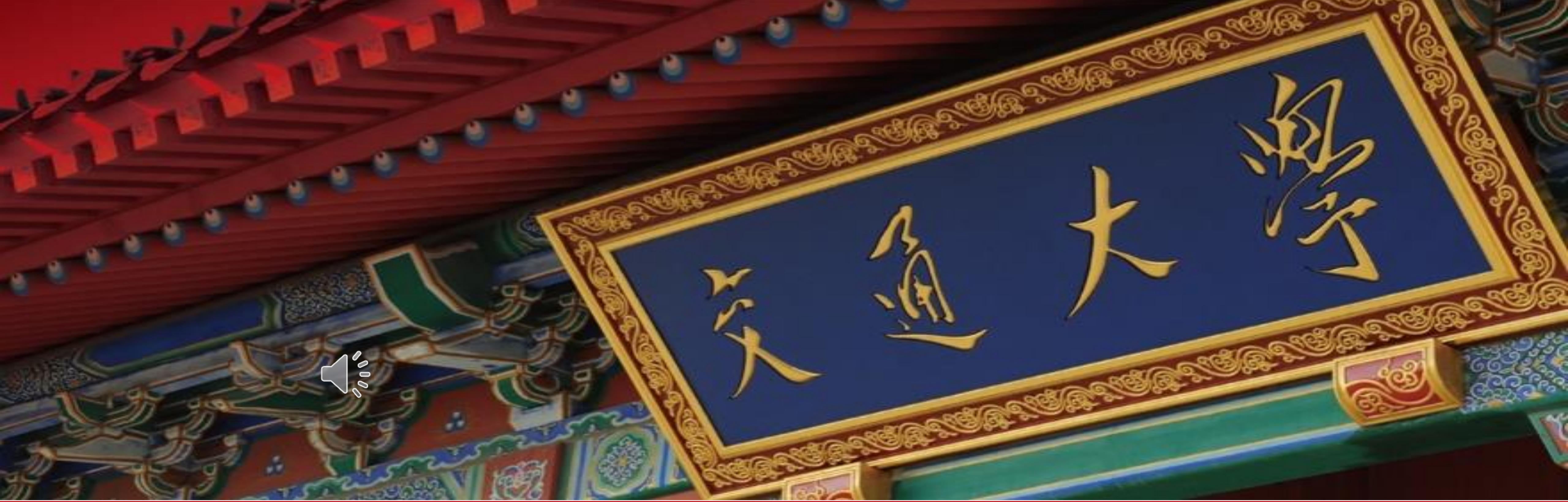

《计算机系统结构》课程直播

2020. 3.10

本次讲课： IEEE754 浮点数

1. 形式
2. 精度
3. 运算与舍入
4. 类型转换

IEEE754 浮点数标准

浮点数标准 IEEE754

- 单精度　Single precision: 32位

s	exp	frac
1	8-bits	23-bits

- 双精度　Double precision: 64 位

s	exp	frac
1	11-bits	52-bits

- 扩展精度　Extended precision: 80 位 (Intel)

s	exp	frac
1	15-bits	63 or 64-bits

单精度浮点数标准 IEEE754...

- 规格化数(Normal):

S(1bit)	E(23~30共8bit)	M(0~22共23bit)

代表数值:　　$(-1)^s \times 1.m \times 2^{e-bias}$

- Bias:
  - Single precision（8bits）: 127 (Exp: 1…254, E: -126…127)
  - Double precision（11-bits）: 1023 (Exp: 1…2046, E: -1022…1023)
- 规格化数的最高数字位总是1，IEEE754标准将这个1缺省存储(隐藏位)，使得尾数表示范围比实际存储多一位

Example

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Represent -0.75_{ten} in single and double-precision formats

  Single: (1 + 8 + 23)

  Double: (1 + 11 + 52)

-

Example

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Represent -0.75_{ten} in single and double-precision formats

Single: (1 + 8 + 23)
1 0111 1110 1000…000

Double: (1 + 11 + 52)
1 0111 1111 110 1000…000

IEEE754浮点数精度

Floating Point (FP) Numbers

- **Floating point numbers**: numbers in scientific notation
  - Two uses
- Use I: real numbers (numbers with non-zero fractions)
  - 3.1415926…
  - 2.1878…
  - $6.62 * 10^{-34}$
- Use II: really big numbers
  - $6.02 * 10^{23}$
- Aside: best not used for currency values
  - Floating Point is Inexact,   e.g. 0.1 (decimal)
  - System.out.print("34.6-34.0=" + (34.6f-34.0f));
  - 34.6-34.0=0.5999985

Floating Point is Inexact

- **Accuracy problems sometimes get bad**
  - FP arithmetic not associative: (A+B)+C not same as A+(B+C)
  - Addition of big and small numbers
  - summing many small numbers)
  - Subtraction of two big numbers
- Example: $(1*10^{30} + 1*10^{0}) - 1*10^{30} = (1*10^{30} - 1*10^{30}) = 0$

- In your code: **never test for equality between FP numbers**
  - Use something like: if (abs(a-b) < 0.00001) then …

IEEE 754 Standard Precision/Range

- Single precision: **`float`** in C
  - 32-bit: 1-bit sign + 8-bit exponent + 23-bit significand
  - Range: $2.0 * 10^{-38} < N < 2.0 * 10^{38}$ (约为 2^{127})
  - Precision: 7 significant (decimal) digits （对应24位尾数位）
  - Used when exact precision is less important (e.g., 3D games)

- Double precision: **`double`** in C
  - 64-bit: 1-bit sign + 11-bit exponent + 52-bit significand
  - Range: $2.0 * 10^{-308} < N < 2.0 * 10^{308}$ (约为 2^{1023})
  - Precision: 15 significant (decimal) digits （对应53位尾数位）
  - Used for scientific computations

- Numbers $>10^{308}$ don't come up in many calculations
  - 10^{80} ~ number of atoms in universe

Exercise

- 设一个变量的值为4098，要求分别用32位补码整数和IEEE 754单精度浮点格式表示该变量（结果用十六进制表示），并说明哪段二进制序列在两种表示中完全相同，为什么会相同？

Exercise

- 设一个变量的值为–2147483647，要求分别用32位补码整数和IEEE754单精度浮点格式表示该变量（结果用十六进制表示），并说明哪种表示其值完全精确，哪种表示的是近似值。

IEEE754浮点数运算与舍入

SHANGHAI JIAO TONG UNIVERSITY

FP Addition – Binary Example

- Consider the following binary example

  $1.010 \times 2^1 \quad + \quad 1.100 \times 2^3$

  Convert to the larger exponent:

  $0.0101 \times 2^3 \quad + \quad 1.1000 \times 2^3$

  Add

  1.1101×2^3

  Normalize

  1.1101×2^3

  Check for overflow/underflow

  Round

  Re-normalize

  IEEE 754 format: 0 10000010 11010000000000000000000

FP Multiplication

- Similar steps:
  - Compute exponent (careful!)
  - Multiply significands (set the binary point correctly)
  - Normalize
  - Round (potentially re-normalize)
  - Assign sign

讨论

对于IEEE754单精度浮点数加减运算，只要对阶时得到的两个价码之差的绝对值|△ E|大于等于（），就无须继续进行后续处理，此时运算结果直接取阶大的那个数

- A. 24
- B. 25
- C. 126
- D. 128

正确答案：B 你选对了

二进制数的最近舍入

- 二进制数 的 Round-To-Even
  - 偶数 “Even”： 最低有效位为 0
  - 数值处于中间 “Half way” 时，要舍弃的位数的形式 = $100\cdots_2$

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

Value	Binary	Rounded	Action	Rounded Value
2 3/32	$10.00\underline{011}_2$	10.00_2	(<1/2—down)	2
2 3/16	$10.00\underline{110}_2$	10.01_2	(>1/2—up)	2 1/4
2 7/8	$10.11\underline{100}_2$	11.00_2	(1/2—up)	3
2 5/8	$10.10\underline{100}_2$	10.10_2	(1/2—down)	2 1/2

Exercise

Sign 1 bit	Exponent 5 bits	Fraction 10 bits
S	E	F

- Given A=2.6125×10^{1}, B=4.150390625×10^{-1},
- Calculate the sum of A and B by hand, assuming A and B are stored by the following format,
- **Assume 1 guard(保护位）, 1 round bit（舍入位）, and 1 sticky bit（粘滞位）** and round to the nearest even（首选“偶数”值舍入）.
- IEEE754规定，浮点运算的中间结果的右边都必须额外多保留两位（**保护位、舍入位**）为获得无限精度求出后的舍入效果，再加一个**粘滞位。**

- $2.6125 \times 10^{1} + 4.150390625 \times 10^{-1}$

  $2.6125 \times 10^{1} = 26.125 = 11010.001 = 1.1010001000 \times 2^{4}$

  $4.150390625 \times 10^{-1} = .4150390625 = .011010100111$

  $=1.1010100111 \times 2^{-2}$ （对阶， 小阶往大阶对）

  Shift binary point 6 to the left to align exponents,

  GR

  1.1010001000 00

  +.0000011010 10 0111 (Guard = 1, Round = 0, Sticky = 1)

  1.1010100010 10 （尾数相加） and（尾数规格化检查）

the extra bits (G,R,S) are more than half of the least significant bit (0).
Thus, the value is rounded up. （舍入）

$1.1010100011 \times 2^{4}$（检查，无溢出）

$= 11010.100011 \times 2^{0} = 26.546875 = 2.6546875 \times 10^{1}$

Rounding

1. BBBGRXXX

Guard bit: 1st bit removed

Round bit: 2nd bit removed

Sticky bit:
OR of remaining bits

- Round up conditions

Value	*Fraction*	*GRS*	*Incr?*	*Rounded*
128	1.0000000	000	N	1.000
15	1.1010000	000	N	1.101
17	1.0001000	100	N	1.000
19	1.0011000	100	Y	1.010
138	1.0001010	101	Y	1.001
63	1.1111100	110	Y	10.000

Postnormalize

- Issue
  - Rounding may have caused overflow
  - Handle by shifting right once & incrementing exponent

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

Arithmetic Latencies

- Latency in cycles of common arithmetic operations
- Source: *Agner Fog,* *https://www.agner.org/optimize/#manuals*
  - AMD Ryzen core

	Int 32	Int 64	Fp 32	Fp 64
Add/Subtract	1	1	5	5
Multiply	3	3	5	5
Divide	14-30	14-46	8-15	8-15

- Divide is variable latency based on the size of the dividend
  - Detect number of leading zeros, then divide
- Why is FP divide faster than integer divide?

浮点数类型转换

练习

设int x=1,float y=2,则表达式x/y的值是：（）

A. 0

B. 1

C. 2

D. 以上都不是

Floating Point in C

- C Guarantees Two Levels
  - **`float`** single precision
  - **`double`** double precision
- Conversions/Casting
  - Casting between **`int`**, **`float`**, and **`double`** changes bit representation
  - **`double`**/**`float`** → **`int`**
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - **`int`** → **`double`**
    - Exact conversion, as long as **`int`** has ⩽ 53 bit word size
  - **`int`** → **`float`**
    - Will round according to rounding mode

Floating Point Puzzles

- For each of the following C expressions, either:
  - Argue that it is true for all argument values
  - Explain why not true

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

Assume neither **d** nor **f** is 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 < 0.0      ⇒   ((d*2) < 0.0)`
- `d > f        ⇒   -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

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}			

Special Properties of the IEEE Encoding

- FP Zero Same as Integer Zero
  - All bits = 0
- Can (Almost) Use Unsigned Integer Comparison
  - Must first compare sign bits
  - Must consider −0 = 0
  - NaNs problematic
    - Will be greater than any other values
    - What should comparison yield?
  - Otherwise OK
    - Denorm vs. normalized
    - Normalized vs. infinity

Summary

- IEEE Floating Point has clear mathematical properties
- Represents numbers of form M x 2^E
- One can reason about operations independent of implementation
  - As if computed with perfect precision and then rounded
- Not the same as real arithmetic
  - Violates associativity/distributivity
  - Makes life difficult for compilers & serious numerical applications programmers

下一节

- 周四 8：00
- 存储系统
- 请做好准备

再见