	−2
向零舍入	1	1	1	2	−1
向下舍入	1	1	1	2	−2
向上舍入	2	2	2	3	−1

Rounding

- Rounding Modes (illustrate with $ rounding)

	$1.40	$1.60	$1.50	$2.50	-$1.50
- 向0舍入	$1	$1	$1	$2	-$1
- 向下舍入(-∞)	$1	$1	$1	$2	-$2
- 向上舍入(+∞)	$2	$2	$2	$3	-$1
- 就近舍入(default)	$1	$2	$2	$2	-$2

  - 向偶数舍入（距离两边一样的中间结果，向偶数舍入）

Closer Look at Round-To-Even

- 默认的舍入模式
  - 如果不用汇编语言，很难改为其他round模式
  - 其他模式都是静态偏移
    - round的方向是确定的
  - 所以round-to-even的好处是数字有两个round方向，可以避免/减小统计偏差

- 应用在10进制数字上的例子
  - E.g., round to nearest hundredth

7.8949999	7.89	(Less than half way)
7.8950001	7.90	(Greater than half way)
7.8950000	7.90	(Half way—round up)
7.8850000	7.88	(Half way—round down)

Rounding Binary Numbers

- 二进制小数
  - "Even" 当最小的数字是0时
  - "Half way" 当bits正好在中间位置时 = $100\ldots_2$

- Examples
  - Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

Postnormalize

- Issue
  - 舍入时可能导致 overflow
  - 可能需要右移M，增加E

Value	Rounded	Exp	Adjusted	Result
128	1.000	7		128
15	1.101	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000/6	64

Interesting Numbers {single,double}

Description	*exp*	*frac*	*Numeric Value*
• Zero	00…00	00…00	0.0
• Smallest Pos. Denorm.	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
– Single ≈ 1.4×10^{-45}			
– Double ≈ 4.9×10^{-324}			
• Largest Denormalized	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
– Single ≈ 1.18×10^{-38}			
– Double ≈ 2.2×10^{-308}			
• Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
– Just larger than largest denormalized			
• One	01…11	00…00	1.0
• Largest Normalized	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
– Single ≈ 3.4×10^{38}			
– Double ≈ 1.8×10^{308}			

宇宙所有原子的数量10^{80}

课堂练习

- 假设浮点数共16位，其中阶码6位，采用类IEEE 754标准
  - 1）求浮点数20.625的二进制形式
  - 2）求浮点数能够表示的规格化的最大负数和最小负数，以及非规格化的最大负数和最小负数（二进制+真值）

主观题 10分

$v = (-1)^s M 2^E$
n: E = Exp – Bias
d: E = 1 – Bias
Bias = 2^3-1 = 7

设置

假设浮点数共16位，其中阶码6位，采用类IEEE 754标准

1）求浮点数20.625的二进制形式

2）求浮点数能够表示的规格化的最大负数和最小负数，以及非规格化的最大负数和最小负数（二进制+真值）

作答

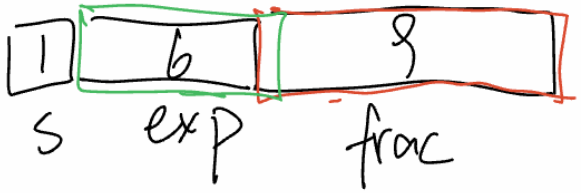
$v = (-1)^s M 2^E$

n: E = Exp – Bias

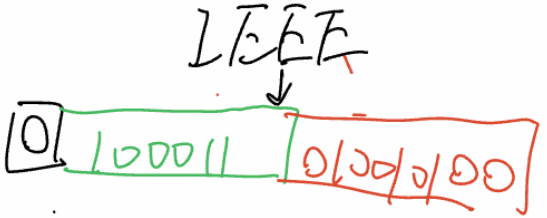
d: E = 1 – Bias

Bias = 2^5-1 = 31

16 bit float. 20.625



$Bias = 2^{k-1} - 1 = 2^{6-1} - 1 = 31$



$20.625_{10} = 10100.101_2$

1.0100101×2^{4} (E)

frac

$exp = E + Bias = 4 + 31 = 35 = 100011_2$

课堂练习（答案）

1.0100101 * 2^4，阶码4+31= 35

Bias = 2^{6-1}-1 = 31

	S	E	M	Value
非规格化	1	000 000	000 000 001	-2^(-9)*2^(-30)
	1	000 000	111 111 111	-(1-2^(-9))*2^(-30)
规格化	1	000 001	000 000 000	-1*2^(-30)
	1	111 110	111 111 111	-(2-2^(-9))*2^31
特殊情况	1	111 111	0 != 0	Infinite NaN

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in C
- Summary

Floating Point Operations: Basic Idea

- $x +_f y = Round(x + y)$

- $x \times_f y = Round(x \times y)$

- Basic idea
  - 首先计算精确结果
  - 然后去适配精度，进行溢出/舍入
    - 当阶过大时，可能溢出，变为+/- ∞
    - 尾数可能需要舍入

Floating Point Multiplication

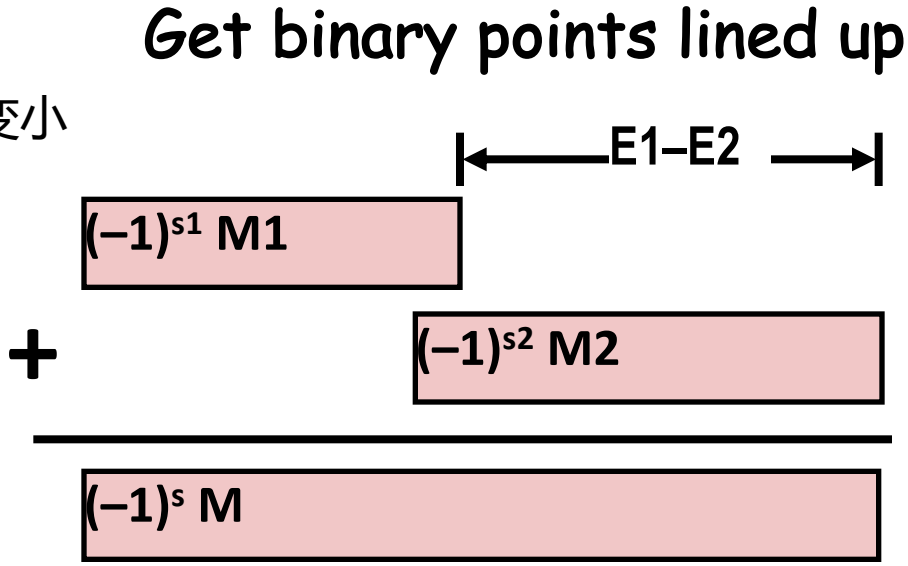
- $(-1)^{s1}$ M1 2^{E1} x $(-1)^{s2}$ M2 2^{E2}
- Exact Result: $(-1)^{s}$ M 2^{E}
  - Sign s: s1 ^ s2
  - Significand M: M1 x M2
  - Exponent E: E1 + E2

- 结果修正
  - 如果M ≥ 2, M右移，对应增大E
  - 如果E越界，就overflow
  - 最后M的值再进行舍入

- 实现
  - 复杂度最高的部分是尾数的乘法

Floating Point Addition

- $(-1)^{s1}$ M1 2^{E1} + $(-1)^{s2}$ M2 2^{E2}
  - 假设E1 > E2，首先对阶，E2->E1, M2变小
- 准确结果: $(-1)^{s}$ M 2^{E}
  - Sign s, significand M:
    - Result of signed align & add
  - Exponent E: E1



- 修正：
  - 如果M ≥ 2, 右移M, 增加E
  - 如果M < 1, 将M 左移 k 位到合法区间, E 减小 k （可能规格化，可能非规格化）
  - 如果E超过边界，则overflow
  - 如果M位数过长，则进行舍入

Mathematical Properties of FP Add

- 是否符合阿贝尔群的特征
  - 加法的封闭性? **Yes**
    - 但是有可能产生无穷或NaN
  - 交换律? **Yes**
  - 结合律? **No**
    - Overflow and inexactness of rounding
    - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
  - 0 是加法单位元? **Yes**
  - 每个元素都有加法逆元? **Almost**
    - Yes, 除了infinities & NaNs
- 单调性
  - $a \geq b \Rightarrow a+c \geq b+c$? **Almost**
    - 除了infinities & NaNs

Mathematical Properties of FP Add

- FP加法不具有结合律
  - 对编译器有很大影响
    - 例如x = a + b + c; y = b + c + d;
    - 编译器试图优化，减少一次运算
      - t = b + c;
      - x = a + t;
      - y = t + d
    - 这样可能产生不同的值，编译器一般会做保守的选择，避免对程序功能产生影响

Mathematical Properties of FP Mult

- 属性
  - 乘法是否封闭？ **Yes**
    - 但可能产生 infinity or NaN
  - 交换律? **Yes**
  - 结合率? **No**
    - 可能导致溢出，或者由于舍入带来的不精确
    - Possibility of overflow, inexactness of rounding
    - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
  - 1是乘法的单位元？ **Yes**
  - 乘法对加法的分配率？ **No**
    - 可能导致溢出，或者由于舍入带来的不精确
    - `1e20*(1e20-1e20)= 0.0, 1e20*1e20 – 1e20*1e20 = NaN`
- 单调性 **Almost**
  - $a \geq b \;\&\; c \geq 0 \Rightarrow a * c \geq b * c$?
    - 除了infinities & NaNs

Ariane 5: 浮点溢出的高昂代价

- 1996.6.4，Ariane 5火箭发射37秒后爆炸（损失5亿美金）

Ariane 5: 浮点溢出的高昂代价

- 原因：导航系统向控制引擎喷嘴的计算机发送了一个无效数据，没有发送飞行控制信息，而是发送了一个诊断位模式，表明将64位浮点数转为16位有符号位时溢出；
- Ariene 4火箭的速度不会超过16位浮点数；
- Ariene 5速度比Ariene 4高出5倍，但直接重用了Ariene 4的代码；
- 将大的浮点数转换为整数是一种常见的程序错误来源

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- Floating point in *C*
- Summary

Floating Point in C

- C Guarantees Two Levels
  - **float** single precision
  - **double** double precision
- Conversions/Casting
  - **int**, **float**和**double**之间的转换会改变bit值
  - **double/float → int**
    - 去掉小数部分
    - 向0舍入
    - 当超过范围或NaN时没有定义，一般设为Tmin（可能过大整数得到负数Tmin）
  - **int → double**
    - 精确的转换，int数值 < 53 bit
  - **int → float**
    - 按照舍入模式来进行舍入

Floating Point Comparison

- 不能用==, !=判定浮点数相等或不等
  - 0.1这样10进展整齐的数在二进制下可能无法精确表示
  - 要进行近似；而且数字在处理几次之后，会得到一些误差（舍入导致）
  - 所以，可能两个0.1的二进制的表达不完全一样
- 采用fabs(f1-f2) <= precision
  - precision为自己预设的精度，如1e-6
- 但以上precision是绝对精度，如果浮点数非常大，可能用1e-6就不合适了
  - 应该用相对精度

Floating Point Comparison

- 相对误差和绝对误差结合的方式
  - bool IsEqual(float a, float b, float absError, float relError) {
  - if (a==b) return true;
  - if (fabs(a-b)<absError) return true;
  - if (fabs(a)<fabs(b)) return (fabs((a-b)/a)<relError) ? true : false;
  - return (fabs((a-b)/b<relError) ? true : false;
  - }

Floating Point Puzzles（课堂练习）

- 对于下面的每一个C语言表达式：
  - 或者证明各种情况都为真
  - 或者解释为什么不为真

```
int x = …;
float f = …;
double d = …;
```

假设d和f都不是NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (double)(float) d`
- `f == -(-f);`

- `2/3 == 2/3.0`
- `d > f  ⇒  -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Summary

- IEEE浮点数标准有清晰的数学属性
- 浮点数表示数字的形式是 $M \times 2^E$
- 和实数代数不一样
  - 违反结合率和分配率
  - 使compiler和一些严格的计算型应用程序很难做

课堂练习

- 分配给你一个任务，编写一个C函数用来计算2^x的浮点表示。你意识到完成这个任务的最好方法是直接创建结果的IEEE单精度表示。
- 当x太小时返回0.0；当x太大时返回+∞。
- 填写下面的代码空白，以计算出正确的结果。
- 假设函数u2f返回的浮点值与它的无符号参数有相同的位表示。

```
float fpwr2(int x) {
  unsigned exp, frac, u;
  if (x < _______) {
    exp = _______;
    frac = _______;
  } else if (x < _______) {
    exp = _______;
    frac = _______;
  } else if (x < _______) {
    exp = _______;
    frac = _______;
  }else {
    exp = _______;
    frac = _______;
  }
  u = exp<<23 | frac;
  return u2f(u);
}
```

练习答案

- 分配给你一个任务，编写一个C函数用来计算2^x的浮点表示。你意识到完成这个任务的最好方法是直接创建结果的IEEE单精度表示。
- 当x太小时返回0.0；当x太大时返回+∞。
- 填写下面的代码空白，以计算出正确的结果。
- 假设函数u2f返回的浮点值与它的无符号参数有相同的位表示。

```
float fpwr2(int x) {
  unsigned exp, frac, u;
  if (x < __-149_____) { //太小，返回0
    exp = __0_____;
    frac = __0_____;
  } else if (x < _-126_) {  //非规格化
    exp = __0_____;
    frac = __1<<(x+149)_____;
  } else if (x < _128_) {  //规格化
    exp = __x+127_____;
    frac = __0_____;
  }else {
    exp = ___255___;  // 太大，返回+∞
    frac = ___0___;
  }
  u = exp<<23 | frac;
  return u2f(u);
}
```