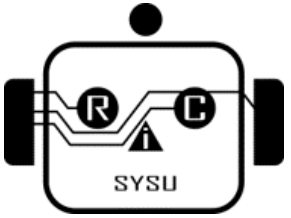
中山大學
SUN YAT-SEN UNIVERSITY



计算机组成原理

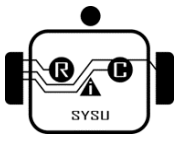
第三章：计算机中的运算

中山大学计算机学院
陈刚

2022年秋季

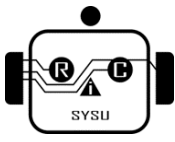
本讲内容

- 基本运算
  - 加法和减法
  - 位操作
- 乘法运算
- 除法运算
- 浮点运算
  - 基本概念和问题
  - 浮点加减法
  - 浮点乘除法
  - 运算精度问题



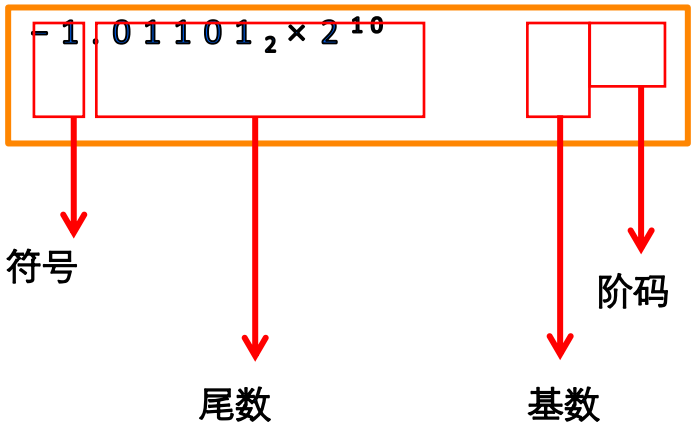
本讲内容

- 基本运算
  - 加法和减法
  - 位操作
- 乘法运算
- 除法运算
- 浮点运算
  - 基本概念和问题
  - 浮点加减法
  - 浮点乘除法
  - 运算精度问题



浮点数的表示

- 基于浮点数表示的计算机算术运算称为浮点运算
- 实数的二进制浮点表示



浮点数的表示

- 基于浮点数表示的计算机算术运算称为浮点运算
- 计算机中二进制浮点数的表示方法—IEEE 754标准

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

符号位　隐藏位　尾数的小数部分　基数　阶码　偏置常数

$-1.01101_2\times 2^{10}$

浮点数的表示

❑IEEE 754单精度表示

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	Exponent								Fraction																						

❑ IEEE 754双精度表示

Bias=127

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
S	Exponent											Fraction																			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fraction																															

Bias=1023

浮点数的表示

IEEE 754单精度表示

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	Exponent								Fraction																						

$$-1.01101_2 \times 2^{10}$$

浮点数的表示

IEEE 754单精度表示

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Exponent								Fraction																						

$-1.01101_2 \times 2^{10}$

浮点数的表示

❑IEEE 754单精度表示

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	Fraction																						

2（实际阶码）+127（bias）=129（阶码）

0	0	0	0	0	0	0	0

最小值

1	1	1	1	1	1	1	1

最大值

$-1.01101_2 \times 2^{10}$

浮点数的表示

IEEE 754单精度表示

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$-1.01101_2 \times 2^{10}$

❑IEEE 754浮点数的编码表示

阶码全0和全1用作特殊值处理

单精度		双精度		表示的数
Exponent	Fraction	Exponent	Fraction	
0	000	0	000	0
0	非零	0	非零	±非规格化数
1~254	任意	1~2046	任意	±浮点数
255	0	2047	0	±无穷
255	非零	2047	非零	NaN（非数）

IEEE754标准规定的五种异常情况

① 无效操作(无意义)

- 操作中有一个数是非有限数，如： 加/减∞、0×∞、 ∞÷y等
- 结果无效，如：源操作数是NaN等

② 除以0(即：无穷大)

③ 数太大(阶码上溢)

如：对于SP，阶码E >1111 1110 (指数大于127)

④ 数太小(阶码下溢)

如：对于SP，阶码E < 0000 0001(指数小于-126)

⑤ 结果不精确(舍入引起)

如：1/3不能精确表示成浮点数

异常情况硬件可以检测到，
可设定由硬件/软件处理

有关浮点数运算的问题

- 实现一套浮点数运算指令，要解决的问题

① Representation(表示)

- Normalized form (规格化形式) 和 Denormalized form (非规格化形式)
- 单精度和 双精度格式

② Range and Precision(表示范围和精度)

③ Arithmetic (运算：+、 -、 ×、 ÷)

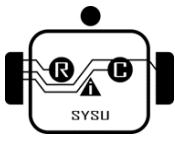
④ Rounding(舍入)

⑤ Exceptions (异常处理)

⑥ Errors(误差)与精度控制

本讲内容

- 基本运算
  - 加法和减法
  - 位操作
- 乘法运算
- 除法运算
- 浮点运算
  - 基本概念和问题
  - 浮点加减法
  - 浮点乘除法
  - 运算精度问题



浮点数加法

十进制科学计数法的加法实例：

$A=0.123 \times 10^5$；$B=0.456 \times 10^2$；求A+B

其计算过程为：

$$0.123 \times 10^5 + 0.456 \times 10^2$$

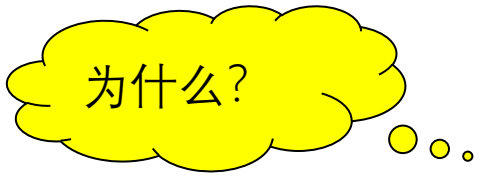
$$= 0.123 \times 10^5 + 0.000456 \times 10^5$$

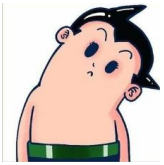
$$= (0.123 + 0.000456) \times 10^5 = 0.123456 \times 10^5$$

进行尾数加法运算前，必须“对阶”！

浮点数加法

- 对阶：目的是使两个操作数的阶码相等(对齐小数点)
  - 规则：小阶向大阶看齐，阶小的数的尾数左移，移位数等于两个阶码差的绝对值。





IEEE754尾数右移时，需要注意的是什么？

1、要将隐含的“1”移到小数部分，空出位补0；

2、移出的低位保留到特定的“附加位”上

浮点数加法

- 两个规格化浮点数分别为A和B
  - $A=M_a \cdot 2^{Ea}$，　$B=M_b \cdot 2^{Eb}$
  - 假设$Ea \geq Eb$

- 浮点数加法步骤
  - 求阶差　　$Ea-Eb$
  - 对阶　　$M_b \cdot 2^{-(Ea-Eb)}$
  - 尾数相加　　$M_a + M_b \cdot 2^{-(Ea-Eb)}$
  - 结果规格化　$A+B = (M_a + M_b \cdot 2^{-(Ea-Eb)}) \cdot 2^{Ea}$

浮点数加法

- 两个规格化浮点数分别为A和B
  - $A=M_a \cdot 2^{Ea}$，　$B=M_b \cdot 2^{Eb}$
  - 假设$Ea \geq Eb$

$$A \pm B = (M_a \pm M_b \cdot 2^{-(Ea-Eb)}) \cdot 2^{Ea}$$

$$A \times B = (M_a \times M_b) \cdot 2^{Ea+Eb}$$

$$A \div B = (M_a \div M_b) \cdot 2^{Ea-Eb}$$

浮点数加法

自然世界中的浮点数

计算机世界中的浮点数

无穷位

超出表示范围：
溢出！

有限位

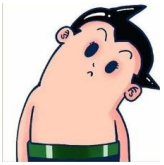
浮点数加法

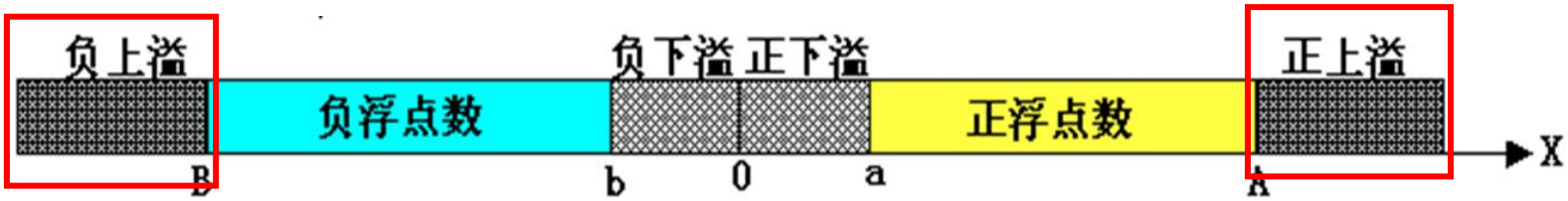
溢 出

- 阶码上溢
- 阶码下溢
- 尾数上溢
- 尾数下溢

超过了阶码可能表示的最大允许值,一般将其认为是 + ∞和 – ∞

SP允许的最大阶码为多少？？**127**！





A—最大正数　　a—最小正数
B—最小负数　　b—最大负数

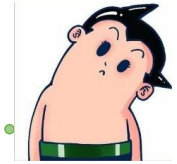
浮点数加法

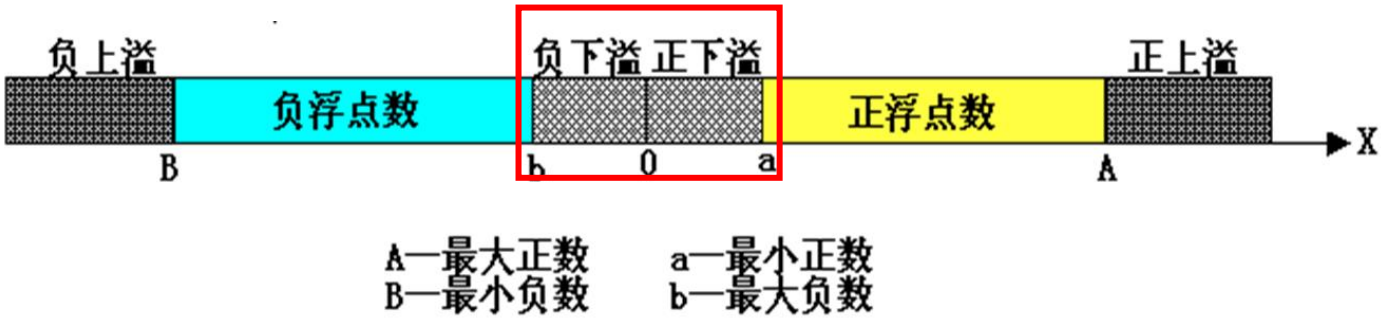
溢 出

- 阶码上溢
- 阶码下溢
- 尾数上溢
- 尾数下溢

超过了阶码可能表示的最小允许值，一般将其认为是0

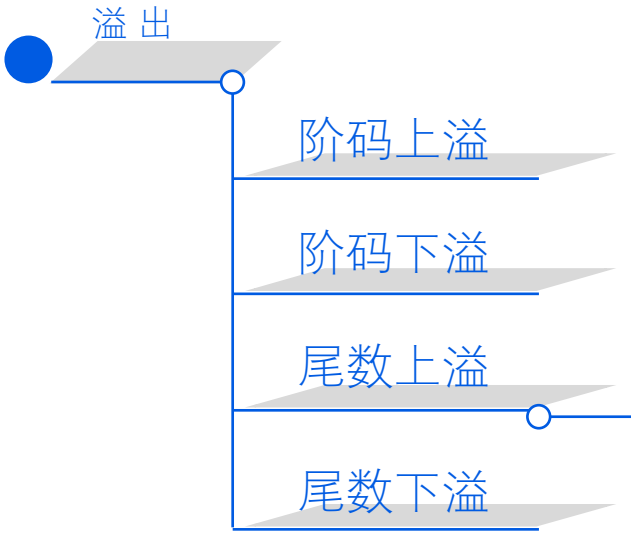
SP允许的最小阶码为多少？？ **–126！**





A—最大正数　　a—最小正数
B—最小负数　　b—最大负数

浮点数加法



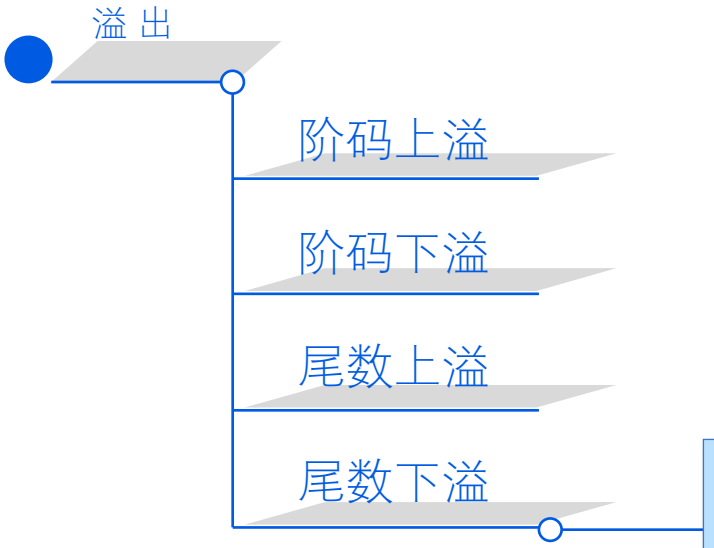
两个同/异符号尾数相加/减，
最高有效位产生了进位，

$1.010\cdots0_2\times2^2$
$+\ 1.100\cdots0_2\times2^2$
$=\ 10.110\cdots0_2\times2^2$
$=>1.0110\cdots0_2\times2^3$

尾数溢出：

- 不一定浮点数溢出，即不一定会发生“异常”
- 右归：将尾数右移，阶码增1来重新对齐

浮点数加法



在将尾数右移时，尾数的最低有效位从尾数域右端移出去，丢失了有效信息

1、进行舍入处理

2、在运算过程中，添加保护位

浮点数加法

- 浮点数加法步骤
  - 求阶差
  - 对阶
  - 尾数相加
  - 规格化并判溢出
  - 舍入
  - 置0

浮点数加法

- 浮点数加法步骤
  - 求阶差
  - 对阶
  - 尾数相加
  - 规格化
  - 舍入
  - 置0

如果尾数比规定位数长，需考虑舍入

浮点数加法

- 浮点数加法步骤
  - 求阶差
  - 对阶
  - 尾数相加
  - 规格化
  - 舍入
  - 置0

尾数为0说明结果也为0，
根据IEEE754，阶码和尾数全为0

浮点数加法

IEEE 754标准的四种舍入方式

- 就近舍入
  - 舍入为最近可表示的数

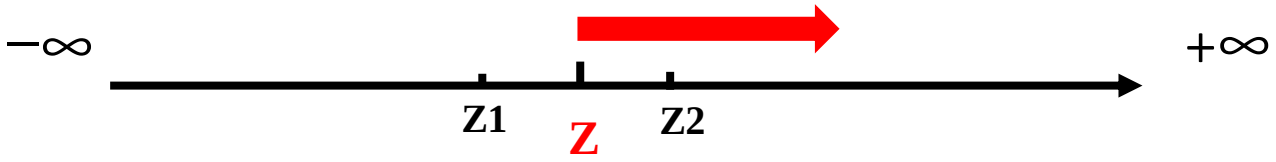
例：**1.110111 ~ 1.1110；　1.110101 ~ 1.1101；**
1.110110 ~ 1.1110；　1.111100 ~ 1.1111

- **附加位为：**
  - 11：入
  - **01：舍**
  - **10：入**
  - 00：保持结果不变

IEEE 754标准的四种舍入方式

- 就近舍入
- 朝+∞方向舍入
  - 舍入为Z2(正向舍入)

例：1.110111 ~ 1.1110；　1.110101 ~ 1.1110；
1.110110 ~ 1.1110；　1.111100 ~ 1.1111；
- 1.110110 ~ - 1.1101



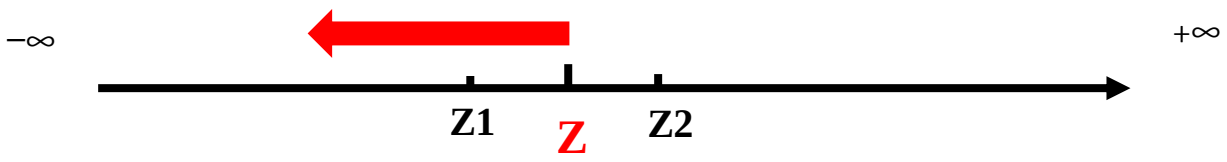
Z1和Z2分别是结果Z的最近可表示的左、右数

浮点数加法

IEEE 754标准的四种舍入方式

- 就近舍入
- 朝+∞方向舍入
- 朝-∞方向舍入
  - 舍入为Z1(负向舍入)

例： **1.110111 ~ 1.1101；** **1.110101 ~ 1.1101；**
- 1.110110 ~ - 1.1110 **- 1.110100~ - 1.1101**



Z1和Z2分别是结果Z的最近可表示的左、右数

浮点数加法

IEEE 754标准的四种舍入方式

- 就近舍入
- 朝+∞方向舍入
- 朝-∞方向舍入
- 朝0方向舍入
  - 总是舍入成Z1与Z2中绝对值较小的数(更接近于0)

例：1.110111 ~ 1.1101；　1.110101 ~ 1.1101；
　　- 1.110110 ~ - 1.1101　- 1.110100~ - 1.1101

Z1和Z2分别是结果Z的最近可表示的左、右数

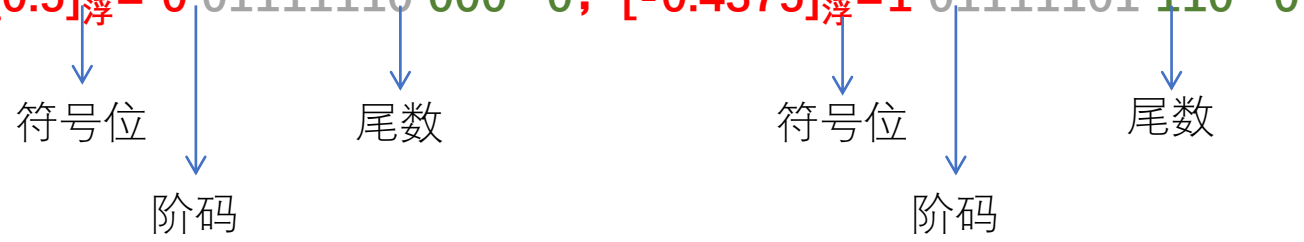
浮点数加法举例

- 用IEEE 754单精度形式，求出浮点数 $X=0.5_{10}$ 与 $Y=-0.4375_{10}$ 之和

解：$0.5_{10}=1/2_{10}=0.1_2=(1.00\cdots0_2)\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

$[0.5]_{浮}=0\ 01111110\ 000\cdots0$，$[-0.4375]_{浮}=1\ 01111101\ 110\cdots0$



浮点数加法举例

- 用IEEE 754单精度形式，求出浮点数 $X=0.5_{10}$ 与 $Y=-0.4375_{10}$ 之和

解： $0.5_{10}=1/2_{10}=0.1_2=(1.00\cdots0_2)\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

$[0.5]_{浮}$= 0 01111110 000…0，$[-0.4375]_{浮}$=1 01111101 110…0

- 对阶（求阶差）：

$$[\Delta E]_{补}=[Ex]_{移}+[-[Ey]_{移}]_{补} \pmod{2^8}$$

$[\Delta E]_{补}$=0111 1110 + 1000 0011=0000 0001，ΔE=1

所以：对Y进行对阶，$[Y]_{浮}$=1 0111 1110 1110…0

浮点数加法举例

- 用IEEE 754单精度形式，求出浮点数 $X=0.5_{10}$ 与 $Y=-0.4375_{10}$ 之和

解： $0.5_{10}=1/2_{10}=0.1_2=(1.00\cdots0_2)\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

$[0.5]_{浮}=0\ 01111110\ 000\cdots0$，$[-0.4375]_{浮}=1\ 01111101\ 110\cdots0$

- 对阶（求阶差）： $[Y]_{浮}=1\ \underline{0111\ 1110}\ \underline{1110\cdots0}$
- 尾数处理： 1.0000...0-(0.1110...0)=0.00100…0

浮点数加法举例

❑ 用IEEE 754单精度形式，求出浮点数 $X=0.5_{10}$ 与 $Y=-0.4375_{10}$ 之和

解： $0.5_{10}=1/2_{10}=0.1_2=(1.00\cdots0_2)\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

$[0.5]_{浮}=0\ 01111110\ 000\cdots0$，$[-0.4375]_{浮}=1\ 01111101\ 110\cdots0$

- 对阶（求阶差）： $[Y]_{浮}=1\ \underline{0111\ 1110}\ \underline{1110\ldots0}$
- 尾数相加： $1.0000\ldots0-(0.1110\ldots0)=0.00100\cdots0$
- 规格化和判溢出： $+(0.00100\cdots0)_2\times2^{-1}=+(1.00\cdots0)_2\times2^{-4}$ (阶码减3)

因为127≥-4 ≥ -126，没有溢出

浮点数加法举例

❑ 用IEEE 754单精度形式，求出浮点数 $X=0.5_{10}$ 与 $Y=-0.4375_{10}$ 之和

解： $0.5_{10}=1/2_{10}=0.1_2=(1.00\cdots0_2)\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

$[0.5]_{浮}=0\ 01111110\ 000\cdots0$，$[-0.4375]_{浮}=1\ 01111101\ 110\cdots0$

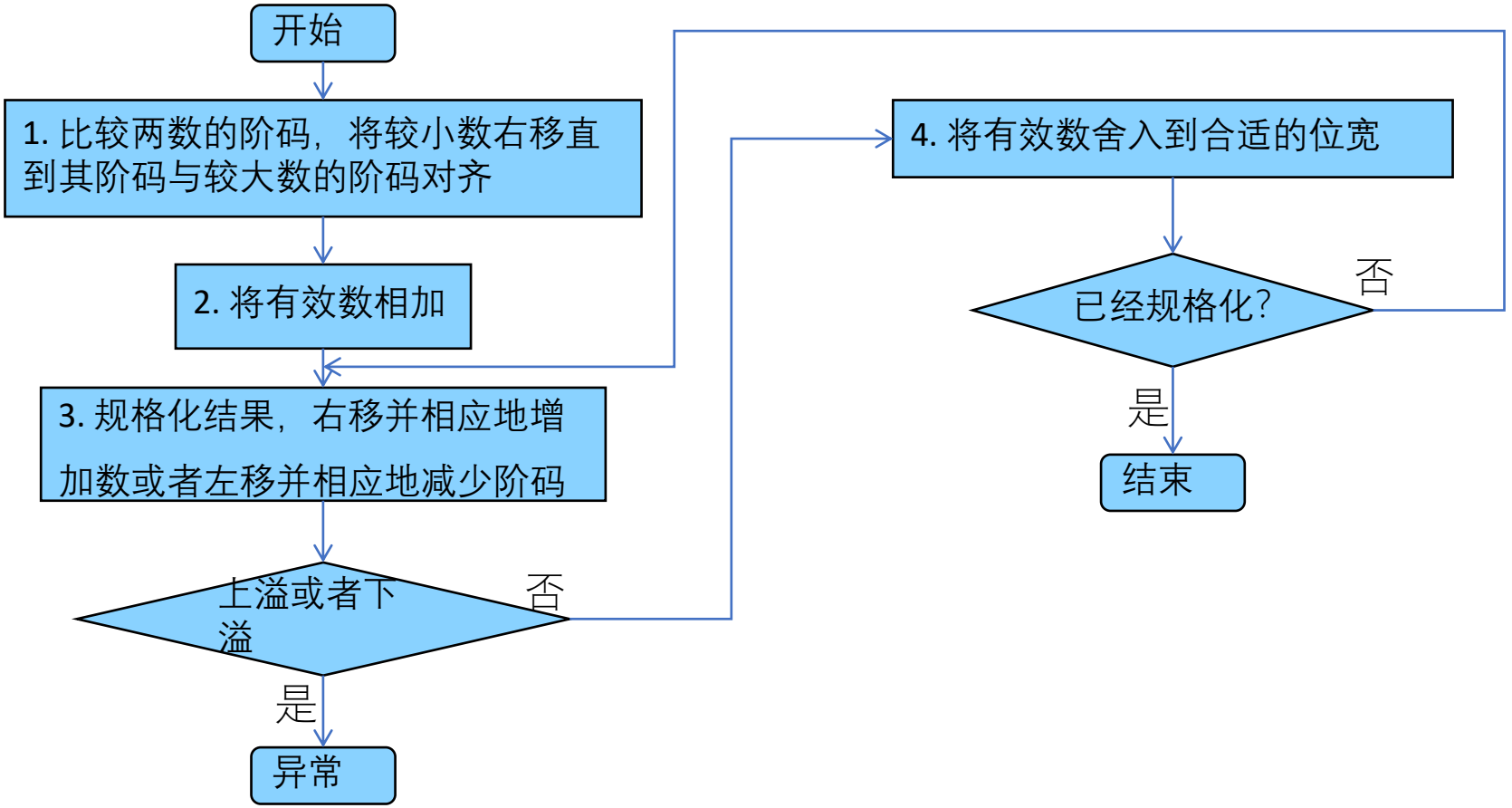
- 对阶（求阶差）： $[Y]_{浮}=1\ \underline{0111\ 1110}\ \underline{1110\ldots0}$
- 尾数相加： $01.0000\ldots0+(10.1110\ldots0)=00.00100\cdots0$
- 规格化和判溢出： $+(0.00100\cdots0)_2\times2^{-1}=+(1.00\cdots0)_2\times2^{-4}$ (阶码减3)

因为127≥-4 ≥ -126，没有溢出

- 舍入： 无需舍入

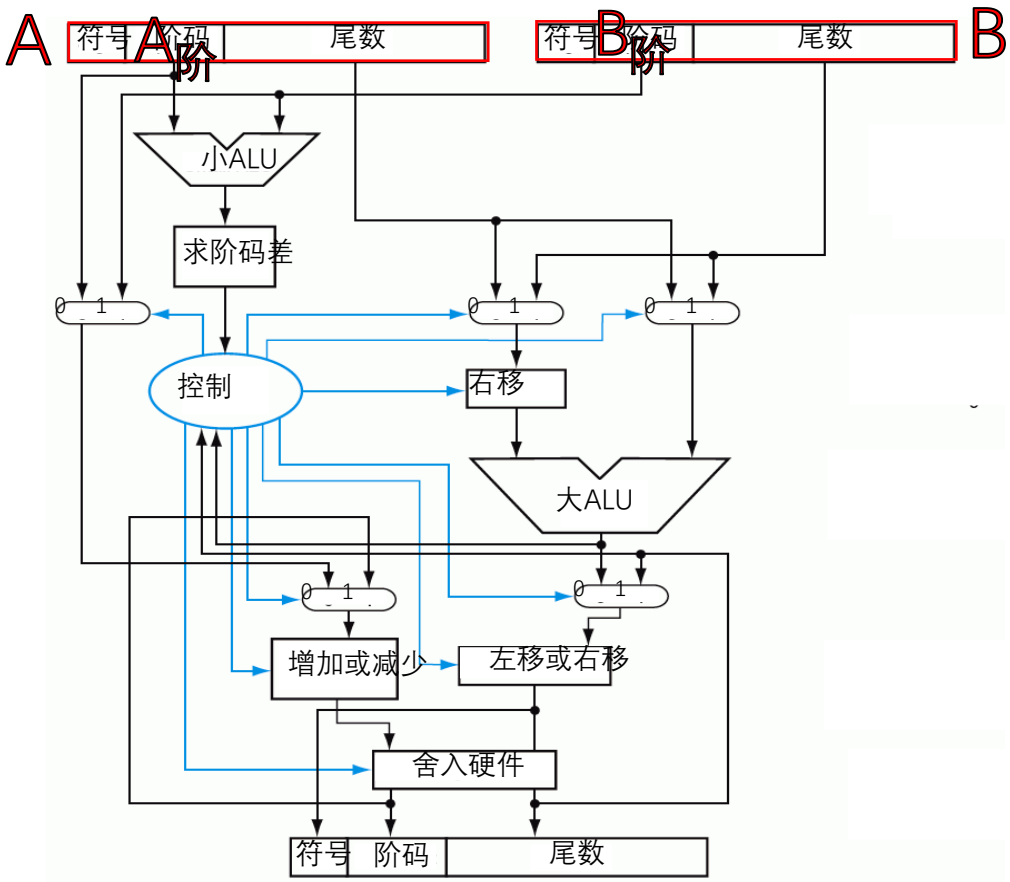
结果为：$(1.00\cdots0)_2\times2^{-4}$ 即 1/16=0.0625

浮点数加法算法


开始
1. 比较两数的阶码，将较小数右移直到其阶码与较大数的阶码对齐
2. 将有效数相加
3. 规格化结果，右移并相应地增加数或者左移并相应地减少阶码
上溢或者下溢
否
是
异常
4. 将有效数舍入到合适的位宽
已经规格化?
否
是
结束

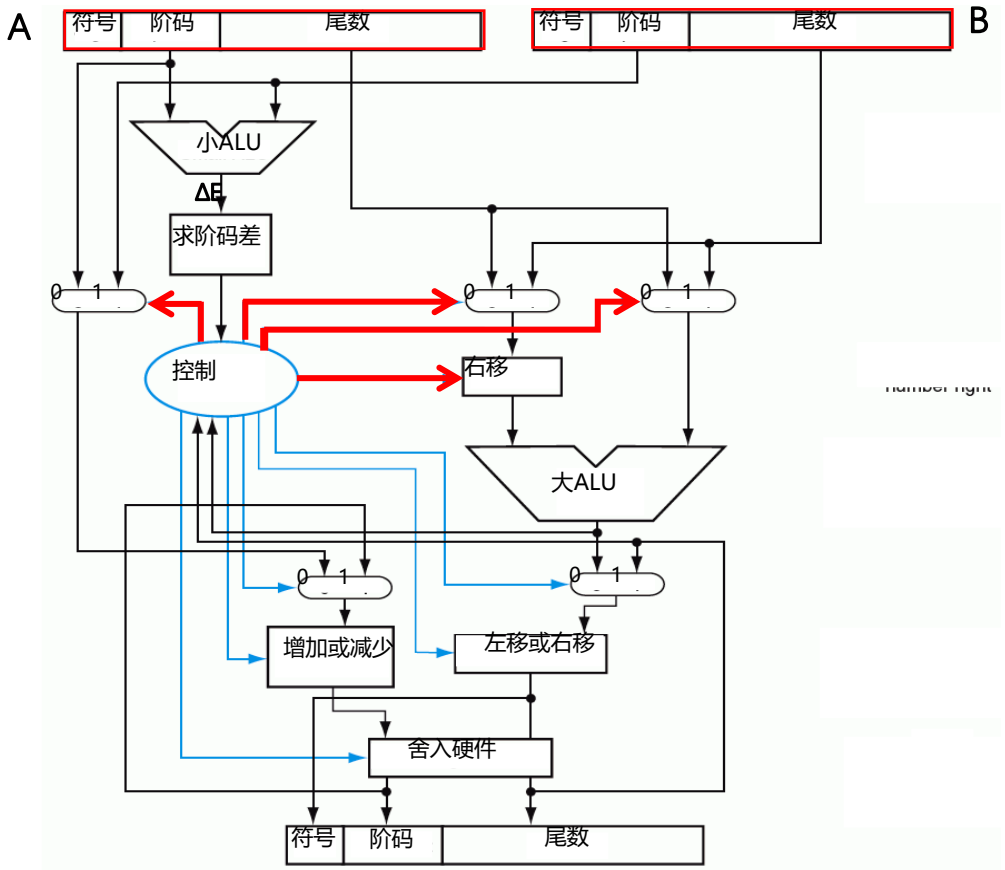
浮点数加法运算的硬件逻辑结构图

比较阶码



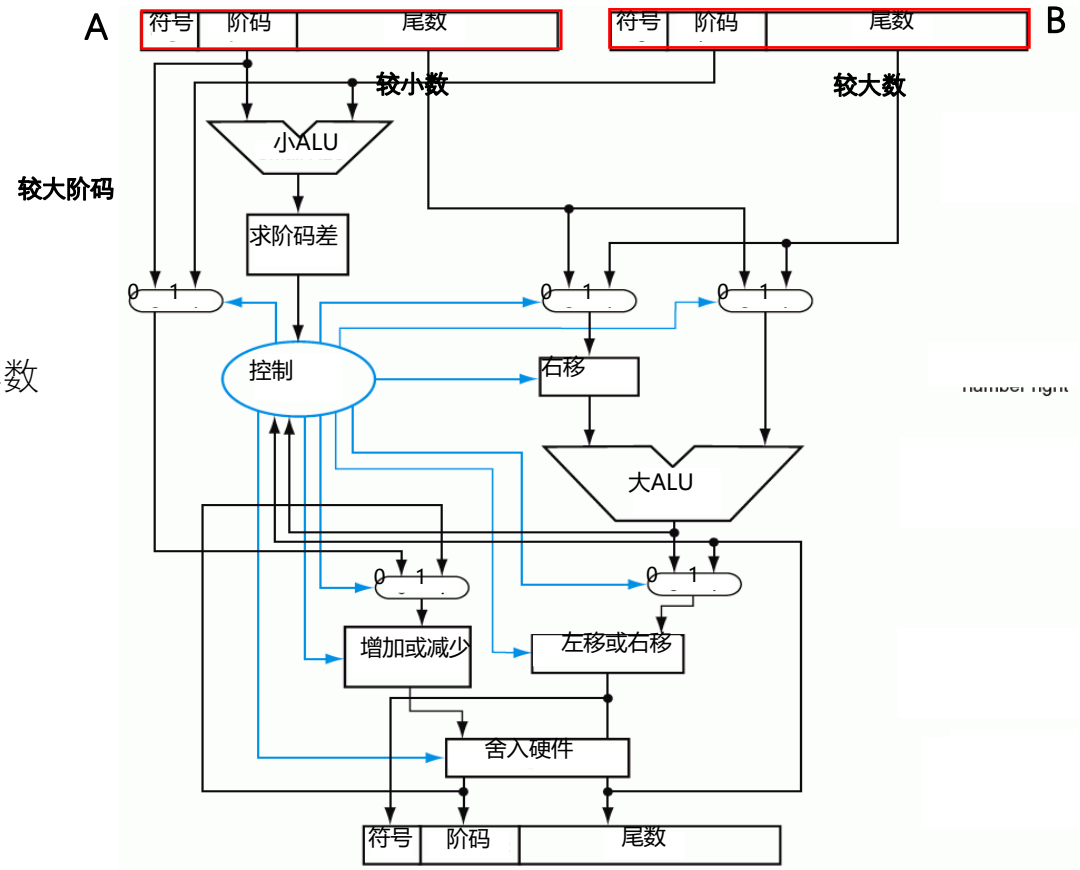
浮点数加法运算的硬件逻辑结构图

右移较小数



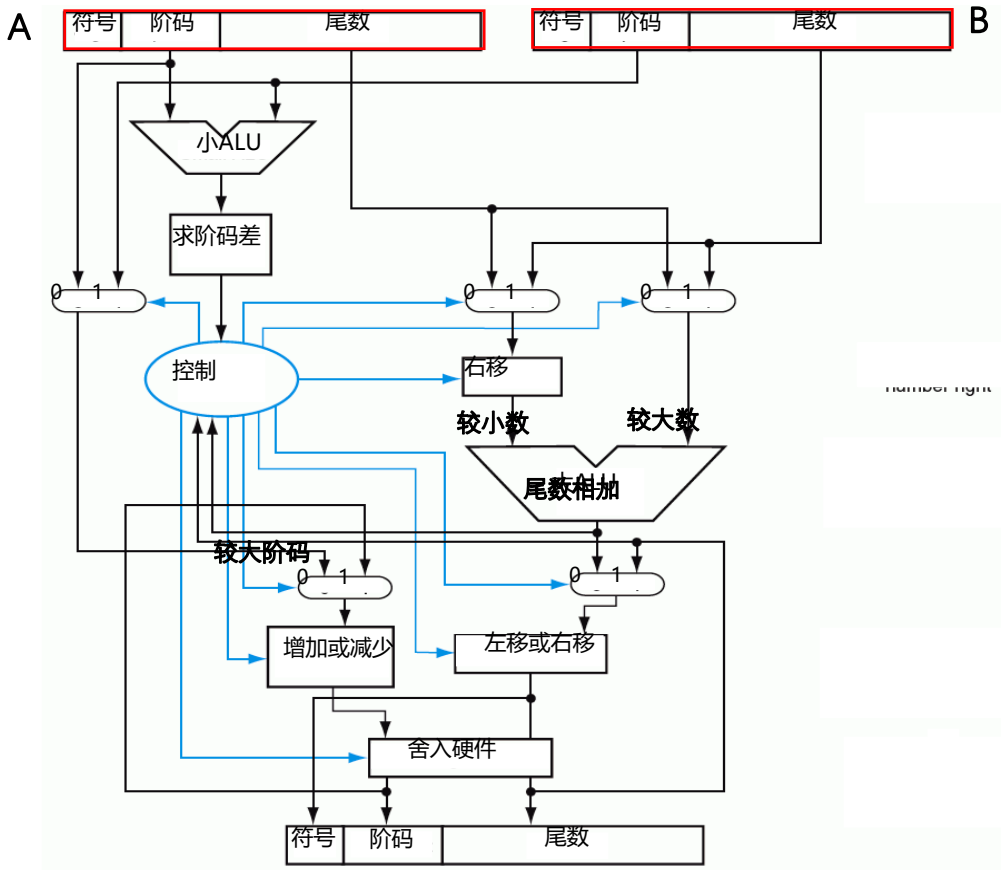
浮点数加法运算的硬件逻辑结构图

右移较小数

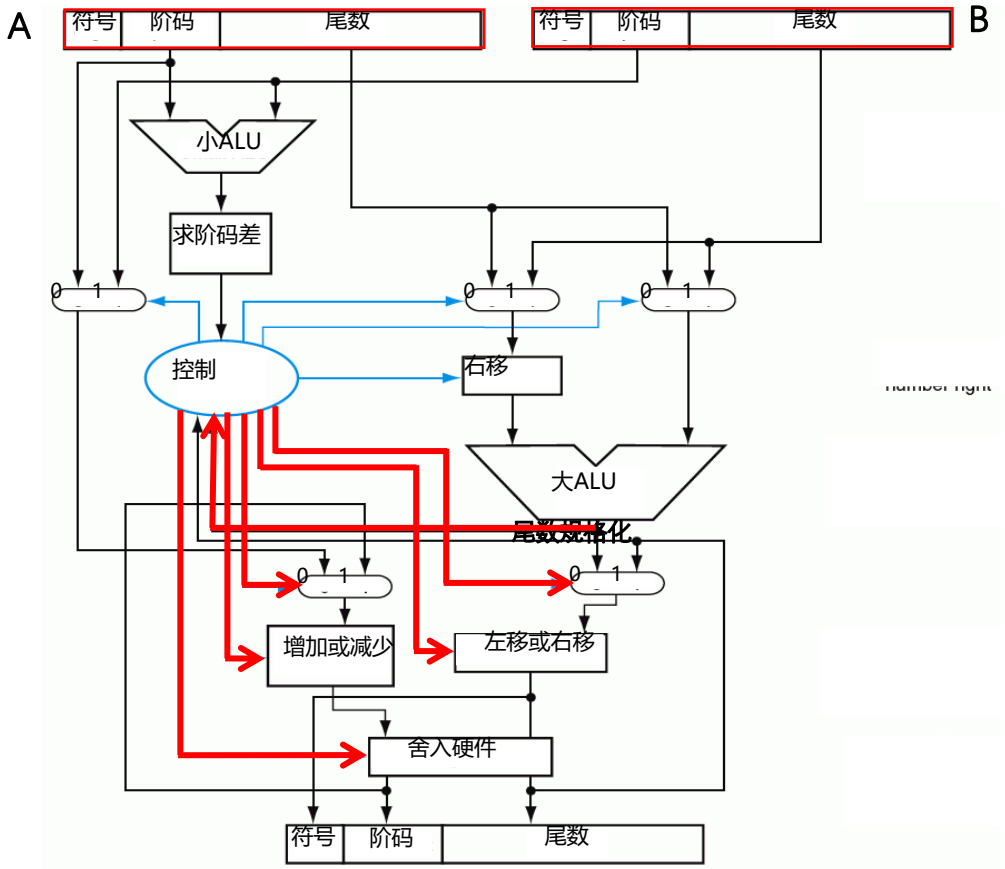


浮点数加法运算的硬件逻辑结构图

尾数相加

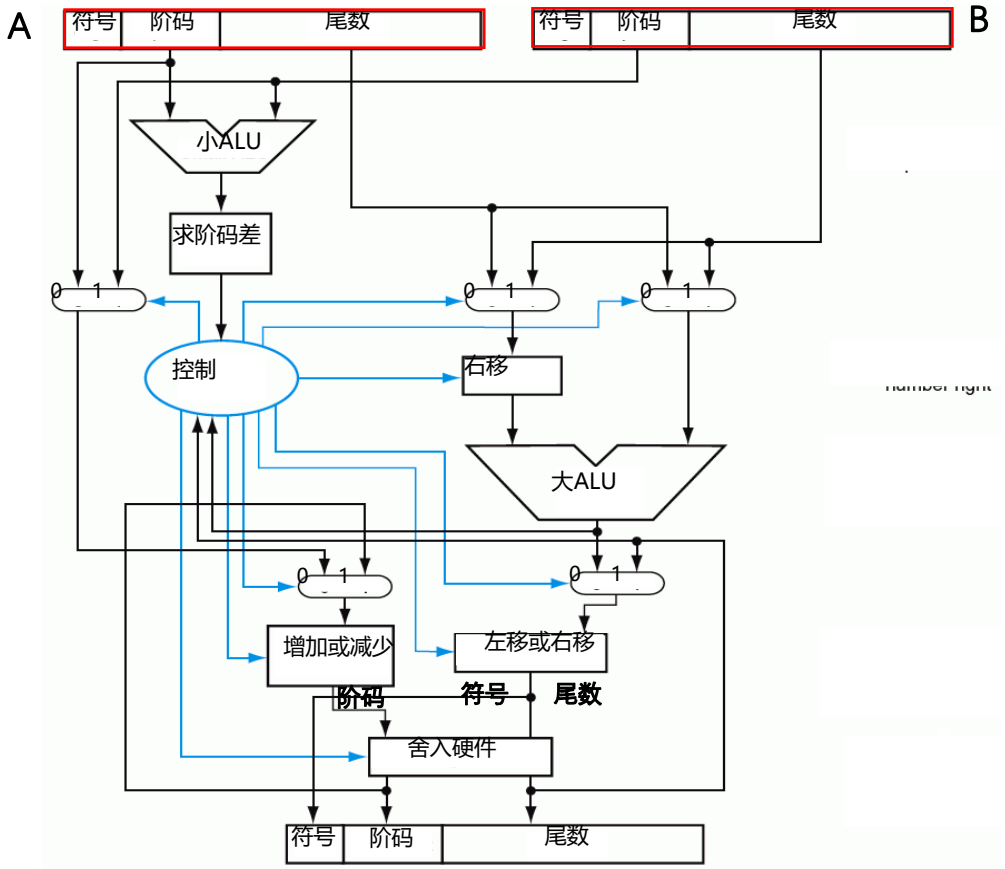


浮点数加法运算的硬件逻辑结构图



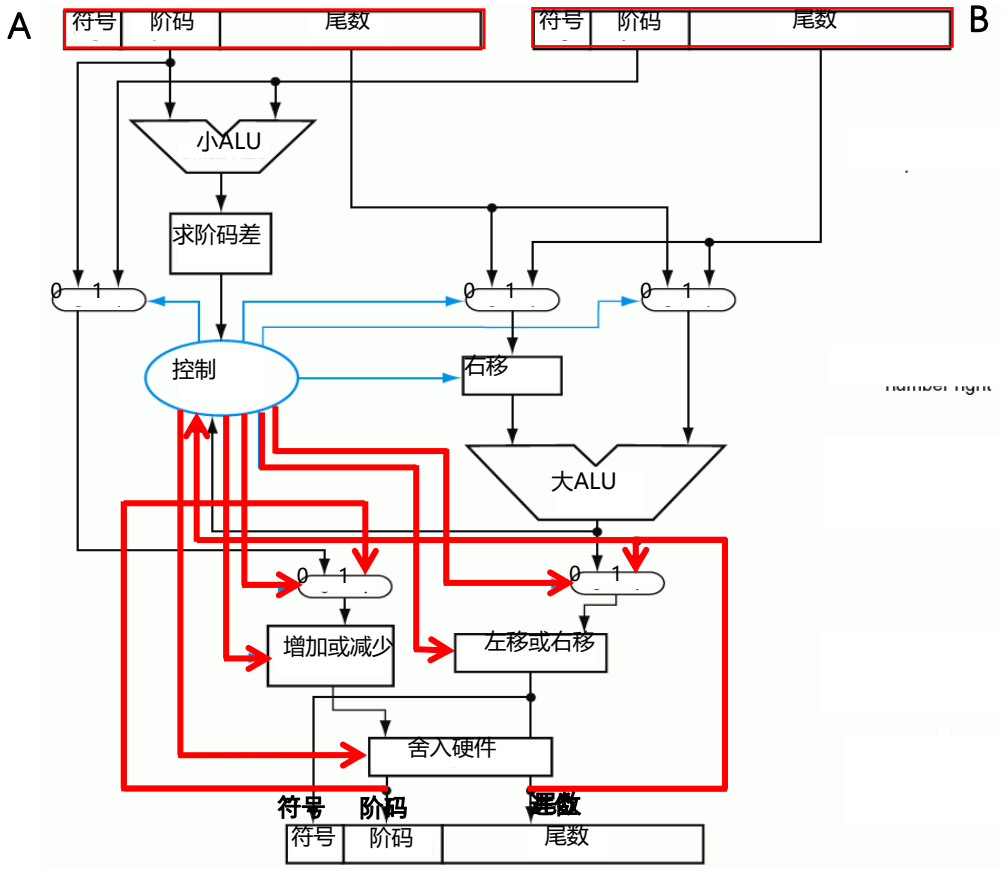
规格化

浮点数加法运算的硬件逻辑结构图



舍入

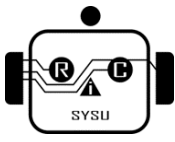
浮点数加法运算的硬件逻辑结构图



舍入

本讲内容

- 基本运算
  - 加法和减法
  - 位操作
- 乘法运算
- 除法运算
- 浮点运算
  - 基本概念和问题
  - 浮点加减法
  - 浮点乘除法
  - 运算精度问题



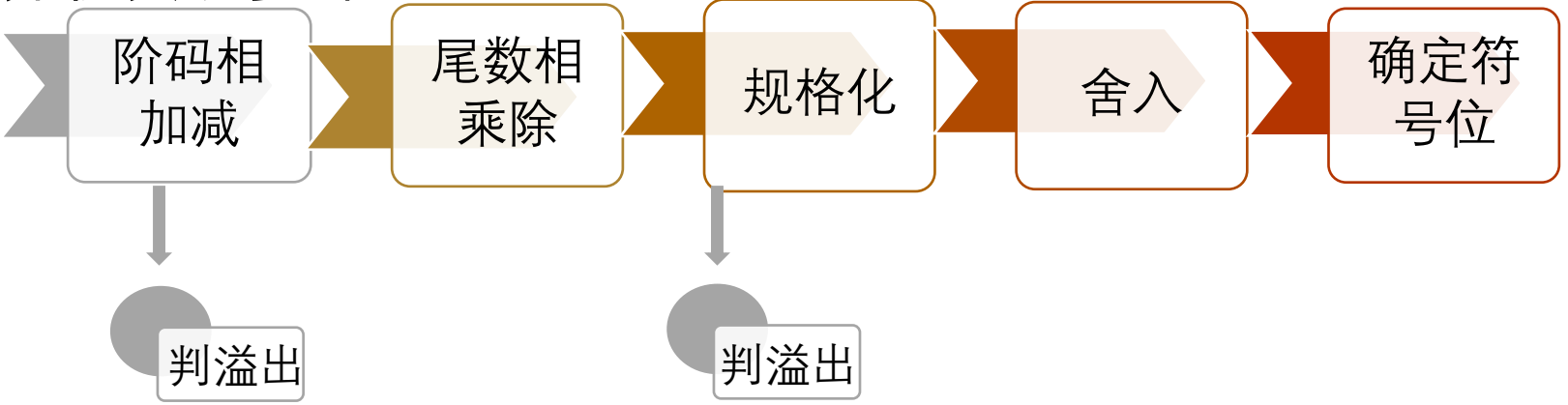
浮点数乘除

- 两个浮点数：$A=M_a \cdot 2^{Ea}$，$B=M_b \cdot 2^{Eb}$

$$A \times B = (M_a \times M_b) \cdot 2^{Ea+Eb}$$

$$A \div B = (M_a \div M_b) \cdot 2^{Ea-Eb}$$

- 计算机实现步骤：



1. 阶码运算

❑ 阶码相加（单精度浮点乘法）

$$[E_1 + E_2]_{移} = f([E_1]_{移}, [E_2]_{移}) = ([E_1]_{移} + [E_2]_{移} + 129) \bmod 2^8$$

$$[E_1 + E_2]_{移} = 127 + E_1 + E_2$$

$$= 127 + E_1 + 127 + E_2 - 127$$

$$= [E_1]_{移} + [E_2]_{移} - 127$$

$$= [E_1]_{移} + [E_2]_{移} + [-127]_{补}$$

$$= ([E_1]_{移} + [E_2]_{移} + 10000001B) \bmod 2^8$$

❑ 阶码相加（单精度浮点乘法）

例：若两个IEEE754操作数的阶码分别为10和－5，求10 + (－5) 的移码

解：

$[E_1]_{移} = 127+10 =137=1000\ 1001B$

$[E_2]_{移} = 127+(–5)=122= 01111010B$

$[E_1+ E_2]_{移} = [E_1]_{移} + [E_2]_{移} +129$

$= 1000\ 1001 + 0111\ 1010 + 1000\ 0001 (\mathrm{mod}\ 2^8)$

$= 1000\ 0100B = 132$

其阶码的和为132–127 = 5，正好等于10 + (–5) = 5

1. 阶码运算

❑ 阶码相减（单精度浮点除法）

$$[E_1 - E_2]_{移} = f([E_1]_{移}, [E_2]_{移}) = ([E_1]_{移} + [-[E_2]_{移}]_{补} + 127) \bmod 2^8$$

$$\Downarrow$$

$$[E_1 - E_2]_{移} = E_1 - E_2 + 127$$

$$= 127 + E_1 - (127 + E_2) + 127$$

$$= [E_1]_{移} - [E_2]_{移} + 127$$

$$= ([E_1]_{移} + [-[E_2]_{移}]_{补} + 01111111B) \bmod 2^8$$

1. 阶码运算

❑ 阶码相减（单精度浮点除法）

例：若两个IEEE754操作数的阶码分别为10和－5，求10-(－5)的移码

解：$[E_1]_{移} = 127+10 =137=1000\ 1001B$

$[E_2]_{移} = 127+(–5)=122= 01111010B$

$[E_1+ E_2]_{移} = [E_1]_{移} + [-[E_2]_{移}]_{补}+127$

$= 1000\ 1001 + \ 1000\ 0110 + 0111\ 1111(\text{mod}\ 2^8)$

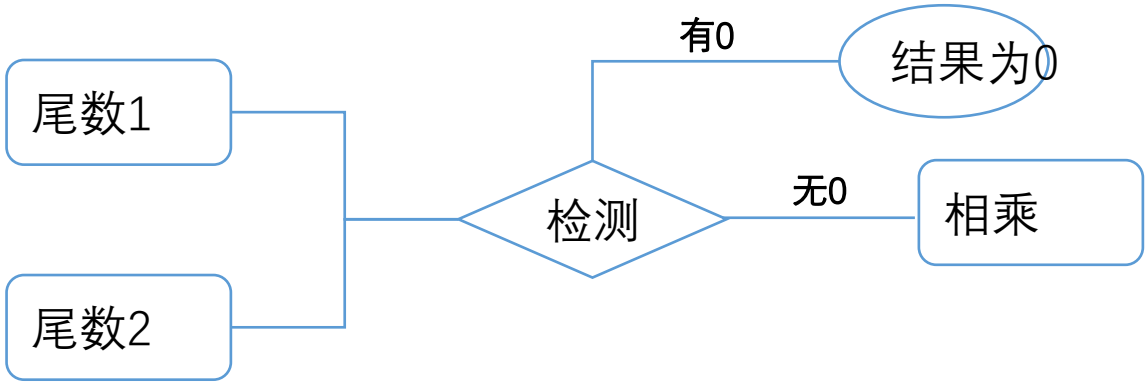
$= 1000\ 1110B \ = 142$

其阶码的差为142–127 = 15，正好等于10 - (–5) = 15

2. 尾数运算

❑ 尾数相乘

■ 预处理



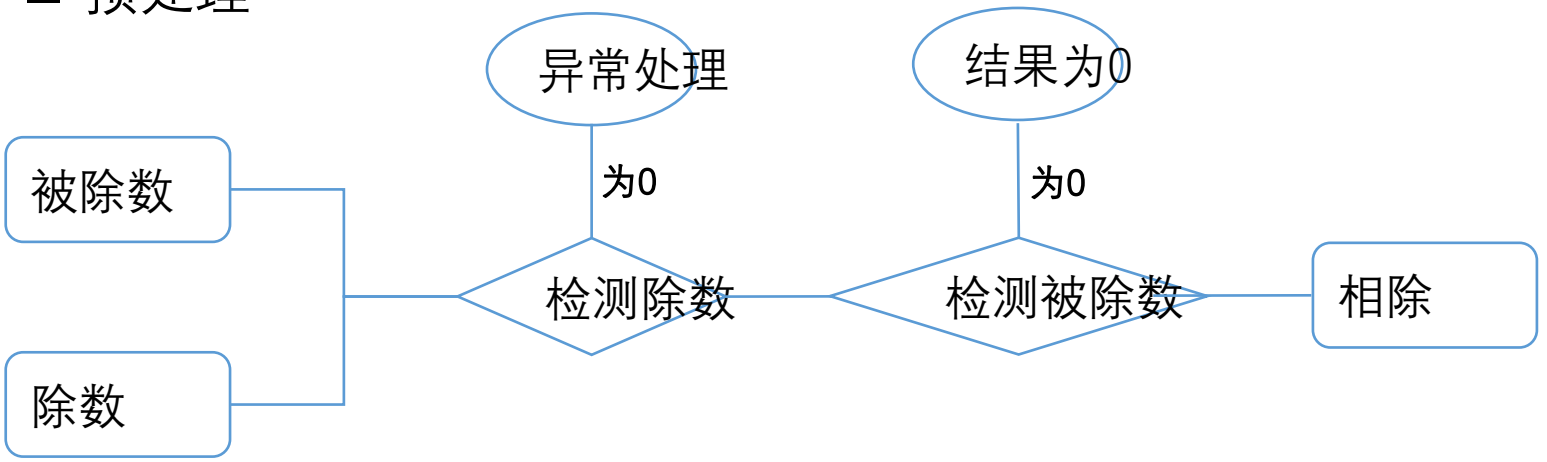
■ 相乘

❑定点小数原码乘法运算

2. 尾数运算

❑ 尾数相除

■ 预处理



■ 相除

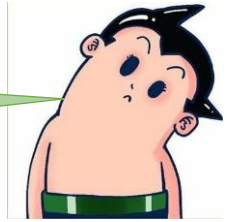
- ❑ 定点小数原码除法运算

3. 规格化

❑ 规格化原则

- ■ 当尾数高位为0，左规
- ■ 当尾数产生进位，右规

乘法运算结果最多左规几次、右规几次？除法呢？



规格化尾数：$1 \leqslant$ 尾数$_1 < 2$，$1 \leqslant$ 尾数$_2 < 2$ （尾数形为1.xxx）

乘法：$1 \leqslant$ 尾数$_1 \times$ 尾数$_2 < 4$　　不需左规、最多右规一次

除法：$0.5 <$ 尾数$_1 \div$ 尾数$_2 < 2$　　不需右规、最多左规一次

4. 舍入并确定符号

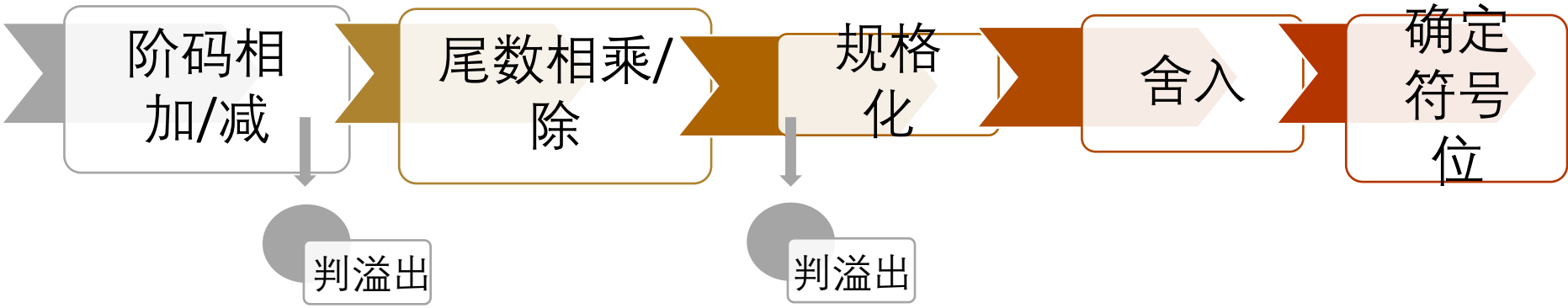
- 舍入
  - 就近舍入
  - 朝+∞舍入
  - 朝—∞舍入
  - 朝0舍入
- 符号确定
  - 同号：结果为正
  - 异号：结果为负

就近舍入

若尾数是0，则需要将阶码也置0

5. 判溢出

- 阶码溢出情况



5. 判溢出

❑ 阶码溢出情况

阶码求和

$E_x + E_y + 129 = E_b$

阶码求差

1xxxx	1xxxx	0xxxx	上溢
0xxxx	0xxxx	1xxxx	下溢

规格化

11111	上溢
00000	若无法规格化，异常

5. 判溢出

❑ 阶码溢出情况

阶码求和

阶码求差

规格化

$E_x - E_y + 127 = E_b$

1xxxx	0xxxx	0xxxx	上溢
0xxxx	1xxxx	1xxxx	下溢
		11111	上溢
		00000	若无法规格化，异常

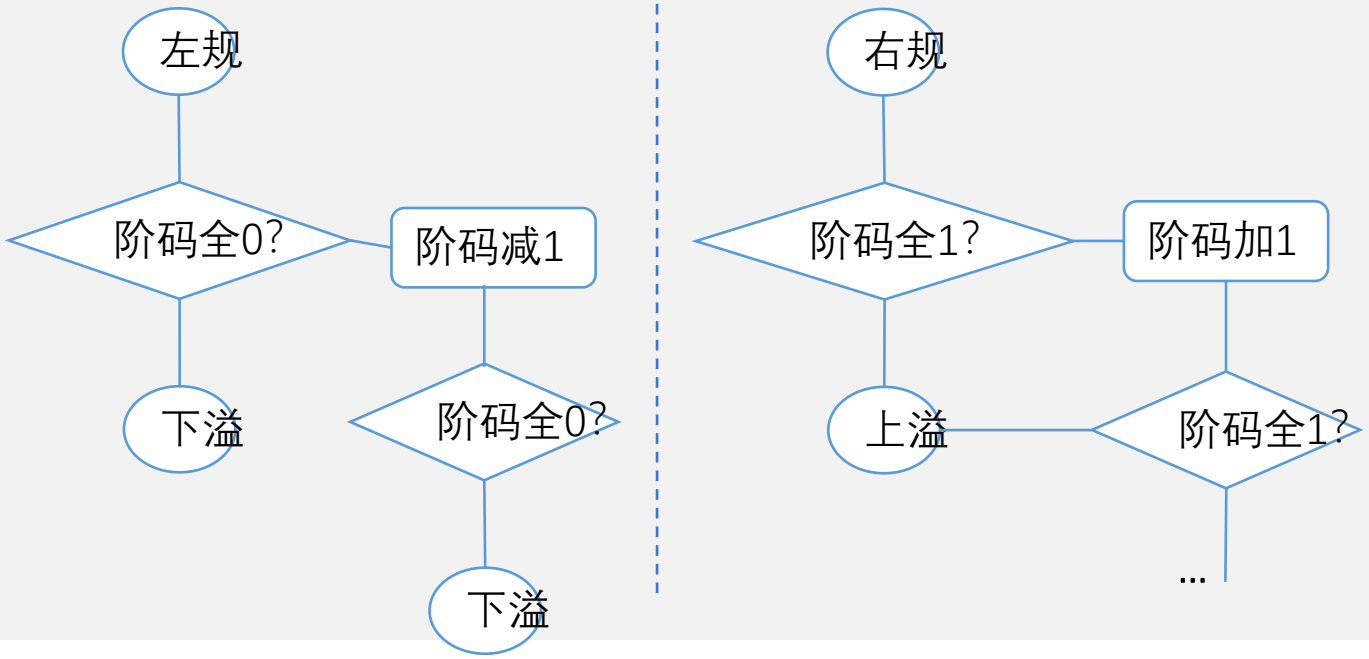
5. 判溢出

❑ 阶码溢出情况

阶码求和

阶码求差

规格化



溢出判断举例

- 例1：若Eb=0000 0001，则尾数左规一次后，结果的阶码Eb=?

解：$Eb = Eb+[-1]_{补} = 0000\ 0001 + 1111\ 1111$

$= 0000\ 0000$　阶码下溢！

- 例2：若Ex=1111 1110，Ey=1000 0000，则乘法运算时，结果的阶码Eb=?

解：$Eb = Ex+Ey+129 = 1111\ 1110 + 1000\ 0000 + 1000\ 0001$

$= 1111\ 1111$　阶码上溢！

浮点数乘法实例

例：用二进制的形式求出浮点数 0.5_{10}与 -0.4375_{10}之积(假设保留4位有效数位的精度)。

解：$0.5_{10} = 1/2_{10} = 0.1_2 = 1.000_2 \times 2^{-1}$

$-0.4375_{10} = -7/16_{10} = -0.0111_2 = -1.110_2 \times 2^{-2}$

■阶码相加：$-1 + (-2) = -3$

■尾数相乘：

$1.000_2 \times 1.110_2 = 1.110000_2$

乘积为：$1.110000_2 \times 2^{-3}$

$$
\begin{array}{r}
1.000_{\text{two}} \\
\times\ 1.110_{\text{two}} \\
\hline
0000 \\
1000 \\
1000 \\
1000 \\
\hline
1110000_{\text{two}}
\end{array}
$$

浮点数乘法实例

例：用二进制的形式求出浮点数0.5_{10}与-0.4375_{10}之积(假设保留4位有效数位的精度)。

解：$0.5_{10} = 1/2_{10} = 0.1_2 = 1.000_2 \times 2^{-1}$

$-0.4375_{10} = -7/16_{10} = -0.0111_2 = -1.110_2 \times 2^{-2}$

■阶码相加：$-1 + (-2) = -3$

■尾数相乘：

$1.000_2 \times 1.110_2 = 1.110000_2$

乘积为：$1.110000_2 \times 2^{-3}$

■规格化并判溢出：

尾数1.110000已是规格化数，且127≥-3≥-126，没有溢出！

■舍入：$1.110_2 \times 2^{-3}$

■确定符号位：

乘积为负数，$-1.110_2 \times 2^{-3}$

所以浮点乘法结果：

$-1.110_2 \times 2^{-3} = -0.00111_2$

$= -7/32_{10}$

$= -0.21875_{10}$

浮点数除法实例

例：用二进制的形式求出浮点数 0.5_{10}与-0.4375_{10}的商(假设保留4位有效数位的精度)。

解：$0.5_{10}=1/2_{10}=0.1_2=1.000_2\times2^{-1}$

$-0.4375_{10}=-7/16_{10}=-0.0111_2=-1.110_2\times2^{-2}$

■阶码相减：$-1-(-2)=1$

■尾数相除：

$1.000_2\div1.110_2=0.1001001_2$

商为：$0.1001001_2\times2^1$

■规格化并判溢出：

规格化：$1.0010010_2\times2^0$

且127≥0≥-126，没有溢出

■舍入：$1.001_2\times2^0$

■确定符号位：

商为负数，$-1.001_2\times2^0$

所以浮点除法结果：

$-1.001_2\times2^0=-1.001_2$

$=-9/8_{10}$

$=-1.125_{10}$

浮点数运算小结

- 浮点数运算：由多个ALU + 移位器实现
  - 加/减运算
    - 对阶、尾数相加减、规格化处理、舍入、判断溢出
  - 乘/除运算
    - 尾数用定点原码乘/除运算实现，阶码用定点数加/减运算实现
  - 溢出判断
    - 当结果发生阶码上溢时，结果发生溢出；发生阶码下溢时，结果为0
  - 精确表示运算结果
    - 中间结果增设保护位、舍入位，等
    - 最终结果四种舍入方式：就近舍入 / 正向舍入 / 负向舍入 / 截去

MIPS的浮点指令

- MIPS支持IEEE 754标准定义的浮点数
  - 单精度浮点数的加法指令(add.s)
  - 双精度浮点数的加法指令(add.d)
  - 单精度浮点数的减法指令(sub.s)
  - 双精度浮点数的减法指令(sub.d)
  - 单精度浮点数的乘法指令(mul.s)
  - 双精度浮点数的乘法指令(mul.d)

- 单精度浮点数的除法指令(div.s)
- 双精度浮点数的除法指令(div.d)
- 单精度浮点数之间的比较指令(c.x.s)
- 双精度浮点数之间的比较指令(c.x.d)
- 浮点数分支指令：为真时分支bclt；为假时分支bclf

MIPS的浮点指令

- MIPS设置了32个独立的浮点寄存器(f_0, f_1, …f_{31})
- 提供两条专门用于浮点数的存储器取存(load/store)指令lwc1和swc1

例：将两个单精度的浮点数从存储器取出装入到浮点寄存器中，然后将两数相加，最后再将结果存回到存储器中。

```
lwc1  $f4,  x($sp)      #Load 32-bit F.P. number into F4
lwc1  $f6,  y($sp)      #Load 32-bit F.P. number into F6
add.s  $f2,  $f4,  $f6    #F2=F4+F6 single precision
swc1   $f2,  z($sp)     #Store 32-bit F.P. number from F2
```

MIPS浮点指令的机器语言

MIPS floating-point machine language

Name	Format	Example						Comments
add.s	R	17	16	6	4	2	0	add.s $f2,$f4,$f6
sub.s	R	17	16	6	4	2	1	sub.s $f2,$f4,$f6
mul.s	R	17	16	6	4	2	2	mul.s $f2,$f4,$f6
div.s	R	17	16	6	4	2	3	div.s $f2,$f4,$f6
add.d	R	17	17	6	4	2	0	add.d $f2,$f4,$f6
sub.d	R	17	17	6	4	2	1	sub.d $f2,$f4,$f6
mul.d	R	17	17	6	4	2	2	mul.d $f2,$f4,$f6
div.d	R	17	17	6	4	2	3	div.d $f2,$f4,$f6
lwc1	I	49	20	2	100			lwc1 $f2,100($s4)
swc1	I	57	20	2	100			swc1 $f2,100($s4)
bc1t	I	17	8	1	25			bc1t 25
bc1f	I	17	8	0	25			bc1f 25
c.lt.s	R	17	16	4	2	0	60	c.lt.s $f2,$f4
c.lt.d	R	17	17	4	2	0	60	c.lt.d $f2,$f4
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits

MIPS的浮点指令应用举例

- 将包含浮点运算的C语言程序编译成MIPS汇编码

例：
```
float  f2c(float  fahr)
       {
       return  ((5.0/9.0)*(fahr －32.0));
       }
```

解：
```
lwc1   $f16, const5($gp)      #f16=5.0(5.0 in memory)
lwc1   $f18, const9($gp)      #  f18=9.0(9.0 in memory)
div.s  $f16, $f16, $f18       #f16=5.0/9.0
lwc1   $f18, const32($gp)     #f18=32.0
sub.s  $f18, $f12, $f18       #f18=fahr － 32.0
mul.s $f0, $f16, $f18         #f0=(5.0/9.0)*(fahr－32.0)
jr           $ra                    #return
```

不同数据类型采用的MIPS指令

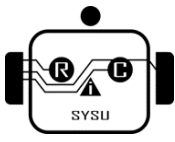
C type	Java type	Data transfers	Operations
int	int	lw, sw, lui	addu, addiu, subu, mult, div, and, andi, or, ori, nor, slt, slti
unsigned int	—	lw, sw, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
char	—	lb, sb, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
—	char	lh, sh, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
float	float	lwc1, swc1	add.s, sub.s, mult.s, div.s, c.eq.s, c.lt.s, c.le.s
double	double	l.d, s.d	add.d, sub.d, mult.d, div.d, c.eq.d, c.lt.d, c.le.d

实例：PowerPC中的浮点部件

- PowerPC中的浮点运算
  - 比MIPS多一条浮点指令：乘加指令
    - 将两个操作数相乘，再与另一个操作数相加，写到结果操作数
    - 可以用一条乘加指令代替两条MIPS浮点指令
    - 可为中间结果多保留几位，得到最后结果后，再考虑舍入，精度高
    - 利用它来实现除法运算和平方根运算
  - 浮点寄存器的数量多一倍(32×SPR, 32×DPR)

实例：80x86中的浮点部件

- 80x86中的浮点运算
  - 采用寄存器堆栈结构：栈顶两个数作为操作数
  - 寄存器堆栈的精度为80位(MIPS和PowerPC最多都是64位)
  - 浮点运算都转换为80位扩展浮点数进行运算，写回存储器时，再转换位32位(float)或64位(double)。有时会有问题
  - 浮点数据存取指令自动完成转换
  - 指令类型：存取、算术、比较、函数(正弦、余弦、对数等)

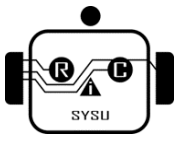


浮点运算小结

- 浮点数运算：由多个ALU + 移位器实现
  - 加/减运算
    - 对阶 、尾数相加减、规格化处理、舍入、判断溢出
  - 乘/除运算
    - 尾数用定点原码乘/除运算实现，阶码用定点数加/减运算实现
  - 溢出判断
    - 当结果发生阶码上溢时，结果发生溢出，发生阶码下溢时，结果为0
  - 精确表示运算结果
    - 中间结果增设保护位、舍入位、粘位
    - 最终结果舍入方式：就近舍入 / 正向舍入 / 负向舍入 / 截去四种方式

本讲内容

- 基本运算
  - 加法和减法
  - 位操作
- 乘法运算
- 除法运算
- 浮点运算
  - 基本概念和问题
  - 浮点加减法
  - 浮点乘除法
  - 运算精度问题



精确的算术运算

- 整数的精确表示问题
  - 一定字长所能表示的整数范围是有限的
  - 任何一个整数只要落在所能表示的最大值和最小值之间，就可以精确表示
- 浮点数的精确表示问题
  - 有些浮点数是无法用有限的字长精确地表示出来，一般只能用近似值代替（舍入）
  - 在任何两个实数(如0~1)之间都有无穷多个实数，对于单精度的浮点数的尾数所能表示的实数的总个数只有区区的2^{24}个

这是与现实世界中数字的最大不同，程序员必须时刻记住这一限制条件，在编程时就可以根据实际情况做一些必要的特殊处理

谬误和陷阱

- 自然界的算术运算是没有精度限制的
- 在计算机中由于机器字长的限制，所有的算术运算都被限制在一定的精度范围内。

谬误1：浮点加法满足结合律，即x+(y+z)=(x+y)+z

例：$x = -1.5_{ten} \times 10^{38}$，$y = 1.5_{ten} \times 10^{38}$，而z=1.0(且三个数都是单精度)，则有：

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

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

因此，浮点加法不满足结合定律!

谬误和陷阱

谬误2：对于整数，左移指令相当于将原数与2的幂次方相乘；右移指令则相当于将原来的整数除以2的幂次方。

例：若将-5_{ten}除以4_{ten}，正确的商应该是-1_{ten}。

-5_{ten}用二进制补码表示为：

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{two}$

如果根据上述错误观点，除以$4_{ten}(2^2)$变成逻辑右移2位，其结果为：

$0011\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$

现在连符号位都变成了0，这样一个结果显然是错误的。右移得到的结果实际上是1073741822_{ten}，而不是-1_{ten}。

三种移位器

逻辑(*logical*) — 移入的数值总是 "0"

算术(*arithmetic*) — 在右移时，进行符号扩展

循环(*rotating*) — 移出的位又从另一端移入 (在MIPS中不支持)

注：需要一位移动。特定指令可能要求进行0⇒32位移动！

谬误和陷阱

谬误2：对于整数，左移指令相当于将原数与2的幂次方相乘；右移指令则相当于将原来的整数除以2的幂次方。

例：若将-5_{ten}除以4_{ten}，正确的商应该是-1_{ten}。

-5_{ten}用二进制补码表示为：

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{two}$

如果使用算术右移指令，在右移时自动进行符号扩展，而非简单地将空出的最高位填0。这样，用算术右移指令将-5_{ten}右移2位后，得到结果为：

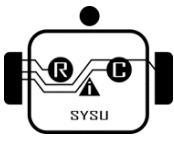
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$

这样得到的是-2_{ten}，而不是-1_{ten}。

谬误和陷阱

谬误3：只有数学家才需要关注浮点数的精度问题

- 1994年11月，各大报刊在Pentium处理器浮点瑕疵问题上大做文章，给Intel公司的声誉带来严重损害，并造成了5亿美元的巨额损失。

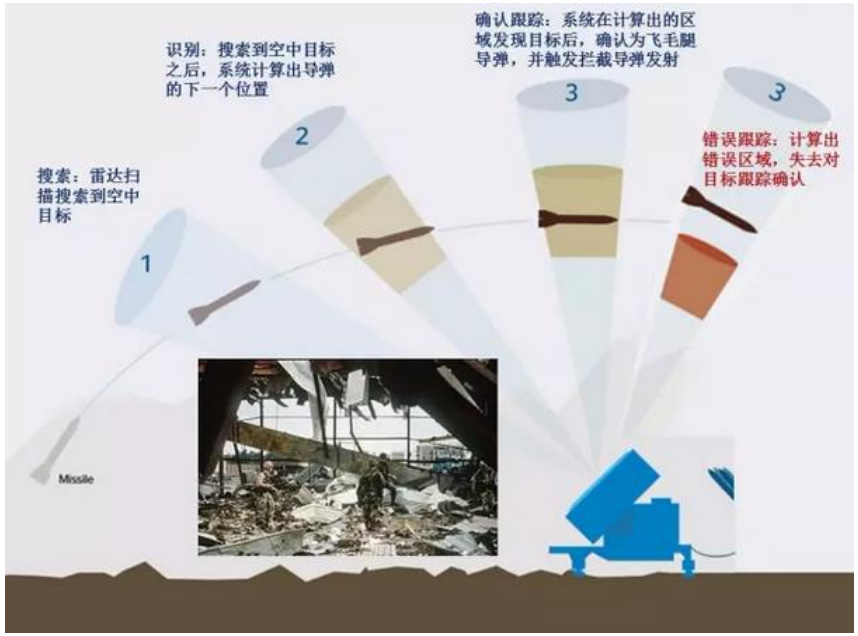


谬误和陷阱

误差的累积如何演变成生死攸关的灾难？

军事案例：1991年2月25日，海湾战争

因数据的精度问题，导致爱国者导弹拦截飞毛腿导弹失败，致使美军军营被击中，28人丧生



谬误和陷阱

爱国者导弹设计

- 24位寄存器（1970s的设计!）
- 计时以0.1秒为基础

但是，计算机无法精确表示0.1（1100循环）！

$$0.0001100110011001100110011001100_2 = \frac{209715}{2097152}$$

$$\left(\frac{1}{10} - \frac{209715}{2097152}\right)(3600 \times 100 \times 10) = \frac{5625}{16384} \approx 0.3433\text{秒}$$

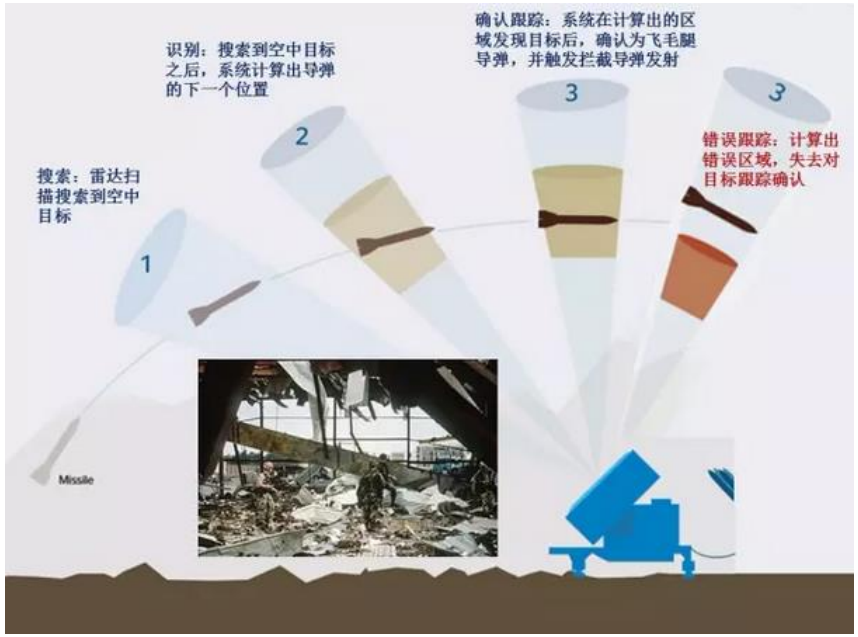
0.3433 * 2000米/秒 ≈ 687米

谬误和陷阱

误差的累积如何演变成生死攸关的灾难？

军事案例：1991年2月25日，海湾战争

因数据的精度问题，导致爱国者导弹拦截飞毛腿导弹失败，致使美军军营被击中，28人丧生



- 考虑到计算机每0.1秒存在百分之0.0001的计时误差，应该怎么做呢？

  A.每两个小时重启一次爱国者导弹的系统

  B.扩展24位模式为64位

  C.修改应用程序

联系方式

- Acknowledgements:
- This slides contains materials from following lectures:
  - Computer Architecture (ETH，NUDT，USTC)

- Research Area：
  - 计算机视觉与机器人应用计算加速，
  - 人工智能和深度学习芯片及智能计算机

- Contact：
  - 中山大学计算机学院
  - 管理学院D101（图书馆右侧）
  - 机器人与智能计算实验室
  - cheng83@mail.sysu.edu.cn