



清华大学  
Tsinghua University

# 第一单元 第六讲

## 浮点数表示和运算 第一单元小结

刘卫东

计算机科学与技术系

# 内容提要



清华大学  
Tsinghua University

## ✚ 浮点数

## ✚ 科学记数法

- ✚ 十进制科学记数法

- ✚ 二进制科学记数法

## ✚ IEEE 754 浮点数标准

## ✚ 浮点数表示

- ✚ 表示范围 vs. 表示精度

## ✚ 浮点数运算

## ✚ 浮点数运算器

## ✚ 本单元小结

# 计算机内的数据



✚ 计算机的功能：处理数据

✚ n 位能表示哪些数据？

✚ 无符号整数： 0 to  $2^n - 1$

✚ 有符号整数：  $-2^{(n-1)}$  to  $2^{(n-1)} - 1$

✚ 其它数据呢？

✚ 大整数？ (如：一个世纪的秒数)

$3,155,760,000_{10}$  ( $3.15576_{10} \times 10^9$ )

✚ 非常小的数？ (如：原子的直径)

$0.00000001_{10}$  ( $1.0_{10} \times 10^{-8}$ )

✚ 有理数 (如：循环小数)

$2/3$  ( $0.666666666\dots$ )

✚ 无理数  $2^{1/2}$  ( $1.414213562373\dots$ )

(无限不循环小数): e ( $2.718\dots$ ),  $\pi$  ( $3.141\dots$ )

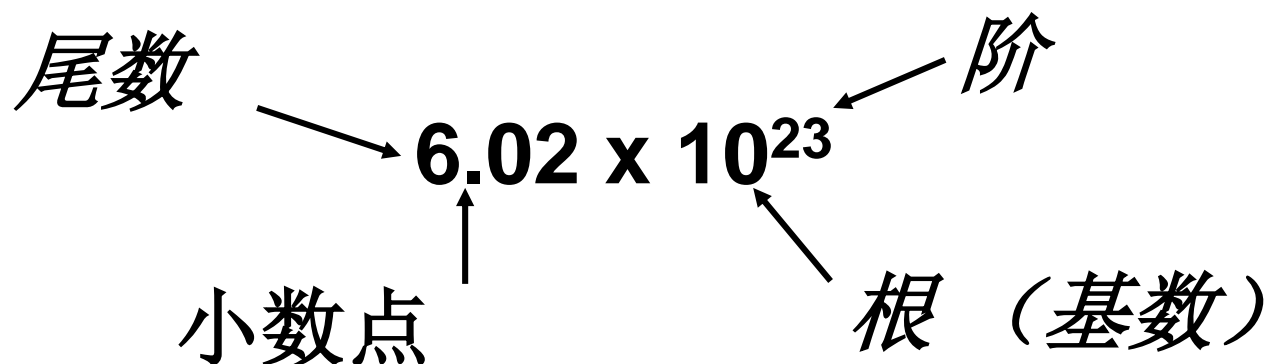
实数

# 实数的表示



## ❖ 实数和浮点数

## ❖ 十进制实数的科学记数法



❖ 固定小数点位置，表示方式唯一

❖ 主要包含3个部分

◆ 尾数、阶、根

# 二进制浮点数科学记数法



清华大学  
Tsinghua University

尾数

阶

$$1.0_2 \times 2^{-1}$$

“二进制小数点”根（基数）

✚ 需要在计算机内表示的部分

▣ 尾数

▣ 阶

# 浮点数表示



浮点数是数学中实数的子集合，由一个纯小数乘上一个指数值两部分组成。在计算机内，其纯小数部分被称为浮点数的尾数，对非0值的浮点数，要求尾数的绝对值必须 $\geq 1$ ，称满足这种表示要求的浮点数为规格化表示；

把不满足这一表示要求的尾数，变成满足这一要求的尾数的操作过程，叫作浮点数的规格化处理，通过移位尾数和修改阶码实现。

尾数包括有：符号位+尾数的绝对值

阶码：符号位+阶码的绝对值

# 浮点数的机器表示



⊕ 尾数：定点小数

⊕ 阶码：整数

⊕ 如何表示？

$$\boxplus X = M_s E_s E_m \dots E_2 E_1 M_{-1} M_{-2} \dots M_{-n}$$

⊠ (n+1)位尾数

⊠ (m+1)位阶码

⊠ 表示范围和表示精度？

# IEEE 浮点数标准754



浮点数:  $X = M_S E_S E_m \dots E_2 E_1 M_{-1} M_{-2} \dots M_{-n}$

IEEE 标准: 阶码用移码, 对规格化数阶码用移127方案

尾数用原码, 对规格化数的尾数用隐藏位技术  
支持正负无穷大的浮点数和非规格化的浮点数  
设置一个非法浮点数编码供编程人员用于排错

	符号位数	阶码位数	尾数位数	总位数
短浮点数:	1	8	23	32
长浮点数:	1	11	52	64



# 浮点数的尾数部分



浮点数:  $X = M_s E_s E_m \dots E_2 E_1 M_{-1} M_{-2} \dots M_{-n}$

**IEEE 标准**: 阶码用移码, 基为2;

## 尾数用原码表示

按 IEEE 的浮点数标准, 尾数用**规格化原码**表示, 即符号位  $M_s$  用 0 表示正, 1 表示负, 且非 0 值尾数数值的最高位**必定为 1**; 既然这位必定为 1, 则在保存浮点数到内存前, 通过尾数左移, 强行把该位去掉, 则用同样多的尾数位就能多存一位二进制数, 有利于提高数据表示精度, 把这种处理方案称为**隐藏位技术**。当然, 在取回这样的浮点数到运算器执行运算时, 必须先恢复该隐藏位。

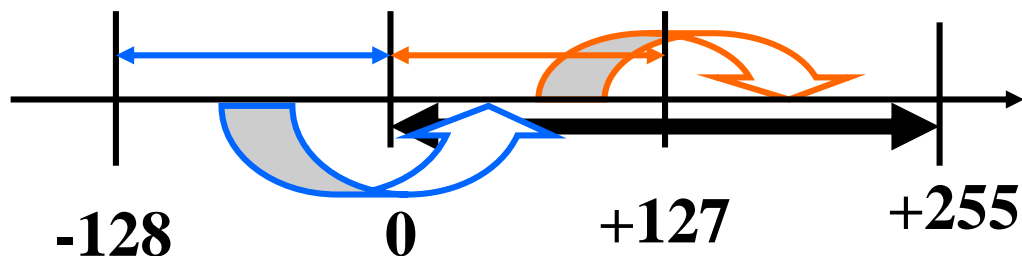
# 阶码的移码表示法



✿ 移码： 整数补码+偏移值

✿  $[E]_{\text{移}} = E + \text{OFFset} \quad -2^n \leq E < 2^n$

✿ 使浮点数0的机器表示为全0



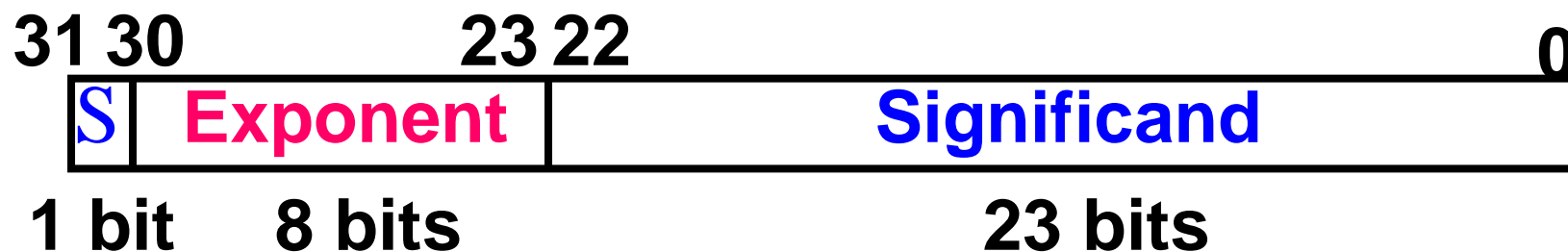
对**移127**的方案，8位移码表示的机器数是数的真值在数轴上**向右平移**了127个位置。

# IEEE754浮点数标准



规格化形式:  $+1.xxxxxxxxxxxx_2 * 2^{yyyy_2}$

单精度浮点数(32 bits)



S 表示 符号位

Exponent 表示 y, 即阶, 移127

Significand 表示 x, 即尾数的后部分

可表示的范围:  $2.0 \times 10^{-38}$  至  $2.0 \times 10^{38}$

# IEEE754 浮点数标准



规定对长、短浮点数的尾数使用**隐藏位**技术，即把规格化**非 0 值尾数**的最高位上的 1 经过左移操作后强行去掉，则原来不能表示的更低一位进到最低一位。对单精度浮点数采用隐藏位之后，就使 23 位的规格化尾数数值位能给出 24 位的精度。

短浮点数采用**移127**的方案，阶码值范围：**00000001~11111110**，表示  $-126 \sim +127$ 。还有 **2 个特定的阶码值**：

**00000000** 跟 **23 位的非 0 尾数**表示非规格化浮点数（隐藏位必定为 0），**00000000** 跟 **全 0 尾数**是浮点数 0。

**11111111** 跟 **全 0 尾数**表示无穷大的浮点数，可正可负，由符号位决定。**11111111** 跟 **非全 0 尾数**时属于**非法数值**。

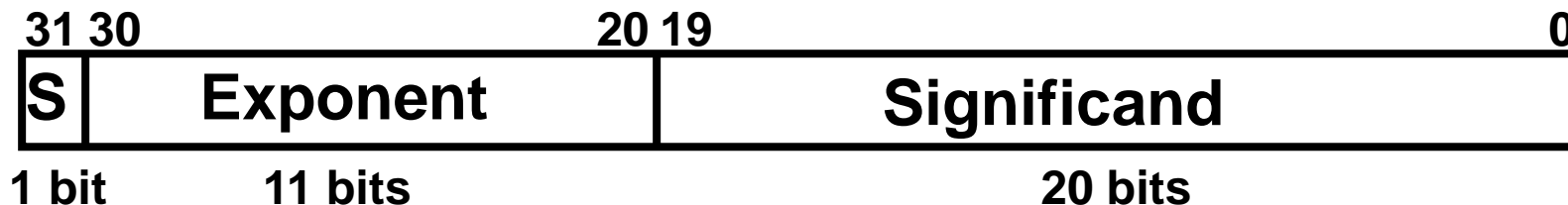
# IEEE 754浮点数标准



S(1位)	E(8位)	M(23位)	X(共32位)
符号位	0	0	0
符号位	1到254之间	不等于0	$(-1)^S \cdot 2^{E-127} \cdot (1.M)$ 规格化
符号位	255	0	无穷大
符号位	255	不等于0	NaN(非数值)

鉴于 IEEE754标准 对计算机界的重要贡献, 发挥关键作用的数学家 Kahan 于1989年被授予图灵奖。

# 双精度浮点数



✱ C 语言中的 `double` 类型

✱ 尾数：原码

✱ 阶：移1023

✱ 十进制的范围扩展到  
 $2.0 \times 10^{-308}$  至  $2.0 \times 10^{308}$

✱ 最主要的好处是精度得到了扩展 (52 位)

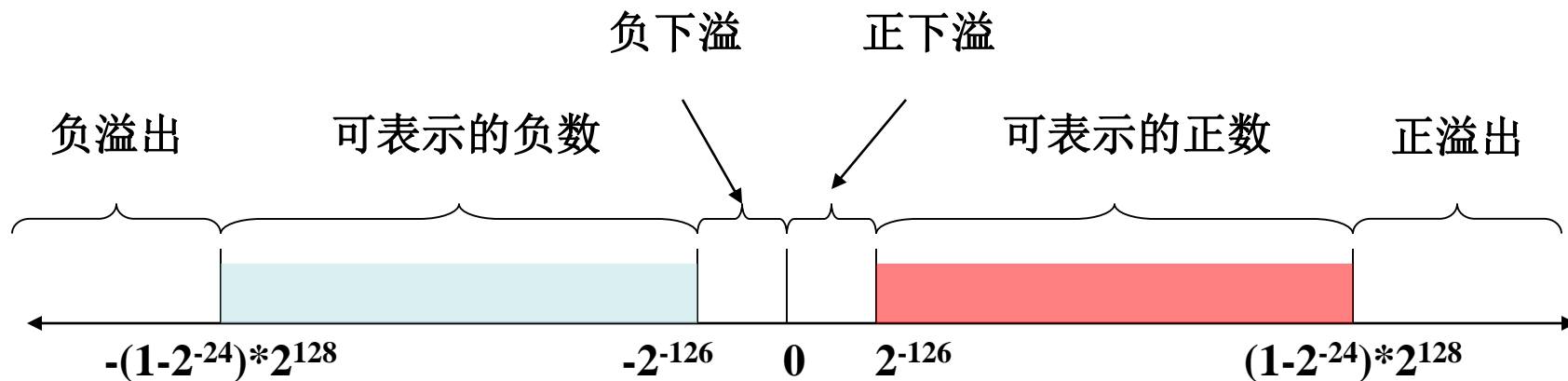
# IEEE754浮点数标准(总结)



- 被几乎所有计算机采纳 (自1985年起)
- 符号位:  $\begin{cases} 1 \text{ 表示负数} \\ 0 \text{ 表示正数} \end{cases}$
- 有效位:
  - 使用原码表示
  - 规格化小数中, 隐含最高位1
  - 单精度为: 23 位, 双精度为 52 位
  - $0 < \text{有效数} < 1$
- 阶: 单精度: 移127, 双精度: 移1023
- 全0用来表示0值
  - 在阶码中保留0给数0

$$(-1)^S * (1 + \text{Significand}) * 2^{\text{Exp}}$$

# 特殊的浮点数值



特殊值	阶	有效数
$\pm 0$	0000 0000	0
非规格化数	0000 0000	非0
NaN	1111 1111	非0
$\pm \infty$	1111 1111	0



# Not a Number



下列结果是什么: `sqrt(-4.0)` or `0/0`?

- ❑ 如果无穷大不是错误的话, 那以上也不算
- ❑ 称其为 Not a Number (NaN)
- ❑ 阶 = 255, 有效位非0

应用

- ❑ NaN可帮助排错
- ❑ 自包含:  $\text{op}(\text{NaN}, X) = \text{NaN}$

# 上溢和下溢



## 上溢

- ❑ 数的绝对值太大 ( $> 2.0 \times 10^{38}$ )
- ❑ 阶的值超出8位能表示的范围

## 下溢

- ❑ 数的绝对值太小
  - ◆  $> 0, < 2.0 \times 10^{-38}$
- ❑ 阶码超出了8位二进制位能表示的范围

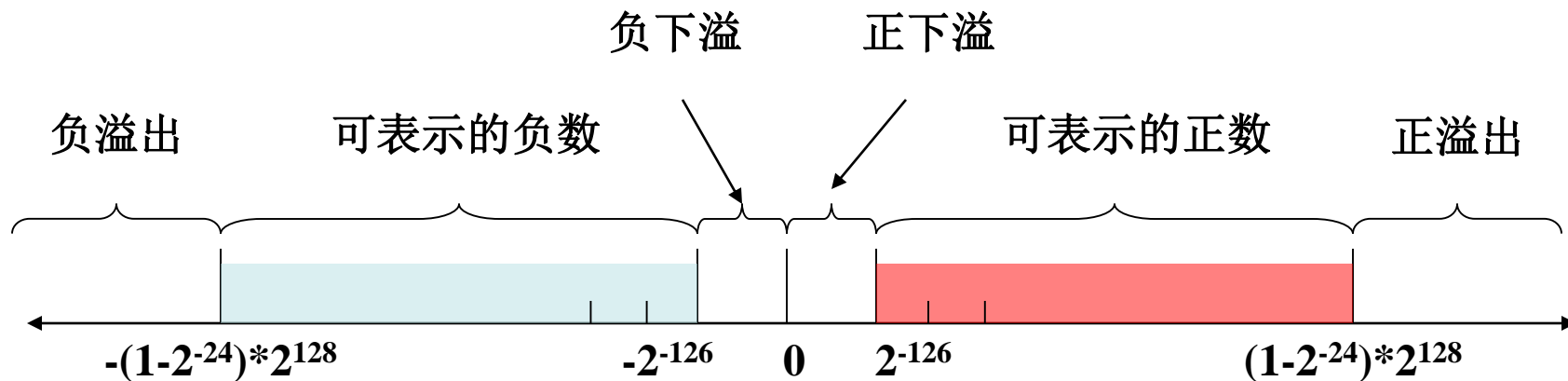
## 如何减少上溢和下溢?

## 如何提高数据表示的精度?

# 浮点数溢出



清华大学  
Tsinghua University



可表示的大于0的最小规格化数（隐藏位的值为1）：

**正数：0 00000001 000000000000000000000000**

可表示的大于0的最小非规格化数（不使用隐藏位技术）：

**正数：0 00000000 0000000000000000000000001**

# 非规格化数



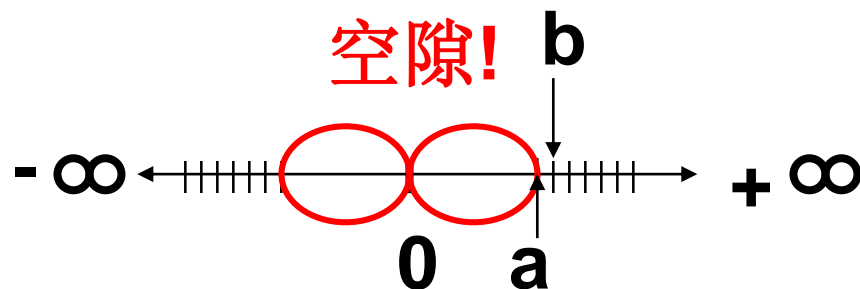
❁ 问题：在0周围还有一些空隙没有用来表示浮点数

❁ 最小的正数：  $a = 1.0 \cdots_2 * 2^{-126} = 2^{-126}$

❁ 次小的正数：  $b = 1.0 \cdots 01_2 * 2^{-126} = 2^{-126} + 2^{-149}$

❁  $a - 0 = 2^{-126}$

❁  $b - a = 2^{-150}$

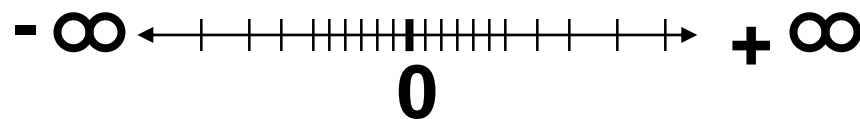


❁ 解决办法：

❁ 使用非规格化数：没有隐含的前导1

❁ 最小的正数：  $a = 2^{-149}$

❁ 次小的正数：  $b = 2^{-148}$



# 舍入



- ✚ 浮点数的算术运算  $\Rightarrow$  舍入
- ✚ 类型转换时也需要舍入
  - ▣ Double  $\Leftrightarrow$  single precision  $\Leftrightarrow$  integer
- ✚ 向上舍入
  - ▣  $2.001 \Rightarrow 3$ ;  $-2.001 \Rightarrow -2$
- ✚ 向下舍入
  - ▣  $1.999 \Rightarrow 1$ ;  $-1.999 \Rightarrow -2$
- ✚ 截断
  - ▣ 丢弃最后的位（向0舍入）

# 浮点数的二—十进制转换



0	0110 1000	101 0101 0100 0011 0100 0010
---	-----------	------------------------------

- 符号位: 0 => 正数

- 阶:

- $0110\ 1000_2 = 104_{10}$

- 移码校正:  $104 - 127 = -23$

- 有效数:

- $1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + \dots$   
 $= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-22}$   
 $= 1.0 + 0.666115$

⊕ 十进制值:  $1.666115 \times 2^{-23} \sim 1.986 \times 10^{-7}$

# 浮点数十—二进制转换



✚ 简单情况：如果除数是2的整数倍，则比较简单

✚ 如： -0.75的二进制

▣  $-0.75 = -3/4$

▣  $-11_2/100_2 = -0.11_2$

▣ 规格化为：  $-1.1_2 \times 2^{-1}$

▣  $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

▣  $(-1)^1 \times (1 + .100\ 0000 \dots 0000) \times 2^{(126-127)}$

1	0111 1110	100 0000 0000 0000 0000 0000
---	-----------	------------------------------

# 浮点数十—二进制转换



## ✚ 除数不是2的整数倍

- ▣ 该数无法精确表示
- ▣ 可能需要多位有效位来保证精度
- ▣ 难点：如何得到有效位？

## ✚ 循环小数有一个循环体

## ✚ 转换

- ▣ 求出足够多的有效位。
- ▣ 根据精度要求（单、双）截断多余的位。
- ▣ 按标准要求给出符号位、阶和有效位。



# 转换举例



- 3.333333...

$$\begin{array}{r} 0.33333333 \\ \times 2 \\ \hline 0.66666666 \end{array}$$

$$\begin{array}{r} 0.66666666 \\ \times 2 \\ \hline 1.33333332 \end{array}$$

$$\begin{array}{r} 0.33333332 \\ \times 2 \\ \hline 0.66666664 \end{array}$$

- 11.0101010...  $\Rightarrow$  - 1.1010101..  $\times 2^1$

- 有效位: 101 0101 0101 0101 0101 0101
- 符号位: 负  $\Rightarrow$  1
- 阶:  $1 + 127 = 128_{10} = 1000\ 0000_2$

1	1000 0000	101 0101 0101 0101 0101 0101
---	-----------	------------------------------

# 浮点数算术运算



## 浮点数加减运算

$$X = M_X \times 2^{E_X} \qquad Y = M_Y \times 2^{E_Y}$$

- (1) 对阶操作，求阶差： $\Delta E = E_X - E_Y$ ，  
使阶码小的数的尾数右移  $|\Delta E|$  位，  
其阶码取大的阶码值；
- (2) 尾数加减；
- (3) 规格化处理；
- (4) 舍入操作，可能带来又一次规格化；
- (5) 判结果的正确性，即检查阶码上下溢出

# 浮点数加运算举例



$$X=2^{+010} \times 0.1101111, \quad Y=2^{+100} \times (-0.1010110)$$

写出X、Y的正确的浮点数表示：

阶码用 4 位移码    尾数用 8 位原码  
(含符号位)                      (含符号位)

$$[X]_{\text{浮}} = 0 \ 1 \ 010 \ 1101111$$

$$[Y]_{\text{浮}} = 1 \ 1 \ 100 \ 1010110$$

为运算方便，尾数的符号位写在数值位之前：

$$[X]_{\text{浮}} = 1 \ 010 \ 0 \ 1101111$$

$$[Y]_{\text{浮}} = 1 \ 100 \ 1 \ 1010110$$

# 浮点数加运算举例



$$X=2^{+010} \times 0.1101100, \quad Y=2^{+100} \times (-0.1010110)$$

(1) 计算阶差（移码计算）：

$$\Delta E = E_X - E_Y = E_X + (-E_Y) = 1\ 010 + 0\ 100 = 0\ 110$$

**注意：**阶码计算结果的符号位在此变了一次反，为 -2 的移码，

是X的阶码值小，使其取 Y 的阶码值1100（即 +4）；  
因此，相应地修改  $[M_X]_{\text{原}} = 0\ 0011011\ 00$ （即右移2位）

(2) 尾数求和：

右移出的00被保存到保护位中

$$\begin{array}{r} 1\ 1010110 \\ -\ 0\ 0011011\ 00 \\ \hline 1\ 0111011\ 00 \end{array}$$

此处是原码加法，符号不相同，绝对值大的减小的，结果符号取决于绝对值大的数

# 浮点数加运算举例



## (3) 规格化处理:

相加结果,数值的最高位为0,应执行1次左规操作,  
故得  $[M_{X+Y}]_{\text{原}} = 1\ 1110110$ , 阶码减1得 **1 011 (为+3)**

(4) 舍入处理: 舍入位是**0**, 按0舍1入规则, 得到最终结果: **1 1110110**

(5) 检查溢出否: 和的阶码为 1011, **不溢出**

计算后的  $[X+Y]_{\text{浮}} = \mathbf{1\ 1011\ 1110110}$

即数的实际值为:  $2^3 \times (-0.1110110)$

# 浮点数乘、除法



## ✚ 算法

✚ 阶码加、减：乘：  $E_X + E_Y$  ， 除：  
 $E_X - E_Y$

- ✚ 对尾数进行乘、除法，求得结果
- ✚ 规格化
- ✚ 舍入，可能再次规格化
- ✚ 进行溢出检查（阶码）

# 浮点数运算



## 对浮点数加/减法

- 移码的减运算

- 无符号数运算

## 对浮点数乘/除法

- 移码的加/减运算（注意溢出）

## 浮点数尾数运算

- 原码运算

# 浮点运算的特点



## ✚ 浮点加、减法不满足结合律!

❏  $x = -1.5 \times 10^{38}$ ,  $y = 1.5 \times 10^{38}$ , and  $z = 1.0$

❏  $x + (y + z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0)$   
 $= -1.5 \times 10^{38} + (1.5 \times 10^{38}) = \underline{0.0}$

❏  $(x + y) + z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0$   
 $= (0.0) + 1.0 = \underline{1.0}$

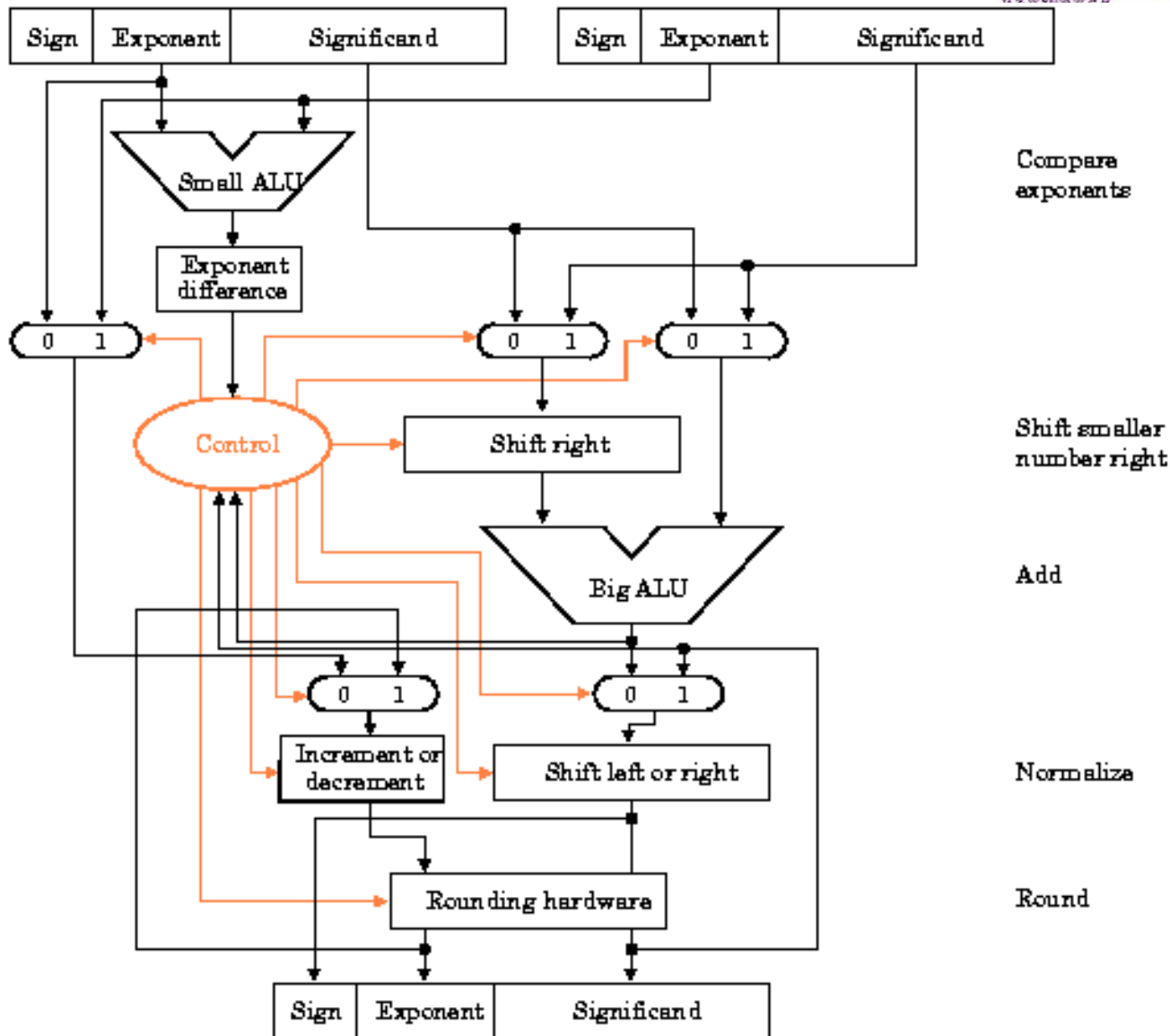
## ✚ 浮点数加法、减法不可结合!

## ✚ 浮点数也不能进行相等比较!

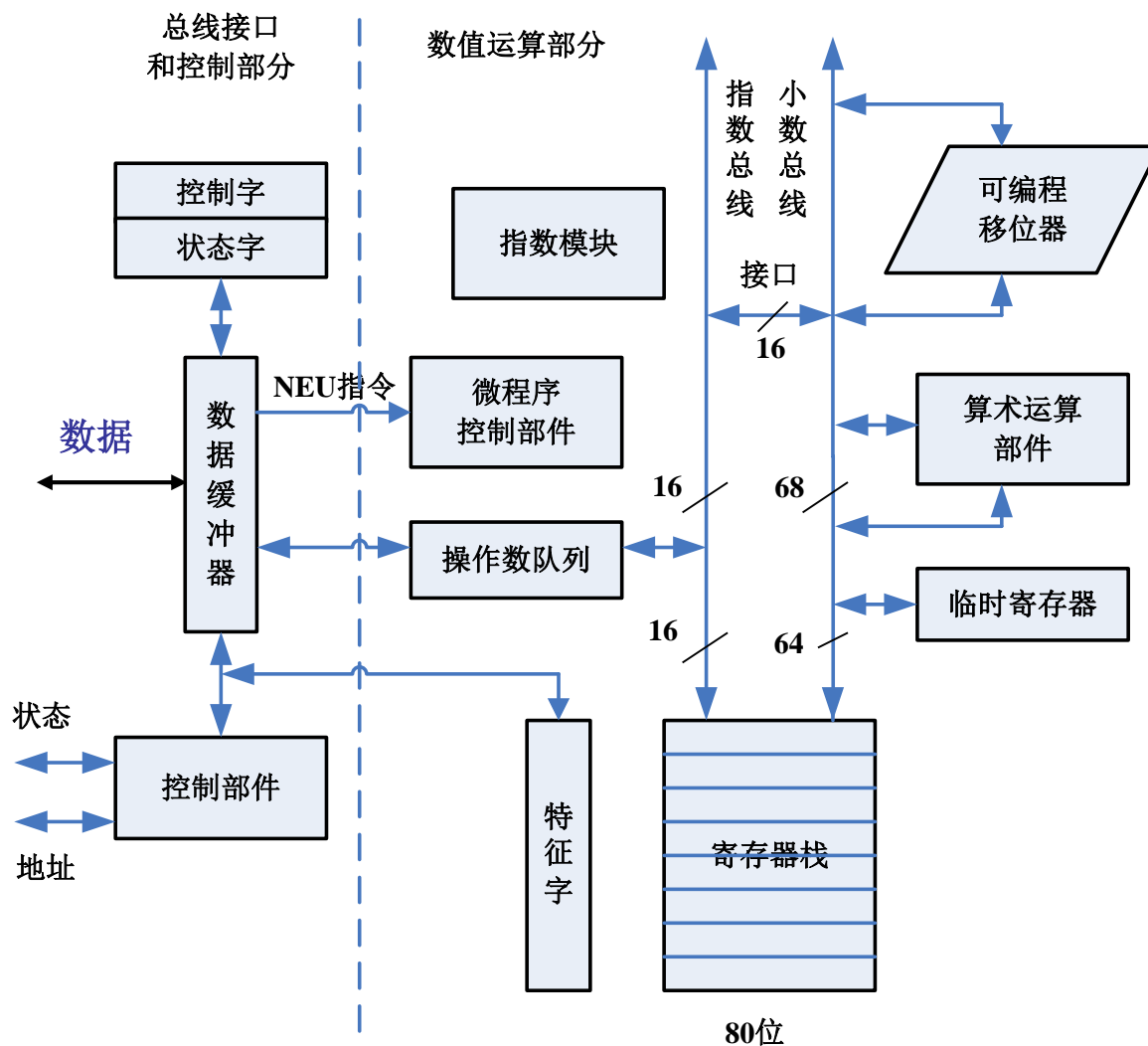
❏ 为什么? 浮点数算术运算的结果是近似值。



# 浮点运算部件



# Intel 80287

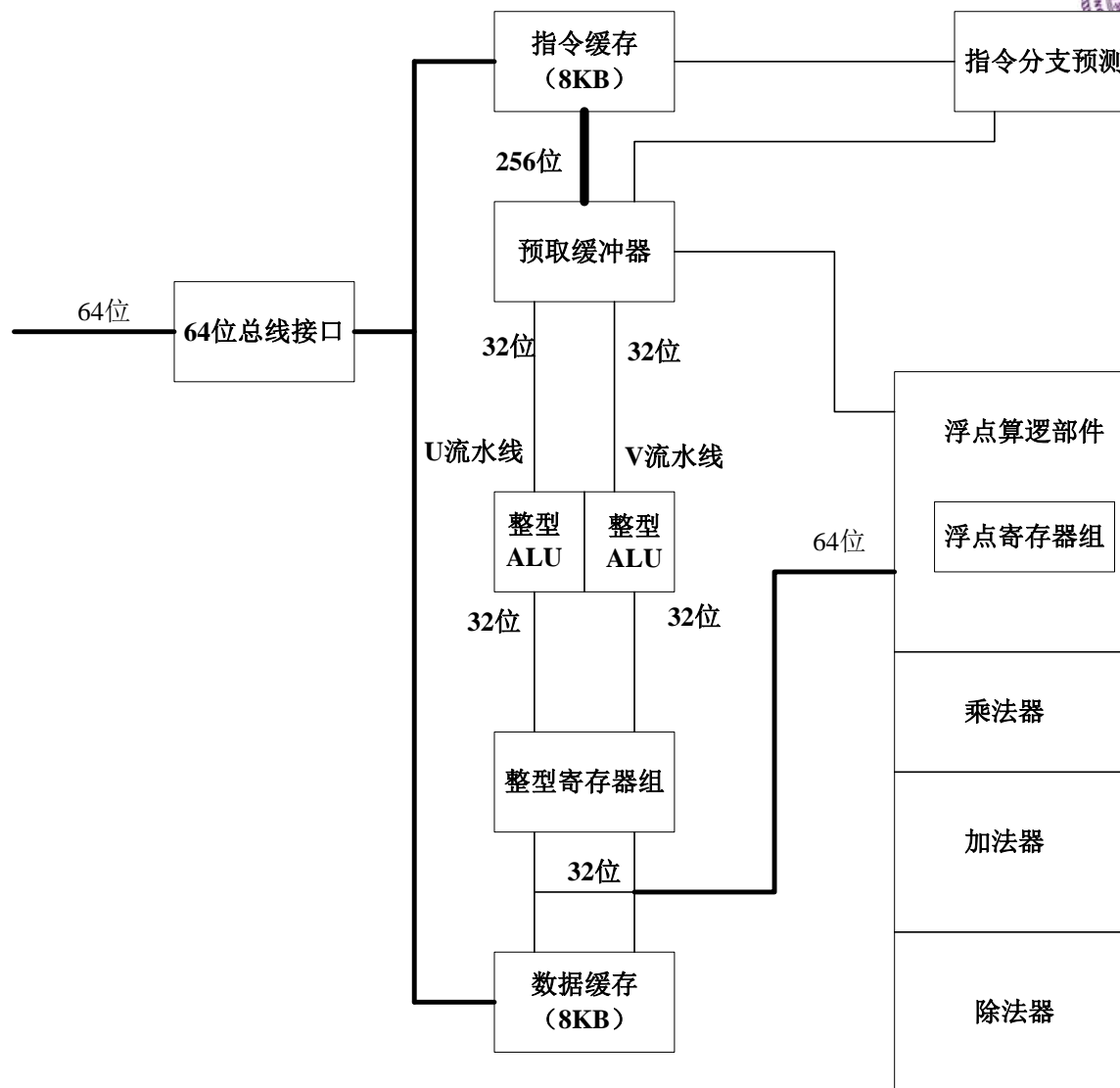


浮点运算部件  
以协处理器方式和  
CPU 连接，有独立  
的控制逻辑；

8个 80位 浮点  
数寄存器，精度更  
高，采用堆栈结构  
并进行了扩展；

支持 3 大类共  
7 种数据，支持约  
60 条指令。

# Pentium结构



Pentium结构简图

# 数据及数据类型



**0011 0100 0101 0101 0100 0011 0100 0010**

**–1.986 \*10<sup>-7</sup>**

**–878,003,010**

**–“4UCB”**

**ADD R0, R1**

- 计算机中的数据可以表示任何事情：指令、操作数等，由上层次的抽象计算机来判断。
- 对存储内容的错误理解：
  - 将ASCII码当作浮点数，指令作为数据，整数可能成为指令，...
  - 程序中的安全漏洞

# 小结



- ❖ 计算机中的浮点数是我们实际使用的数的近似值
- ❖ IEEE 754 浮点数标准是浮点数运算中广为接受的标准
- ❖ 浮点数运算一般由浮点运算器完成
  - ❑ 阶码运算
  - ❑ 尾数运算
- ❖ 计算机中的二进制位只有在上下文才有意义，单独的一个字不代表任何含义。

# 本单元小结



- ❖ 程序和指令
- ❖ 数据表示
- ❖ 检错纠错码
- ❖ 数据运算
- ❖ 运算器基本功能
- ❖ 运算器组成
- ❖ **VHDL硬件描述语言**

# 程序和指令



## ✚ 程序

- ❏ 高级语言程序
- ❏ 汇编语言程序
- ❏ 机器语言程序

## ✚ 指令

- ❏ 指挥硬件进行操作的命令
- ❏ 指令系统
- ❏ 指令格式
- ❏ 寻址方式

# 数据表示



## ❖ 数据类型

### ❑ 数值型

◆ 整数、浮点数

### ❑ 非数值型

◆ 字符型、逻辑型

### ❑ 其他型

◆ 多媒体数据

◆ 程序

## ❖ 二进制数据表示



# 整数的原反补码表示



正数的原码、反码、补码表示均相同，  
符号位为 0，数值位同数的真值。

零的原码和反码均有 2 个编码，补码只 1 个码

负数的原码、反码、补码表示均不同，  
符号位为 1，数值位：原码为数的绝对值  
反码为每一位均取

反码

补码为反码再在最

低位+1

由  $[X]_{\text{补}}$  求  $[-X]_{\text{补}}$ ：每一位取反后再在最低位+1

# 浮点数表示法



## 单精度浮点数

- 用32位字表示一个浮点数
- 符号位（0正1负）、阶（移127码）、有效数（23位原码，规格化表示中隐含最高位1）

## 双精度浮点数

- 用64位双字表示一个浮点数
- 符号位（0正1负）、阶（移1023码）、有效数（52位原码，规格化表示中隐含最高位1）

# 数据运算



## 逻辑运算

- 逻辑门

## 加法运算

- 加法器

## 减法运算

- 加法器

## 乘、除法运算

- 加法器、移位寄存器

## 浮点运算器

- 浮点运算器

# 运算器基本功能



- ✚ 完成算术、逻辑运算
  - ▣  $+$ 、 $-$ 、 $\times$ 、 $\div$ 、 $\wedge$ 、 $\vee$ 、 $\neg$ 。
- ✚ 获得运算结果的状态
  - ▣ C、Z、V、S
- ✚ 取得操作数
  - ▣ 寄存器组、立即数
- ✚ 输出、存放运算结果
  - ▣ 寄存器组、数据总线
- ✚ 暂存运算的中间结果
  - ▣ Q寄存器、移位寄存器
- ✚ 理解、响应控制信号

输出Y

/OE

二选一

F

F3  
F=0000  
OVR  
Cn+4

A L U  
S R

Cn

三选一

二选一

运算器，三大件  
运算 暂存 乘除快  
多路选通连起来

数据组合有内外  
运算功能指明白  
存移输出巧安排

运算功能选择  
← I5 I4 I3

数据组合选择  
← I2 I1 I0

运算结果处理  
← I8 I7 I6

输入D

B锁存器

A锁存器

Q寄存器

Q3

Q0

三选一

B 16个 A  
通用寄存器

A口地址

B口地址

RAM3

三选一

RAM0

# VHDL语言



清华大学  
Tsinghua University

✿ 掌握用VHDL语言描述硬件结构和功能的基本流程和方法

# 阅读与思考



## 阅读：

教材第3章

## 思考

- IEEE754浮点数尾数用原码,阶码用移码的原因?
- 运算器主要功能是什么? 都是怎样在电路上实现的?
- 运算器的功能如此简单, 为什么程序能完成十分复杂的功能?

## Project2: ALU 实验

- 具体要求见：实验指导书5.3
- 各班课代表与实验室联系实验时间(10月17日前完成)
- 本实验按班到实验室分组独立完成，在网络学堂上提交实验报告

## 其他说明

- 以后的实验按组进行