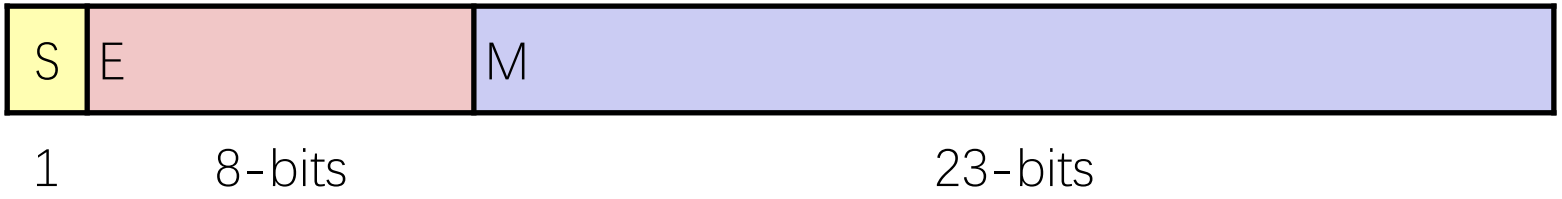
# 浮点数的运算

# 本节内容

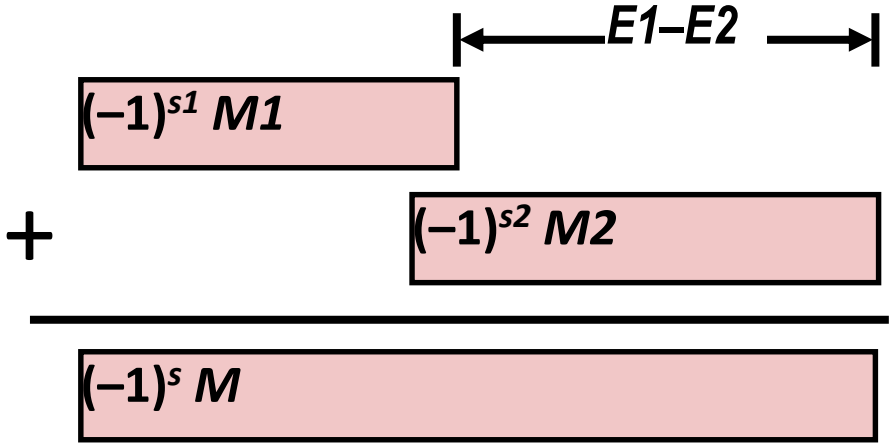

- 浮点加法
- 浮点乘法
- 运算结果的舍入

# 浮点数的算术运算

- 基本特点：浮点运算的结果不是精确的
- 基本思想：
  - $x +_f y = Round(x + y)$
  - $x \times_f y = Round(x \times y)$
  - 进行计算，将计算结果表示为规定的浮点数标准格式
    - 如果阶（E）太大，就溢出
    - 将运算结果的尾数位舍入到尾数（ **M** ）规定的长度



# 浮点数加法




- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$ ，假定 $E_1 > E_2$
- 运算结果: $(-1)^{s}\ M\ \ 2^{E}$

# 浮点加法的运算步骤

- 运算步骤:
  - 对阶（大阶往小阶对）
  - 尾数加减
  - 规格化（左规，右规）
  - 舍入
  - 检查溢出

# 举例


- 给定:
  - $A=2.6125\times10^{1}$, $B=4.150390625\times10^{-1}$
- 计算A+B

$$2.6125\times10^{1} + 4.150390625\times10^{-1}$$

计算过程：

1. 对阶：小阶往大阶对，

$2.6125\times10^{1}$

$= 26.125$

$= 11010.001$

$= 1.1010001000\times2^{4}$

$4.150390625\times10^{-1}$

$= .4150390625$

$= .011010100111$

$=1.1010100111\times2^{-2}$

$= 0.000001101010\ 0111\times2^{4}$　（小数点左移6位）

$$2.6125\times10^{1} + 4.150390625\times10^{-1}$$

计算过程:

2. 尾数相加

1.1010001000 00

+.0000011010 10 0111

---

1.1010100010 10 0111

3. 规格化检查

1.1010100010 10 0111 是IEEE 754规格化尾数

$$2.6125\times10^{1} + 4.150390625\times10^{-1}$$

计算过程：

4. 舍入：保留有效的尾数位，舍弃无法表示的尾数位

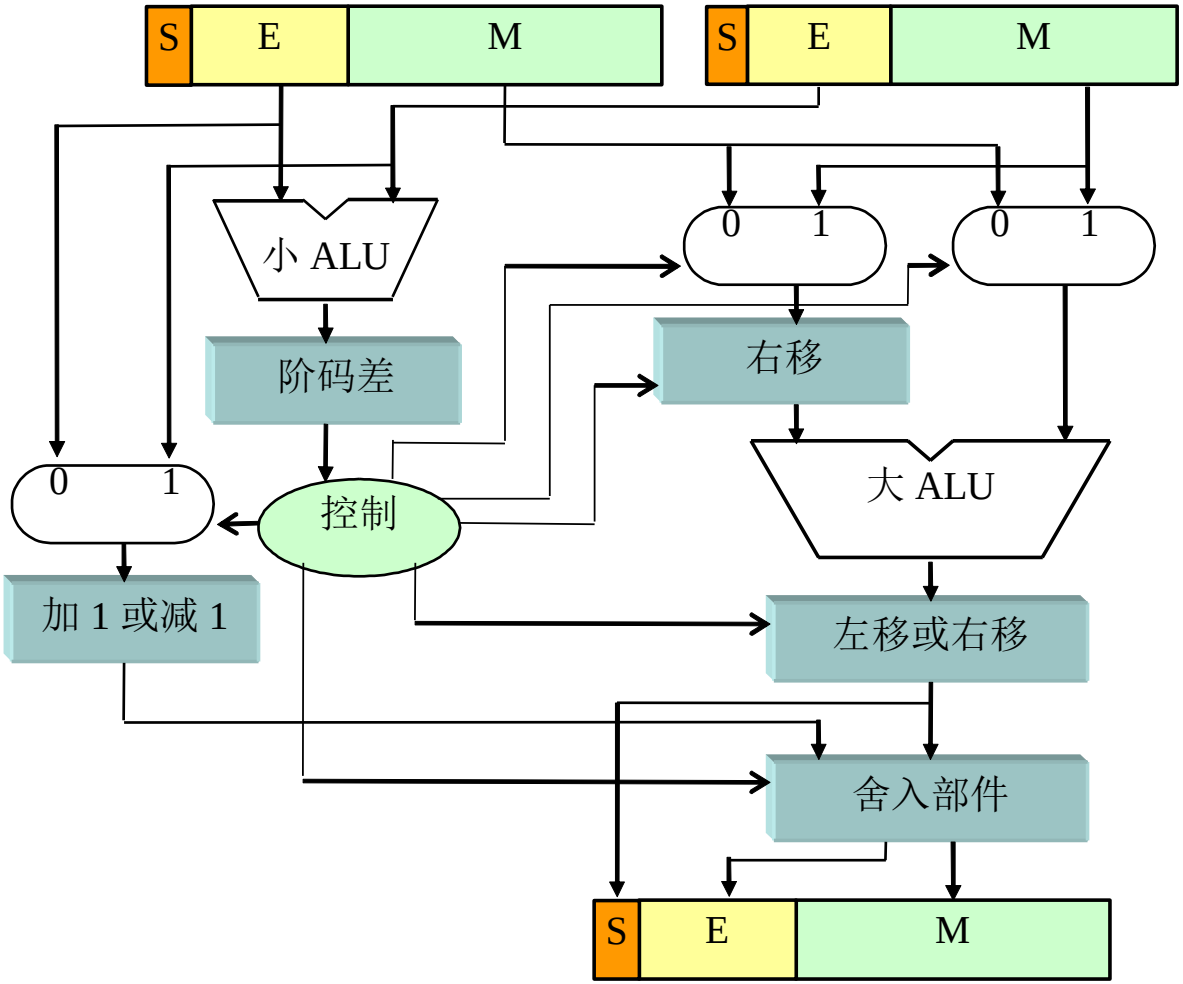
   1.1010100010 10 0111 → 1.1010100011 （10位尾数位）

5. 溢出检测：检查阶码是否在表示范围内能表达

$1.1010100011 \times 2^{4}$（阶数为4，是阶码可以表达的，无溢出）

$$
\begin{aligned}
& 1.1010100011 \times 2^{4} \\
= \ & 11010.100011 \times 2^{0} \\
= \ & 26.546875 \\
= \ & 2.6546875 \times 10^{1}
\end{aligned}
$$

# 浮点加法运算电路



# 讨论

float x,y,z;  // IEEE 754 单精度浮点数

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

- (x+y)+z == x+(y+z) 表达式是否为true?
- 不为true

$(x+y)+z = (-1.5\times10^{38}+1.5\times10^{38}) +1.0 = 1.0$

$x+(y+z) = -1.5\times10^{38}+ (1.5\times10^{38}+1.0) = 0.0$

# 浮点乘法

- $(-1)^{s1}$ ***M1*** $2^{E1}$ x $(-1)^{s2}$ ***M2*** $2^{E2}$
- 结果: $(-1)^{s}$ ***M*** $2^{E}$
- 步骤:
  - 阶码加
  - 尾数乘
  - 规格化:如果 $M \geqslant 2$, $M$ 每右移移位，$E$加*1*
  - 舍入:将$M$ 舍入到 尾数规定的位数
  - 检查溢出:如果 $E$ 超出了表达范围, 运算溢出

# 浮点运算结果的舍入

- 4种舍入模式:

| | $1.40 | $1.60 | $1.50 | $2.50 | –$1.50 |
|---|---|---|---|---|---|
| 向0（截断） | $1 | $1 | $1 | $2 | –$1 |
| 向正无穷大（向上） | $2 | $2 | $2 | $3 | –$1 |
| 向负无穷大（向下） | $1 | $1 | $1 | $2 | –$2 |
| 最近舍入(默认) | $1 | $2 | $2 | $2 | –$2 |

# 最近舍入（Round to Nearest）模式

- 又称为：首选“偶数”值（Round-To-Even）
  - 不是.5的舍入，同四舍五入
  - .5的舍入，取偶数。例如：取两位小数

| 最近舍入 | 四舍五入 |
| --- | --- |
| Round(0.5) = 0 | Round(0.5) = 0 |
| Round(1.5) = 2 | Round(1.5) = 2 |
| Round(2.5) = 2 | Round(2.5) = 3 |

# 二进制数的最近舍入

- 当要数值处于中间，即舍弃的位数的形式 $= 100\cdots_2$
  - 保证舍入后，最低有效位为 0
- 举例：
  - 保留小数点后两位

| 二进制编码 | 舍入后 | 动作 |
| --- | --- | --- |
| $10.00011_2$ | $10.00_2$ | (<1/2—down) |
| $10.00110_2$ | $10.01_2$ | (>1/2—up) |
| $10.11100_2$ | $11.00_2$ | ( 1/2—up) |
| $10.10100_2$ | $10.10_2$ | ( 1/2—down) |

# 为什么使用最近舍入?


- 当数值处于中间时
  - 一半时间向上舍入，一半时间向下舍入
- 减少运算中产生的误差
  - 例如：多个正数的加法，不能只往一个方向舍入，否则误差越来越大

# 小结

- 浮点加法、浮点乘法
  - 运算结果是不精确的
- 浮点运算结果的舍入
  - 最近舍入有利于减少运算中产生的误差