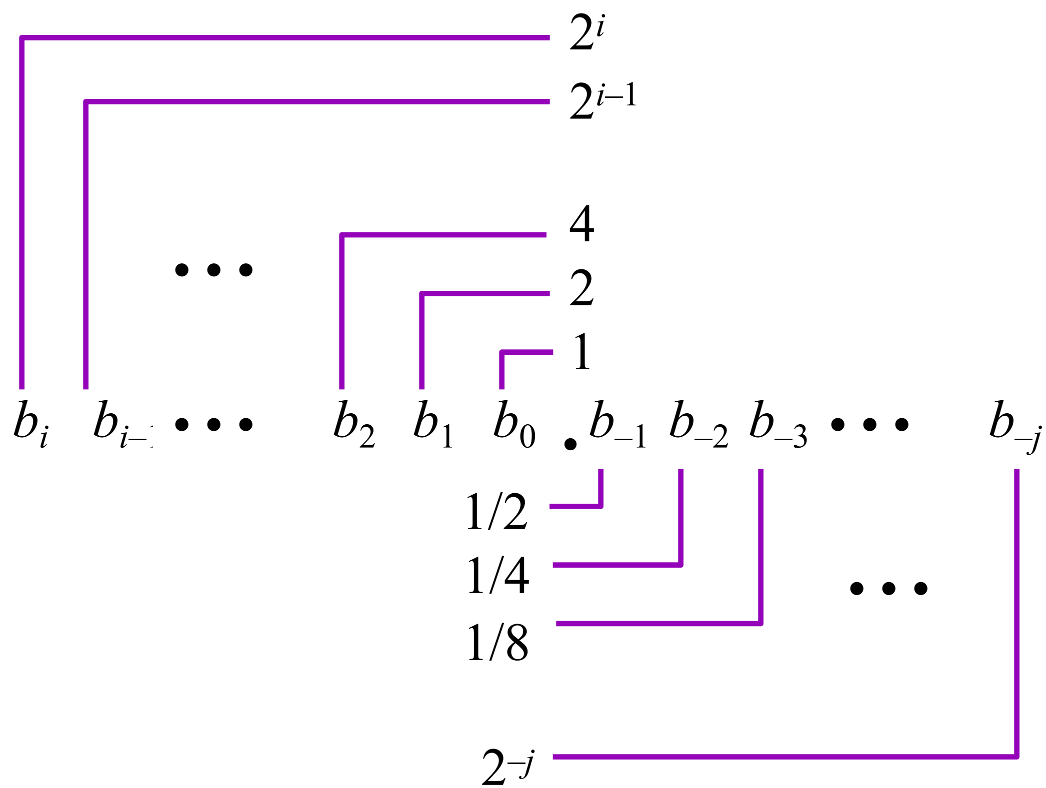
汇编语言程序设计

浮 点 数

- IEEE的浮点数标准
- Rounding（舍入）
- C语言中的浮点数

IEEE浮点数标准

- IEEE的754标准
  - 1985年建立
- 二进制表示方式



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

浮点数示例

- 值 二进制表示

值	二进制表示
5-3/4	101.11_2
2-7/8	10.111_2
63/64	0.111111_2

- 局限性
  - 只能精确地表示$X/2^k$这类形式的数据（k为整数）

- 值 二进制表示

值	二进制表示
1/3	$0.0101010101[01]\cdots_2$
1/5	$0.001100110011[0011]\cdots_2$
[redacted]	[redacted] $0001100110011[0011]\cdots_2$

计算机中的浮点数二进制表示

- 数字形式
  - $(-1)^s\ M\ \ 2^E$
    - 符号：s
    - 尾数：M， 是一个位于区间[1.0,2.0)内的小数
    - 阶码：E
- 编码

s	exp	frac

  - exp域：E
  - frac域：M

单精度浮点数：exp域宽度为8 bits，frac域宽度为23 bits

总共32 bits。

双精度浮点数：exp域宽度为11 bits，frac域宽度为52bits

总共64 bits。

扩展精度浮点数： exp域宽度为15 bits，frac域宽度为63bits

[redacted] *（1 bit wasted）*

- 浮点数的类型
  - 规格化浮点数
  - 非规格化浮点数
  - 一些特殊值

规格化浮点数 （ Normalized ）

- 满足条件
  - exp ≠ 000…0 且 exp ≠ 111…1

- 真实的阶码值需要减去一个偏置（biased）量

  E = Exp – Bias
  - Exp : exp域所表示的无符号数值
  - Bias的取值
    - 单精度数：127（Exp：1…254，E：-126…127）
    - 双精度数：1023（Exp：1…2046，E：-1022…1023）
      - Bias = $2^{e-1} - 1$，e = exp域的位数

- frac域的第一位隐含为1

  M = $1.xxx\ldots x_2$
  - 因此，第一位的“1”可以省去，xxx…x：bits of frac
  - Minimum when 000…0（M = 1.0）
  - [redacted]1…1（M = 2.0 – ε）

规格化浮点数示例

Float F = 15213.0

- $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$

尾数

$M = 1.\underline{1101101101101}_2$

$\text{frac} = \underline{1101101101101}0000000000_2$

阶码

$E = 13$

$Bias = 127$

$Exp = 140 = 10001100_2$

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0110	0110	1101	1011	0100	0000	0000
140:	100	0110	0					
15213:			1110	1101	1011	01		

非规格化浮点数（ Denormalized ）

- 满足条件
  - exp = 000…0
- 其它域的取值
  - $E = -Bias + 1$
  - $M = 0.xxx\cdots x_2$
    - xxx…x: bits of frac
- 具体示例
  - exp = 000…0, frac = 000…0
    - 表示0
    - 注意有 +0 与 -0
  - exp = 000…0, frac ≠ 000…0
    - 表示“非常接近”于0的浮点数
    - 会逐步丧失精度
      - [redacted]dual underflow”

一些特殊值

- 满足条件
  - exp = 111…1

- 具体示例
  - exp = 111…1， frac = 000…0
    - 表示无穷
      - 可用于表示数值的溢出
    - 有“正无穷”与“负无穷”
    - E.g.， 1.0/0.0 = +∞， -1.0/0.0 =-∞

  - exp = 111…1， frac ≠ 000…0
    - Not-a-Number (NaN)
    - E.g.， sqrt(-1)， ∞ – ∞， ∞ * 0

一种“小”浮点数实例

- 8位浮点数表示
  - exp域宽度为4 bits, frac域宽度为3bits
- 其他规则符合IEEE 754规范
  - 规格化 / 非规格化
  - 表示0, NaN与无穷

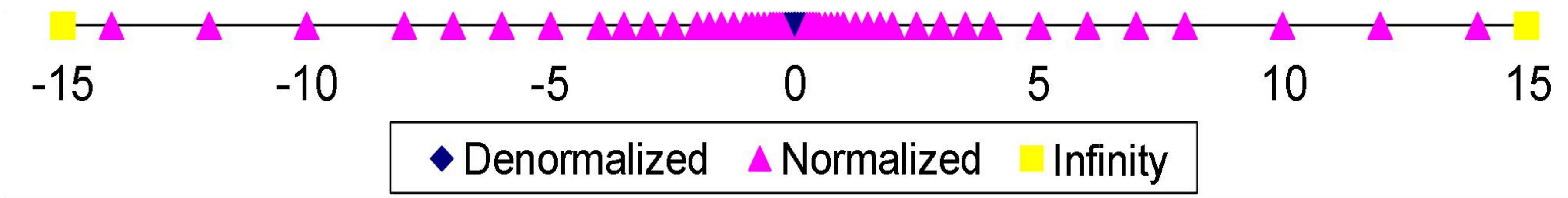
7	6 3	2 0
s	exp	frac

Exp	**exp**	E	2^E	
0	0000	-6	1/64	(非规格化数)
1	0001	-6	1/64	
2	0010	-5	1/32	
3	0011	-4	1/16	
4	0100	-3	1/8	
5	0101	-2	1/4	
6	0110	-1	1/2	
7	0111	0	1	
8	1000	+1	2	
9	1001	+2	4	
10	1010	+3	8	
11	1011	+4	16	
12	1100	+5	32	
13	1101	+6	64	
14	1110	+7	128	
		/a		(无穷, NaN)

取值范围

	s	exp	frac	*E*	Value	
	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	接近于0
非规格化数	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	最大的非规格化数
	0	0001	000	-6	8/8*1/64 = 8/512	最小的规格化数
	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	接近于1（<1）
规格化数	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	接近于1（>1）
	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	最大的规格化数
	0	1111	000	n/a	inf	

数轴上的分布



一些特例

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}			

浮点数的一些编码特性

- （几乎）可以直接使用无符号整数的比较方式
  - 反例：
    - 必须先比较符号位
    - 考虑+0、-0的特例
    - 还有NaN的问题…
      - (不考虑符号位的话)，NaN比其他值都大
      - 实际的比较结果如何？
  - 其他情况都可以直接使用无符号整数的比较方式
    - 规格化 vs. 非规格化
    - 规格化 vs. 无穷

	s	exp	frac	*E*	Value	
非规格化数	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	接近于0
	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	最大的非规格化数
规格化数	0	0001	000	-6	8/8*1/64 = 8/512	最小的规格化数
	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	接近于1（<1）
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	接近于1（>1）
	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	最大的规格化数
	0	1111	000	n/a	inf	

给定一个实数，如何给出其浮点数表示？

- 基本流程
  - 首先计算出精确值
  - 然后将其转换为所需的精度
    - 可能会溢出（如果指数绝对值很大）
    - 可能需要完成舍入(rounding)操作

- 各种舍入模式

	$1.40	$1.60	$1.50	$2.50	–$1.50
Zero	$1	$1	$1	$2	–$1
Round down	$1	$1	$1	$2	–$2
Round up	$2	$2	$2	$3	–$1
Nearest Even (default)	$1	$2	$2	$2	–$2

给定一个实数，如何给出其浮点数表示？

- 基本流程
  - 首先计算出精确值
  - 然后将其转换为所需的精度
    - 可能会溢出（如果指数绝对值很大）
    - 可能需要完成舍入(rounding)操作

- 各种舍入模式

	\$1.40	\$1.60	\$1.50	\$2.50	–\$1.50
Zero	\$1	\$1	\$1	\$2	– \$1
Round down	\$1	\$1	\$1	\$2	– \$2
Round up	\$2	\$2	\$2	\$3	– \$1
Nearest Even (default)	\$1	\$2	\$2	\$2	– \$2

向偶数舍入(Round-To-Even)

- 这是计算机内默认的舍入方式，也称为“向最接近值的舍入”
  - 其它方式会产生系统误差

- 关键的设计决策是确定两个可能结果的中间数值的舍入
  - 确保舍入后的最低有效数字是偶数
  - 比如：向百分位舍入

1.2349999	1.23	(Less than half way)
1.2350001	1.24	(Greater than half way)
1.2350000	1.24	(Half way—round up)
1.2450000	1.24	(Half way—round down)

向偶数舍入(Round-To-Even)

- 对于二进制数而言
  - “Even” 意味着最低有效数字需为0
  - 而最低有效数字右侧的位串为100…。
- 实例
  - 舍入到小数点后2位

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

7	6 3	2 0
s	exp	frac

- 具体步骤
  - 将数值规格化（小数点前为1）
  - 舍入（round to even）以便符合尾数的位数需求
  - 后调整
- 实例
  - 将8位无符号数转换为8位浮点数（exp域宽度为4 bits，frac域宽度为3bits）

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010

规格化

7	6 … 3	2 … 0
s	exp	frac

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

舍入

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

调整（Postnormalize）

- 舍入操作可能引起溢出

Value	Rounded	E	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

C语言中的浮点数

float　　单精度浮点数

double　　双精度浮点数

- 类型转换
  - 当int（32位宽），float，与double等类型间进行转换时，基本的原则如下：
  - double或float 转换为int
    - 尾数部分截断
    - 如果溢出或者浮点数是NaN，则转换结果没有定义
      - 通常置为 Tmin or Tmax
  - int转换为double
    - 能够精确转换
  - int转换为float
    - [redacted]能被舍入（即精度受损）

◦ 以下判断是否成立，如不成立请给出反例。

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

假设d 与 f 都不是 NaN

- x == (int)(float) x
- x == (int)(double) x
- f == (float)(double) f
- d == (float) d
- f == -(-f);
- 2/3 == 2/3.0
- d < 0.0 ((d*2) < 0.0)
- d > f -f > -d
- d * d >= 0.0
- (d+f)-d == f

给定一个浮点格式，有 k 位指数和 n 位小数，对于下列数，写出阶码 E、尾数 M、小数 f 和值 V 的公式。另外，请描述其位表示。

A. 数 5.0。

B. 能够被准确描述的最大奇整数。

C. 最小的正规格化数。