北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

2.6 浮点运算方法和浮点运算器

提纲

2.6.1 浮点加法、减法运算

- 1、浮点加减运算

- 设有两个浮点数 x 和 y ,它们分别为

$$x = 2^{E_x} \cdot M_x$$

$$y = 2^{E_y} \cdot M_y$$

- 其中E_x和E_y分别为数 x 和 y 的阶码,M_x和M_y为数 x 和 y 的尾数。两浮点数进行加法和减法的运算规则是:

$$x \pm y = (M_x 2^{E_x - E_y} \pm M_y) 2^{E_y}, \quad (M_x \text{右移})$$

设$E_x <= E_y$

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- 浮点加减运算的步骤
  - 对阶，使两数的小数点位置对齐。
  - 尾数求和，将对阶后的两尾数按定点加减运算规则求和(差)。
  - 规格化，为增加有效数字的位数，提高运算精度，必须将求和(差)后的尾数规格化。
  - 舍入，为提高精度，要考虑尾数右移时丢失的数值位。
  - 判断结果，即判断结果是否溢出

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- ①对阶

- 这一步操作是将两个加数的小数点对齐。
  - 小阶向大阶看齐，阶码较小的数，其尾数向右移，每右移一位，阶码加“1”，直到两数阶码相同为止。
- 尾数右移时可能会发生数码丢失，影响精度。

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- ①对阶
- 例：两浮点数x＝0.1101×2^{01}，y=-(0.1010)×2^{11}，求x+y。
- （1）首先写出x、y在计算机中的补码表示。

  $[x]_{补}$=00,01;00.1101，$[y]_{补}$=00,11;11.0110
- （2）在进行加法前，必须先对阶，故先求阶差：

  $[\Delta j]_{补}=[i_x]_{补}-[j_y]_{补}$=00,01+11,01=11,10
- 即Δj=-2，表示x的阶码比y的阶码小，再按小阶向大阶看齐的原则，将x的尾数右移两位，其阶码加2。

  得$[x]'_{补}$=00,11;00,0011
- 此时，Δj=0，表示对阶完毕。

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- ②尾数求和
- 将对阶后的两个尾数按定点加(减)运算规则进行运算。
  - 注意：并不考虑溢出——溢出由阶码决定

- 接上例，两数对阶后得：

  $[x]'_{补}$=00,11;00.0011

  $[y]_{补}$=00,11;11.0110

  则$[S_x+S_y]_{补}$=00.0011+11.0110=11.1001

  即$[x+y]_{补}$=00,11;11.1001

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- ③规格化
- 尾数S的规格化是指尾数满足条件：$\frac{1}{2} \le |S| < 1$

- 如果采用双符号位的补码，则
  - 当S>0时，其补码规格化形式为

    $[S]_{补}$=00.1××…×
  - 当S<0时，其补码规格化形式为

    $[S]_{补}$=11.0××…×
- 但对S<0时，有两种情况需特殊处理。
- S=-1/2，则$[S]_{补}$=11.100…0。对于补码而言，它不满足于上面的规格化表示式。为了便于硬件判断，特规定-1/2是规格化的数(对补码而言)。
- S=-1，则$[S]_{补}$=11.000…0。因小数补码允许表示-1，故-1视为规格化的数。

2.6.1 浮点加法、减法运算

- 1、浮点加减运算
- ③规格化
- 规格化又分左规和右规两种。
  - 左规。当尾数出现00.0××...×或11.1××...×时，需左规。左规时尾数左移一位，阶码减1，直到符合补码规格化表示式为止。
  - 右规。当尾数出现01.××...×或10.××...×时，表示尾数溢出，这在定点加减运算中是不允许的，但在浮点运算中这不算溢出，可通过右规处理。右规时尾数右移一位，阶码加1。

- 接上例，求和结果为$[x+y]_{补}$=00,11;11.1001
- 尾数的第一数值位与符号位相同，需左规，即将其左移一位，同时阶码减1，得$[x+y]_{补}$=00,10;11.0010。

2.6.1 浮点加法、减法运算

- ④舍入
- 在对阶和右规的过程中，可能会将尾数的低位丢失，引起误差，影响精度，为此可用舍入法来提高尾数的精度。
- 常用的舍入方法：
  - “0舍1入”法：“0舍1入”法类似于十进制运算中的“四舍五入”法，即在尾数右移时，被移去的最高数值位为0，则舍去；被移去的最高数值位为1，则在尾数的末位加1。这样做可能使尾数又溢出，此时需再做一次右规。
    - ✓ 特点：最大误差是最低位上的-1/2到接近于1/2之间，正误差可以和负误差抵消。属于比较理想的方法，但实现起来比较复杂。
  - “恒置1”法：尾数右移时，不论丢掉的最高数值位是“1”或“0”，都使右移后的尾数末位恒置“1”。这种方法同样有使尾数变大和变小的两种可能。
    - ✓ 特点：误差范围扩大，但正负误差可以相互抵消，实现相对容易。

2.6.1 浮点加法、减法运算

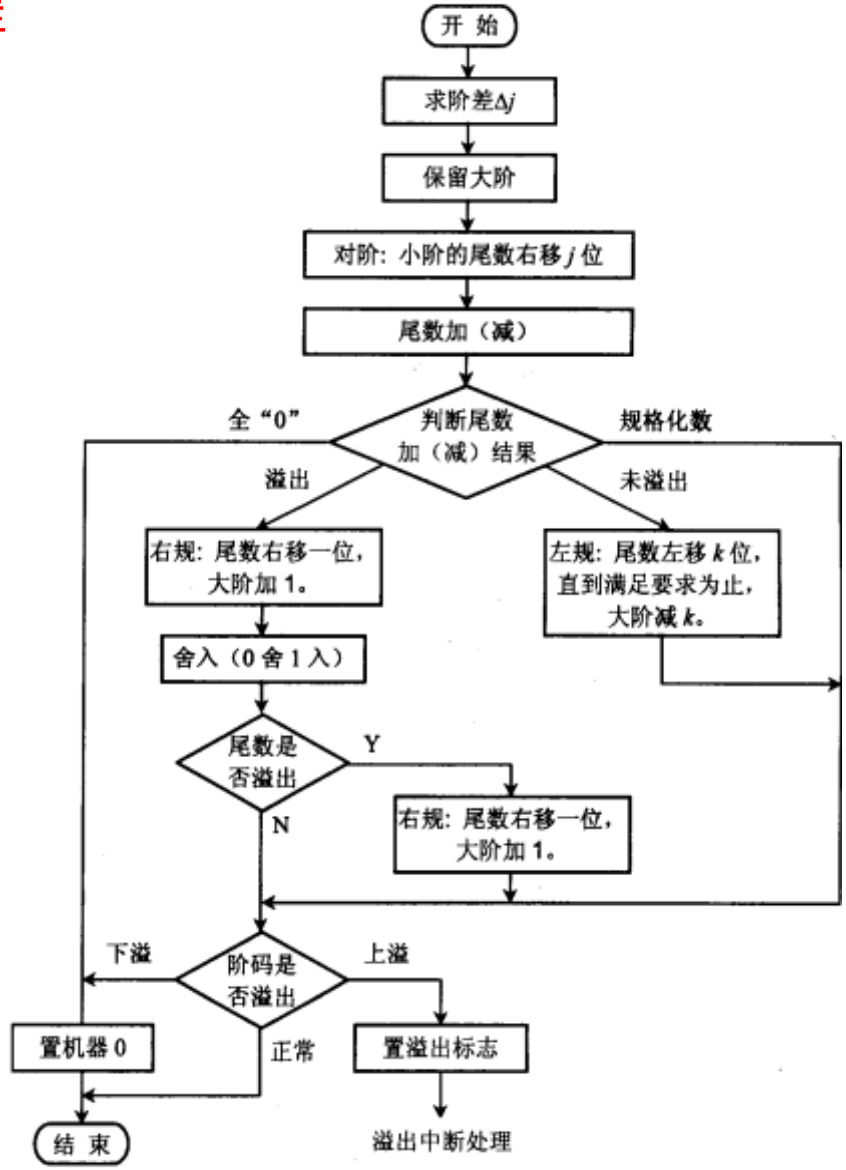
- 1、浮点加减运算

- ⑤溢出判断

- 在浮点规格化中已指出，当尾数之和(差)出现01.××...×或10.××...×时，并不表示溢出，只有将此数右规后，再根据阶码来判断浮点运算结果是否溢出。

2.6.1 浮点加法、减法运算

- 二、浮点数加减运算流程



2.6.1 浮点加法、减法运算

- 二、浮点数加减运算流程

- 大型计算机和高档微型机中，浮点加减法运算是由硬件完成的。低档的微型机浮点加减法运算是由软件完成的，但无论用硬件实现或由软件实现加减法运算，基本原理是一致的。

- 浮点加减法运算要经过对阶、尾数求和、规格化、舍入和溢出判断五步操作。其中尾数运算与定点加减法运算相同，而对阶、舍入、规格化和溢出判断，则是浮点加减法与定点加减法运算不同的操作。

- 在补码浮点运算中，阶码与尾数可以都用补码表示。在硬件实现的运算中，阶符和数符常常采取双符号位，正数数符用00表示，负数数符用11表示。

2.6.2 浮点乘法和除法运算

- 设有两个浮点数 x 和 y：

$$x = 2^{E_x} \cdot M_x$$

$$y = 2^{E_y} \cdot M_y$$

- $x \times y = 2^{(E_x + E_y)} \cdot (M_x \times M_y)$
- $x \div y = 2^{(E_x - E_y)} \cdot (M_x \div M_y)$
- 乘除运算分为四步
  - 0操作数检查
  - 阶码加减操作
  - 尾数乘除操作
  - 结果规格化和舍入处理

2.6.3 浮点运算流水线

- 一、提高并行性的两个渠道

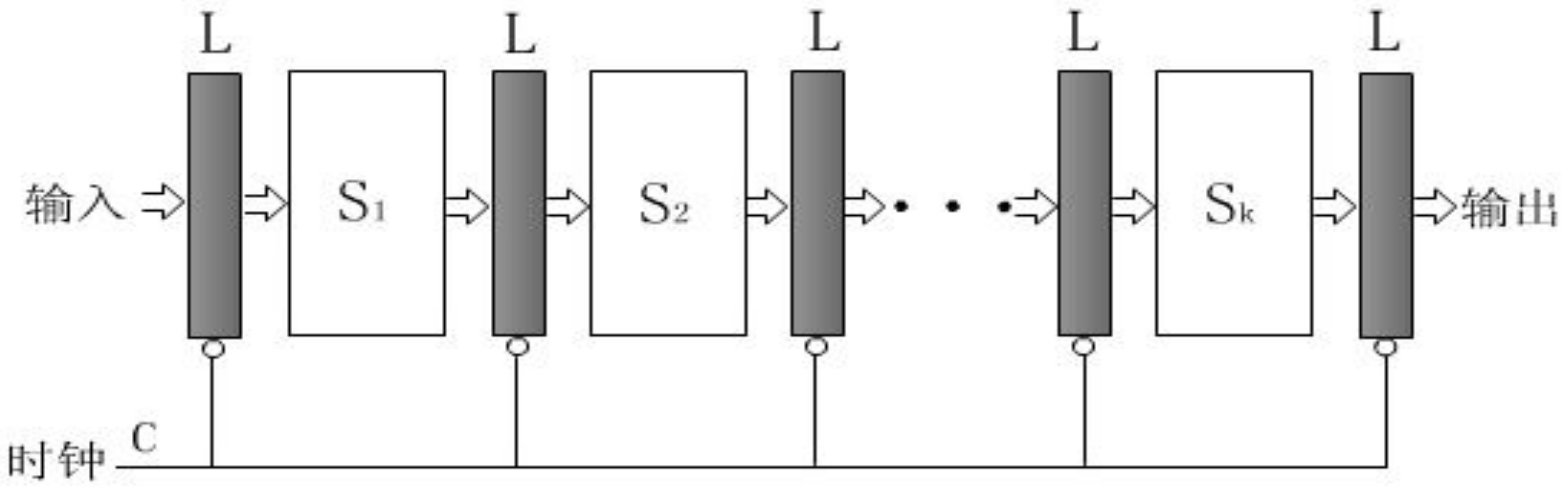
- 空间并行性：增加冗余部件，如增加多操作部件处理机和超标量处理机
- 时间并行性：改善操作流程如：流水线技术

2.6.3 浮点运算流水线

- 二、流水技术原理
- 在流水线中必须是连续的任务，只有不断的提供任务才能充分发挥流水线的效率
- 把一个任务分解为几个有联系的子任务。每个子任务由一个专门的功能部件实现
- 在流水线中的每个功能部件之后都要有一个缓冲寄存器，或称为锁存器
- 流水线中各段的时间应该尽量相等，否则将会引起“堵塞”和“断流”的现象
- 流水线需要有装入时间和排空时间，只有当流水线完全充满时，才能充分发挥效率

2.6.3 浮点运算流水线

- 二、流水技术原理



- 设过程段 S_i所需的时间为τ_i，缓冲寄存器的延时为τ_l，线性流水线的时钟周期定义为

$$\tau = \max\{\tau_i\} + \tau_l = \tau_m + \tau_l$$

- 流水线处理的频率为 $f = 1/\tau$。

2.6.3 浮点运算流水线

- 二、流水技术原理

- 一个具有k 级过程段的流水线处理 n 个任务需要的时钟周期数为$T_k = k + (n - 1)$
- 而同时，非流水线顺序完成的时间为：TL = n×k

- k级线性流水线的加速比：

$$C_k = \frac{TL}{Tk} = \frac{n \cdot k}{k + (n - 1)}$$

2.6.3 浮点运算流水线

- 三、流水线浮点运算器

- $A = a \times 2^P$, $B = b \times 2^q$
- 在4级流水线加法器中实现上述浮点加法时,分为以下操作:
- (1) 求阶差
- (2) 对阶
- (3) 相加
- (4) 规格化

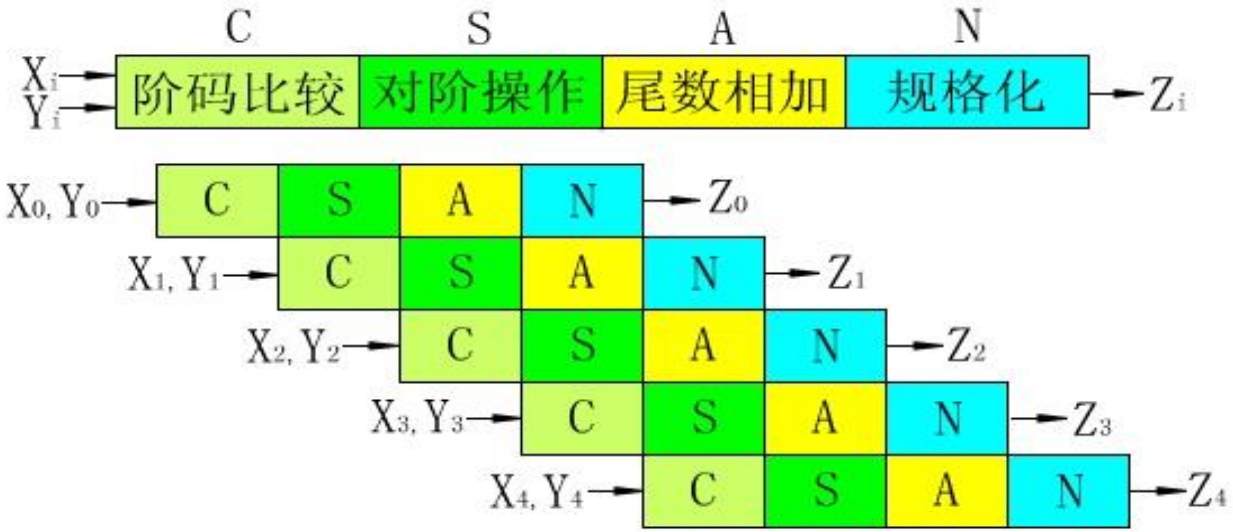


图2.21 向量加法计算的流水时空图

2.6.4 浮点运算器实例

- 浮点运算器实例
  - CPU之外的浮点运算器（数学协处理器）如80287
    - ✓ 完成浮点运算功能，不能单用。
    - ✓ 可以和80386或80286异步并行工作。
    - ✓ 高性能的80位字长的内部结构。有8个80位字长以堆栈方式管理的寄存器组。
    - ✓ 浮点数格式完全符合IEEE标准。
  - CPU之内的浮点运算器（486DX以上）

作业

- 第二章:
- 作业：1、3、4、5、6、7(1)、8(1)、9(1)、10(1)、11