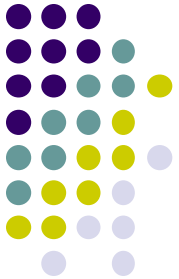


计算机组成与系统结构

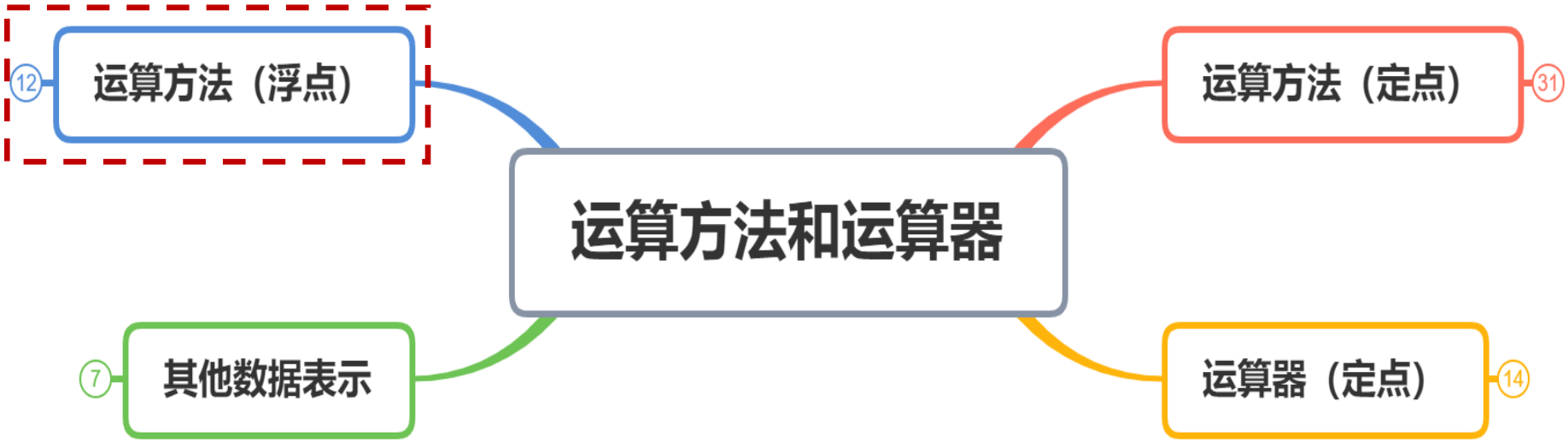
第二章 运算方法和运算器（4）

吕昕晨

lvxinchen@bupt.edu.cn

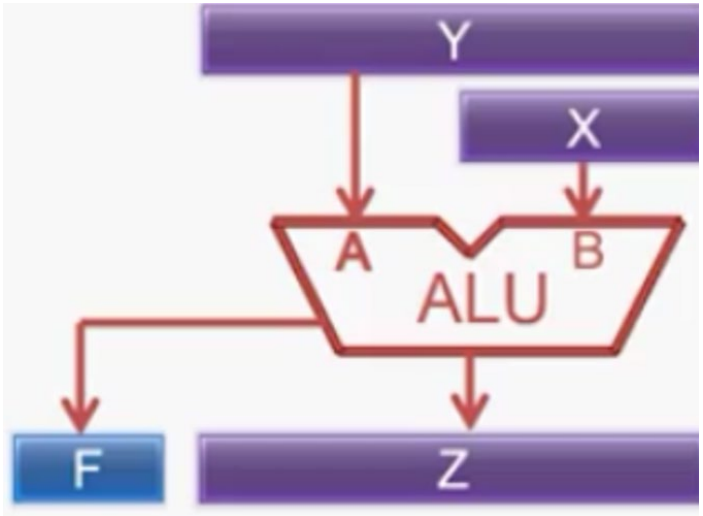
网络空间安全学院

运算方法与运算器



第二章剩余教学安排

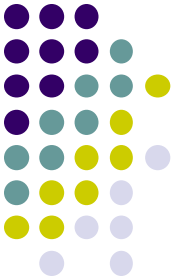
- 上节课
  - 定点数（理论+硬件）
    - 乘法（重点）
    - 除法（重难点）
- 本节课
  - 浮点数
    - 加减、乘除法
  - 运算器总线结构



习题课（第一/二章）

下周

第二章 运算方法和运算器



- 复习：浮点表示法（IEEE 754）
- 浮点加、减法
- 浮点乘、除法
- 浮点运算器流水线结构
- 运算器总线结构

IEEE 754标准——32位浮点数

- 基数R=2，基数固定，采用隐含方式来表示它。
- 32位的浮点数：
  - S数的符号位，1位，在最高位，0表示正数，1表示负数
  - M是尾数， 23位，在低位部分，采用纯小数表示
  - E是阶码，8位，采用移码表示。移码比较大小方便
  - 注意：
    - 关于尾数：**尾数域最左位(最高有效位)总是1，故这一位不予存储**，而认为隐藏在小数点的左边。
    - 关于阶码：由于采用移码，将浮点数的指数真值e变成阶码E时，应将指数e加上一个固定的偏移值127(01111111)，即E=e+127。

IEEE 754总结：浮点数转十进制

- 将16进制转换为2进制，根据标准得到S、E、M
  - S：1位、E：8位、M：23位
- 写出实际指数e、实际尾数1.M
  - 指数e=阶码E-127（移码）
  - 实际尾数为1.M（省略尾数多余的0）
- 写出二进制表达式，并计算十进制
  - x=(符号位+/-)1.M ×2^e
  - 根据R进制基本公式，计算x的十进制

IEEE 754总结：十进制转浮点数

- 十进制转二进制
  - 十进制转二进制（整数、小数分别转换）
- 写出浮点数的标准公式，得到S、1.M、e
  - x=(符号位+/-)1.M ×2^e
  - 写出S=0/1、1.M、e
- 写出M、E
  - 写出M（注意省略开头1）、E=e+127
- 写出浮点数表示
  - 根据格式写出S、E、M的二进制，并写出二进制格式
  - 二进制转换16进制

IEEE 754习题

某C语言程序变量num声明为float型（IEEE 754标准）。

GDB调试该程序，观察到num存放在起始地址为0x400508的内存中，内容依次为0x00、0x80、0xC5、0xC2（注意：该计算机存储方式为小端模式）。

回答下列问题：

1）变量num的754标准存储格式为_____（十六进制）。

2）该浮点数的符号位为_____，指数为_____。

3）num的十进制值为_____。

IEEE 754习题——解答

某C语言程序变量num声明为float型（IEEE 754标准）。

GDB调试该程序，观察到num存放在起始地址为0x400508的内存中，内容依次为0x00、0x80、0xC5、0xC2（注意：该计算机存储方式为**小端模式——数据低字节存放在内存低地址空间中**）。

1）变量num的754标准存储格式为_____（十六进制）

C2 C5 80 00H（小端模式）

2）该浮点数的符号位为_____，指数为_____。

1 100 0010 1 100 0101 1000 0000 0000 0000 0000

E-127= 0000 0110 = 6

3）num的十进制值为_____。

$-1.M * 2^6 = -(1.100010\underset{(右移6位)}{.}11)_2 = -(98.75)_{10}$

大小端模式

多字节数据在内存里一定是占连续的几个字节

最高有效字节(MSB)

最低有效字节(LSB)

4字节 int：01 23 45 67 H　　19088743 D

0000 0001 0010 0011 0100 0101 0110 0111 B

便于人类阅读

大端方式

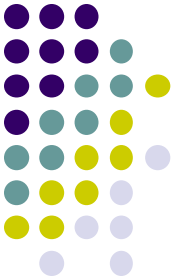
	0800H	0801H	0802H	0803H	
…	01H	23H	45H	67H	…

便于机器处理

小端方式

	0800H	0801H	0802H	0803H	
…	67H	45H	23H	01H	…

第二章 运算方法和运算器



- 复习：浮点表示法（IEEE 754）
- 浮点加、减法（重点）
- 浮点乘、除法
- 浮点运算器流水线结构
- 运算器总线结构

浮点加减法运算

（若无特殊说明，非754标准）

- 设有两个浮点数x和y，它们分别为
  - $x = 2^{E_x} \cdot M_x$
  - $y = 2^{E_y} \cdot M_y$
  - 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
  - $x \pm y = (M_x 2^{E_x - E_y} \pm M_y) 2^{E_y}$
  - 设$E_x <= E_y$
  - 不失一般性，统一成较大数阶码

浮点加减法运算运算步骤

1）操作数检查（避免无效操作）

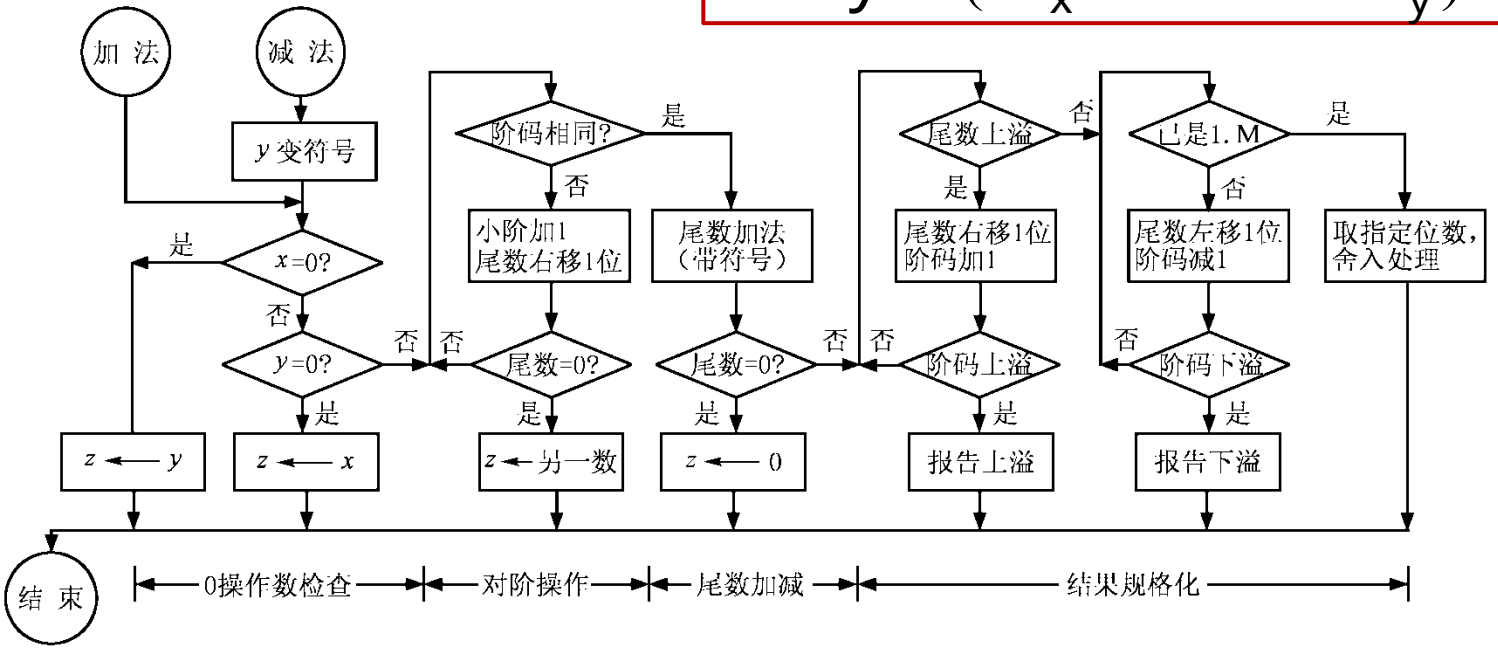
2）比较阶码并完成对阶（小阶向大阶对齐）

3）尾数加减运算

4）结果规格化（规范化，最高位为1）

5/6）舍入处理/溢出判断

$$x \pm y = (M_x 2^{Ex - Ey} \pm M_y) 2^{Ey}$$

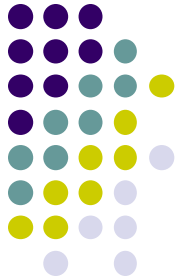


浮点加减法示例——补码

[例] 设$x=2^{010}\times0.11011011$,$y=-2^{100}\times0.10101100$， 求x+y

（采用补码计算，尾数精度为8位）

浮点加减法示例——补码（1）



[例] 设$x=2^{010}\times0.11011011$,$y=-2^{100}\times0.10101100$， 求x+y

（采用补码计算，尾数精度为8位）

1、0操作数检查（非0）

2、对阶：阶码对齐后才能加减。规则是阶码小的向阶码大的数对齐；

- 若△E＝0，表示两数阶码相等,即E x ＝E y ；
- 若△E>0，表示E x >E y ；
- 若△E<0，表示E x >E y 。
- 当E x ≠E y 时，要通过尾数的移动以改变E x 或E y ，使之相等。

- $[x]_{浮}$→Ex=00010，Mx=0.11011011
- $[y]_{浮}$→Ey=00100，My=1.01010100（补码，取反加一）；
- 对阶，x向y对齐
- $[x]_{浮,对阶}$→Ex=00100，Mx=0.00110110(11)，尾数精度8位

浮点加减法示例——补码（2）

3、尾数相加（带符号位）

- 0.00110110(11)

  ＋ 1.01010100

  1.10001010(11)

4、规格化

- 纯小数格式应为 （+/-） 0.1XXXXX
- 规则
  - 尾数右移1位，阶码加1
  - 尾数左移1位，阶码减1

左移 v.s. 右移？

浮点加减法示例——补码（3）

0. 0 0 1 1 0 1 1 0 (11)

＋ 1. 0 1 0 1 0 1 0 0

1. 1 0 0 0 1 0 1 0 (11)

S>0	规格化形式	S<0	规格化形式
真值	0.1× × … ×	真值	-0.1 × × … ×
原码	0.1 × × … ×	原码	1.1 × × … ×
补码	0.1 × × … ×	补码	1.0 × × … ×

补码：带符号判定移位（方式2）

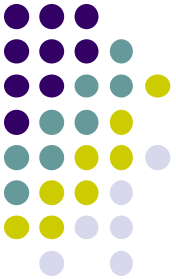
4、规格化

- 规则
  - 尾数右移1位，阶码加1
  - 尾数左移1位，阶码减1
- 左规处理，结果为1.00010101(10)，阶码为00011

Ex=00100

建议：转回原码，确认规格化结果是否正确（方式1）

浮点加减法示例——补码（4）



5、舍入处理（对阶和向右规格化时）

- 就近舍入(0舍1入)：类似”四舍五入”
- 朝0舍入：截尾
- 朝＋∞舍入：正数多余位不全为0，进1；负数，截尾
- 朝－∞ 舍入：负数多余位不全为0，进1；正数，截尾

采用0舍1入法处理，得到1.00010110

1.00010101(10)

6、溢出判断和处理

阶码符号位为00，不溢出。得最终结果为

$$x+y = 2^{011} \times (-0.11101010)$$

浮点加减法示例——原码计算

[例] 设$x=0.5_{10}$，$y=-0.4375_{10}$，尾数用原码，求$(x+y)_{浮}$

浮点加减法示例——原码计算

[例] 设$x=0.5_{10}$，$y=-0.4375_{10}$，尾数用原码，求$(x+y)_{浮}$

1、写出操作数（浮点数格式）

$x=0.1_2=0.1_2\times 2^0=1.000\times 2^{-1}$

$y=-0.0111_2=-0.0111_2\times 2^0=-1.110\times 2^{-2}$

2、对阶：y阶码更小，调整y使对齐

$y=-1.110\times 2^{-2}=-0.111$（尾数右移1位）$\times 2^{-1}$

3、尾数相加

$x+y=1.000-0.111=0.001$ （$\times 2^{-1}$）

4、规格化

$x+y=0.001\times 2^{-1}=1.000$（小数点右移3位）$\times 2^{-4}$

5、舍入操作

尾数有效位为4位，不进行舍入操作

单选题 1分

设置

$x=0.1101 \times 2^{01}$ ； $y=-0.1010 \times 2^{11}$

尾数和阶符都采用补码表示，都采用双符号位表示法，尾数有效位4位，就近舍入规则。 求x+y=?

A -0.1011×2^{01}

B -0.1011×2^{10}

C -0.1101×2^{01}

D -0.1101×2^{10}

习题求解过程

$x=0.1101 \times 2^{01}$ ； $y=-0.1010 \times 2^{11}$

$[x]_{浮}$=0001，00.1101 （双符号位补码）

$[y]_{浮}$=0011，11.0110

对阶：x向y对齐，小数点左移2位

$[x]_{浮}$=0011，00.0011（01）

尾数和为11.1001（01）

规格化（小数点右移1位）11.0010(10)，阶码减1→0010

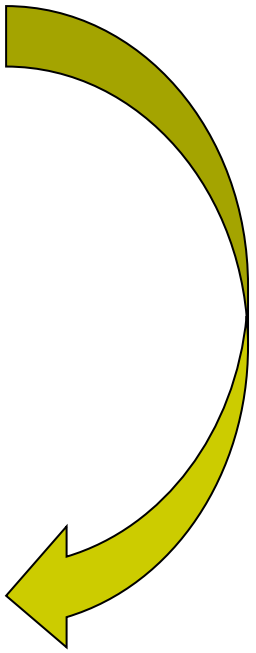
舍入（就近舍入）11.0011

$x+y=-0.1101*2^{10}$

扩展：补码四舍五入（-0.5问题）

对如下补码进行四舍五入：

	1）11 0010.11	2）11 0010.01
原码：	11 1101.01	11 1101.11
真值：	-13.25	-13.75
四舍五入：	-13	-14
原码：	11 1101	11 1110
补码：	11 0011	11 0010
规律：	0舍1入（就近舍入）	



建议：在补码形式下，进行舍入处理

扩展：补码四舍五入（-0.5问题）

对如下补码进行四舍五入：

3）11 0010.10

原码：11 1101.10

真值：-13.5

四舍五入：-13

原码：11 1101

补码：11 0011

输入：

```
1 import numpy as np
2
3 print(np.round(0.5))
4 print(np.round(1.5))
```

输出：

```
1 0.0
2 2.0
```

python3.5

Math.round()的四舍五入规则是加0.5向下取整

转载 熊仙森 2019-02-18 21:35:59 5787 收藏 4

分类专栏： Java 文章标签： Math.round()

问：Math.round(15.5) 等于多少？Math.round(-15.5) 等于多少？

答：**分别等于 16 和 -15。**

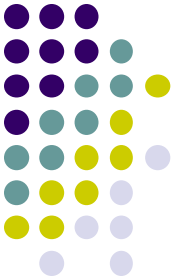
因为四舍五入的原理是在参数上加 0.5 然后进行下取整。 所以类似的 Math.round(15.6) 计算方式为 15.6 + 0.5 = 16.1，接着向下取整数为 16；Math.round(-15.6) 计算方式为 -15.6 + 0.5 = -15.1，接着向下取整数为 -16。

这是一个很小但是很坑的知识点，切记 **不是四舍五入，是加 0.5 向下取整数。**

注：不同编译器对round定义不同

规律：0舍1入（就近舍入）

第二章 运算方法和运算器



- 复习：浮点表示法（IEEE 754）
- 浮点加、减法
- 浮点乘、除法
- 浮点运算器流水线结构
- 运算器总线结构

浮点乘法和除法运算

- 设有两个浮点数 x 和 y：

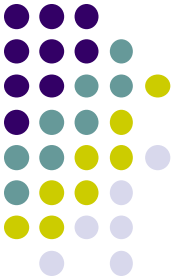
$$x = 2^{E_x} \cdot M_x$$

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

- $x \times y = 2^{(E_x + E_y)} \cdot (M_x \times M_y)$
- $x \div y = 2^{(E_x - E_y)} \cdot (M_x \div M_y)$

- 乘除运算分为四步（**无对阶、符号位单独处理**）
  1. 0操作数检查
  2. 阶码加减操作
  3. 尾数乘除操作
  4. 结果规格化和舍入处理

浮点乘法示例——原码计算



[例] 设$x=0.5_{10}$，$y=-0.4375_{10}$，尾数用原码，求$(x \times y)_{浮}$

1、写出操作数（浮点数格式）

$x=0.1_2=0.1_2*2^0=1.000*2^{-1}$

$y=-0.0111_2=-0.0111_2*2^0=-1.110*2^{-2}$

2、阶码相加

$-1+-2=-3$

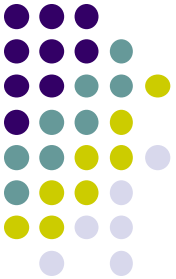
3、尾数相乘

$x \times y=1.110 \quad (\times 2^{-3})$

4、规格化与确定符号

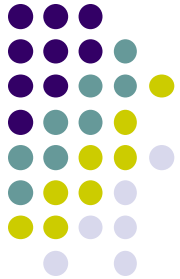
$x \times y=-1.110 \times 2^{-3}$

第二章 运算方法和运算器



- 复习：浮点表示法（IEEE 754）
- 浮点加、减法
- 浮点乘除法
- 浮点运算器流水线结构
- 运算器总线结构

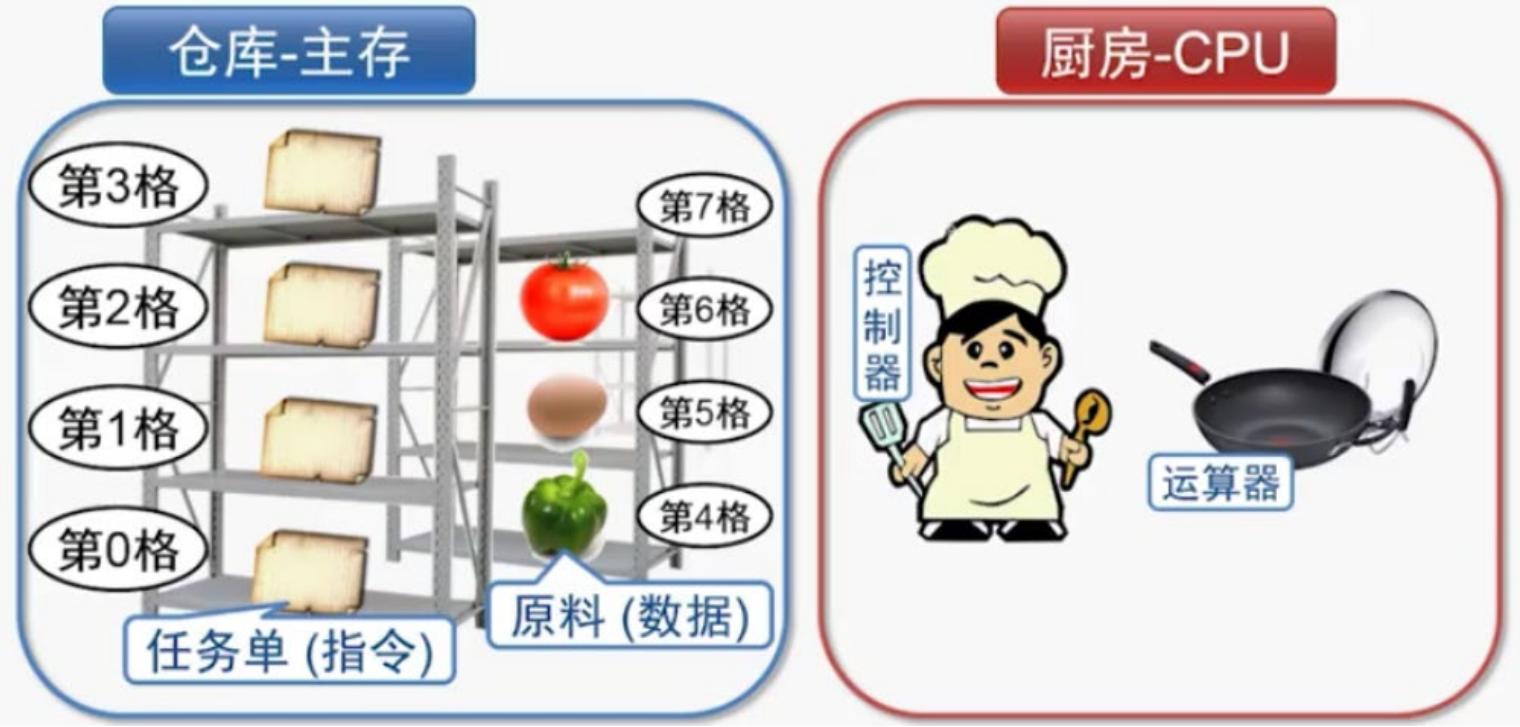
浮点运算流水线



- 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

流水线技术原理——餐厅模型

- 另一种类比：大厨→控制+运算
- 非流水线：大厨单线程工作：洗、切、炒（3T）
- 流水线：洗、洗/切、洗/切/炒……（最好为T）



流水线浮点运算器——时空图

$A = a \times 2^{P},\quad B = b \times 2^{q}$

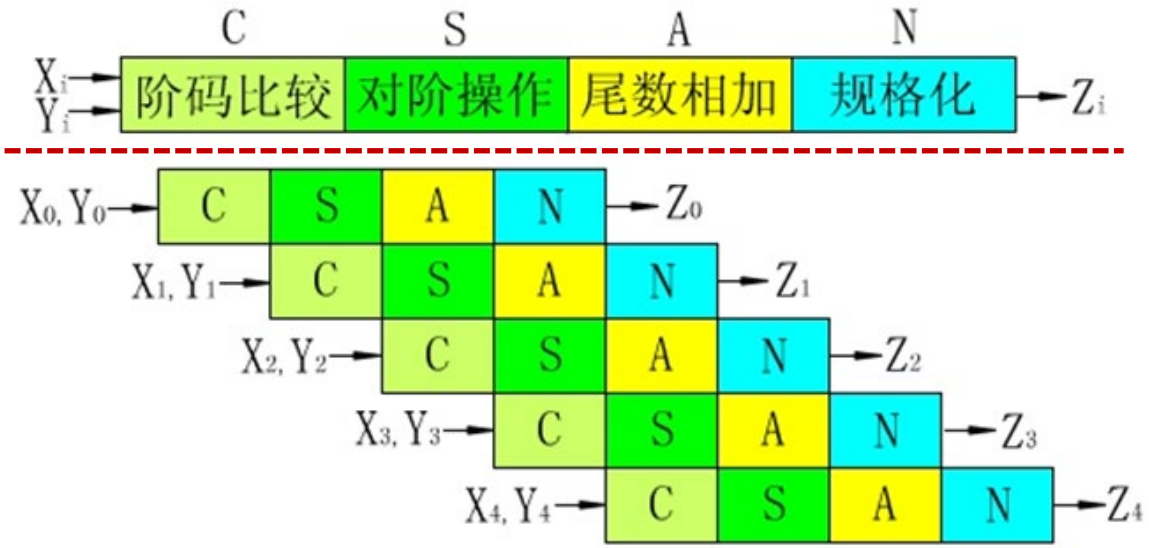
在4级流水线加法器中实现上述浮点加法时，分为以下操作：

(1) 求阶差

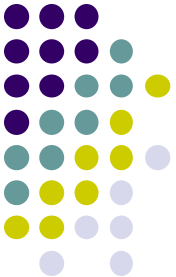
(2) 对阶

(3) 相加

(4) 规格化

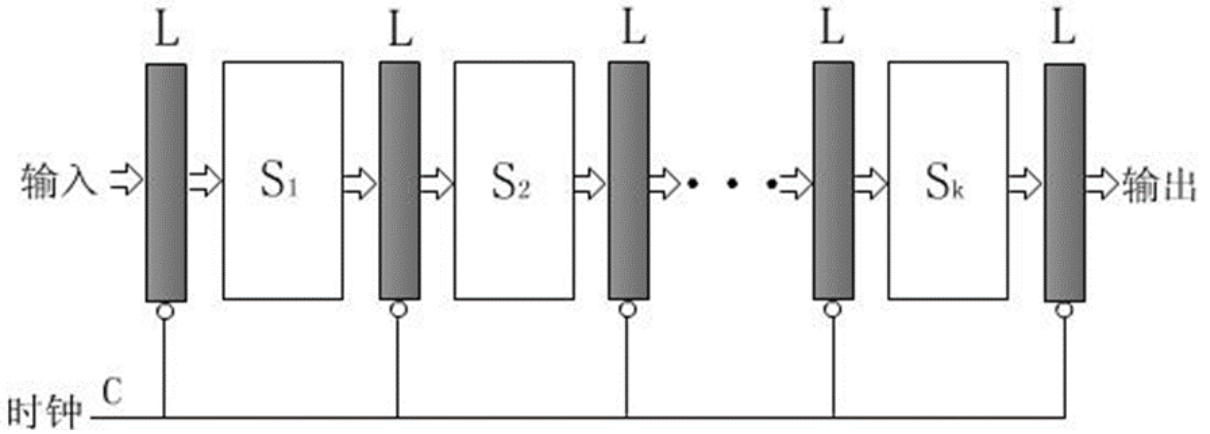


流水线技术原理



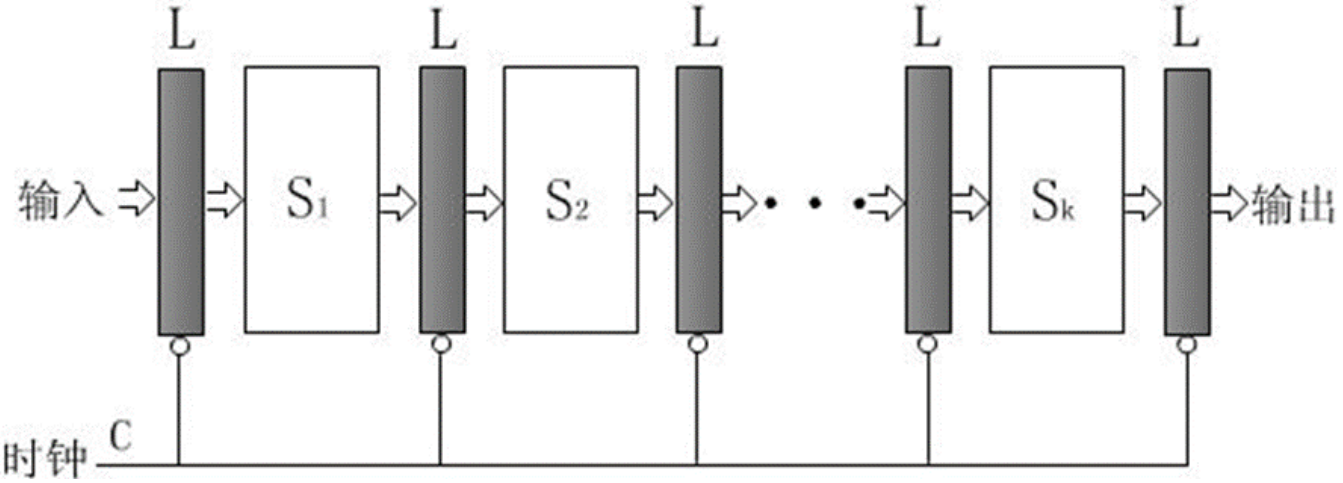
- 在流水线中必须是连续的任务，只有不断的提供任务才能充分发挥流水线的效率
- 把一个任务分解为几个有联系的子任务。每个子任务由一个专门的功能部件实现
- 在流水线中的每个功能部件之后都要有一个缓冲寄存器，或称为锁存器（各级流水独立）
- 流水线中各段的时间应该尽量相等，否则将会引起“堵塞”和“断流”的现象
- 流水线需要有装入时间和排空时间，只有当流水线完全充满时，才能充分发挥效率

流水线原理——理想情况加速比



- 流水线加速比（k级流水，执行n条指令，C_k）
  - 假设各级执行时间相等，不考虑缓存器延迟
  - 非流水线，串行执行$T_L = n\,k\,\tau$（每一级时间）
  - 流水线方式，$T_k = k\,\tau + (n-1)\,\tau$
  - 加速比，$C_k = T_L / T_k = nk/(k+n-1)$
  - 当$n \to \infty, C_k = k$（最多提升k倍性能）

流水线原理——非理想情况

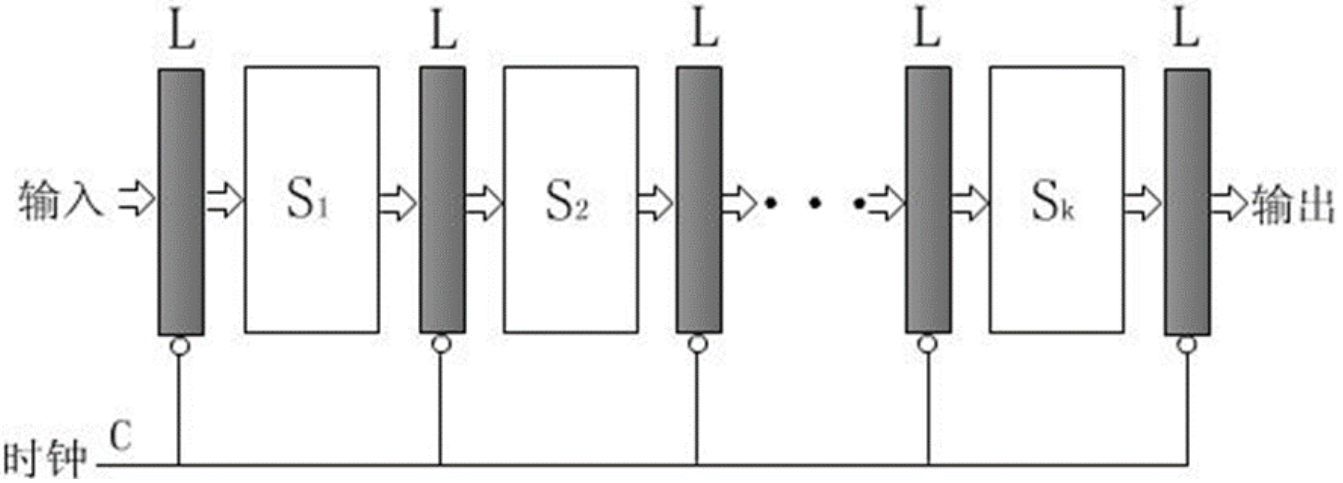


- 设过程段 S_i所需的时间为τ_i,缓冲寄存器的延时为τ_l,线性流水线的时钟周期定义为

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

- 流水线处理的频率为 $f = 1/\tau$

流水线原理——非理想情况



- 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 $f = 1/\tau$

$$C_k = T_L / \tau$$

流水线计算例题

某4级流水线浮点加法器，各级所需时间为：求阶差 $\tau1 = 70ns$，对阶 $\tau2 = 60ns$，相加$\tau3 = 90ns$，规格化 $\tau4 = 80ns$，缓冲寄存器L的延时为 $\tau1 = 10ns$。

求：1）加速比；2）若各级流水线时间均为75ns（含缓冲器）加速比为?

流水线计算例题（1）

某4级流水线浮点加法器，各级所需时间为：求阶差 $\tau1 = 70ns$，对阶 $\tau2 = 60ns$，相加$\tau3 = 90ns$，规格化 $\tau4 = 80ns$，缓冲寄存器L的延时为 $\tau1 = 10ns$。

求：1）加速比；2）若各级流水线时间均为75ns（含缓冲器）加速比为?

解答:

1）流水加法器时钟周期为:

$\tau = \max\{\tau_i\} + \tau_l = \tau_m + \tau_l$=90+10=100ns

同样电路非流水线时间为70+60+90+80=300ns

加速比为Ck=300/100=3

流水线计算例题（2）

某4级流水线浮点加法器，缓冲寄存器L的延时为 $\tau 1 = 10ns$。

求：2）若各级流水线时间均为75ns（含缓冲器）加速比为?

解答:

2）流水加法器时钟周期为75ns

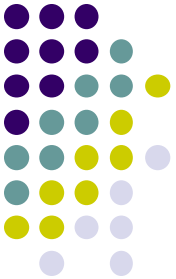
不含缓冲每一级时间为75-10=65ns

同样电路非流水线时间为65 × 4=260ns

加速比为Ck=260/75=3.466……

或 加速比为300/75=4

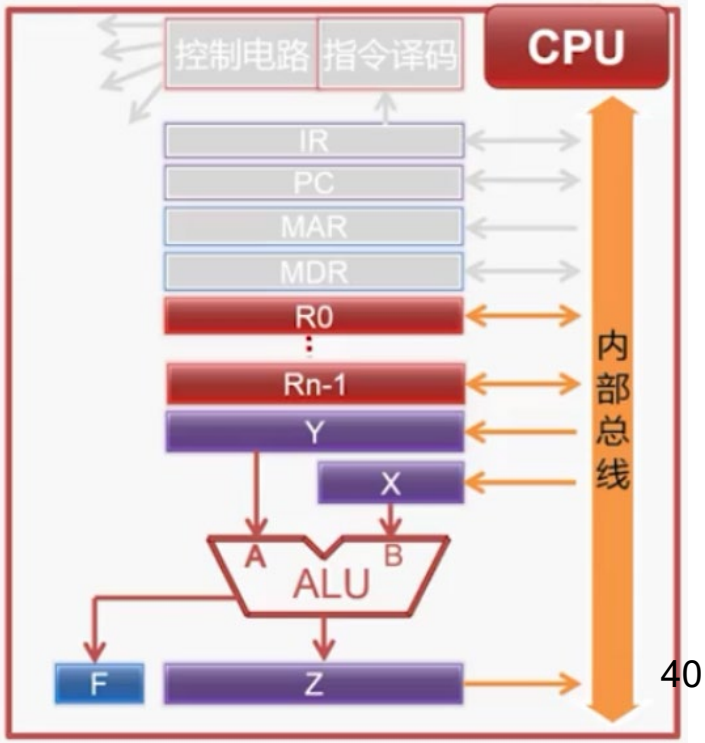
第二章 运算方法和运算器



- 复习：浮点表示法（IEEE 754）
- 浮点加、减法
- 浮点乘除法
- 浮点运算器流水线结构
- 运算器总线结构

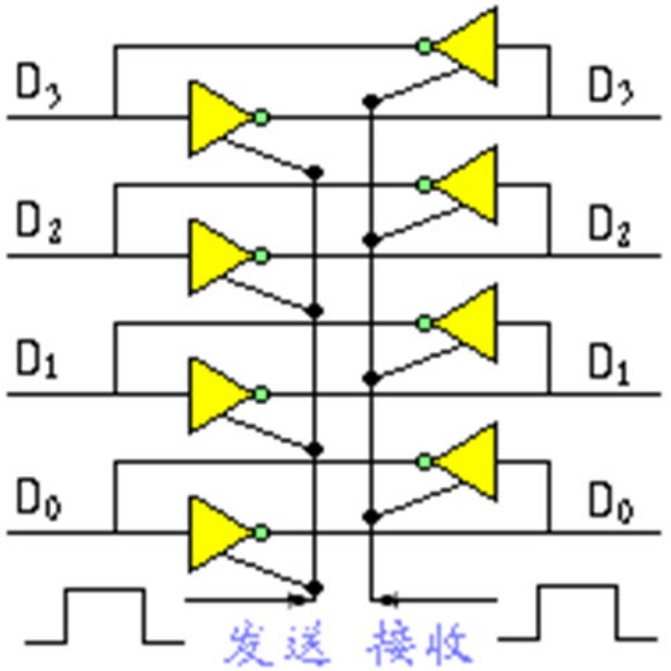
内部总线

- 内部总线
  - 机器内部各部份数据传送频繁，可以把寄存器间的数据传送通路加以归并，组成总线结构
  - 分类
    - 所处位置
      - 内部总线（CPU内）
      - 外部总线（系统总线）
    - 逻辑结构
      - 单向传送总线
      - 双向传送总线



带缓冲的双向数据总线

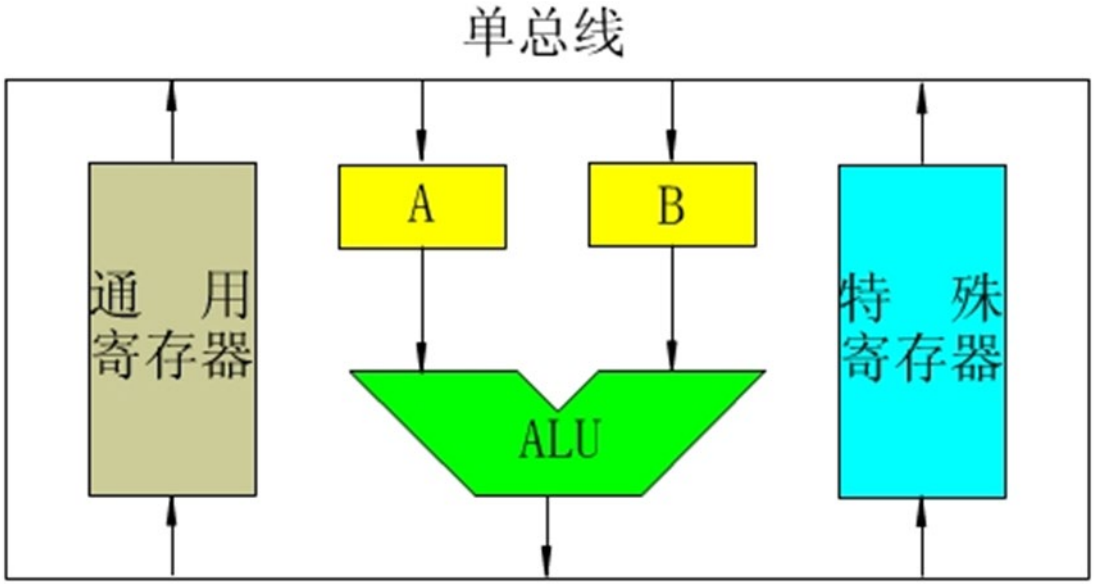
- 三态门逻辑电路
  - 高阻态（等效断路）
  - 连通态（输出0/1）
  - 开关功能
  - 数据选择器、总线结构
- 4位带缓冲双向数据总线
  - 8个三态缓冲器（发送、接收）
  - 控制端
    - 发送信号
    - 接收信号



(a) 带有缓冲器的双向数据总线

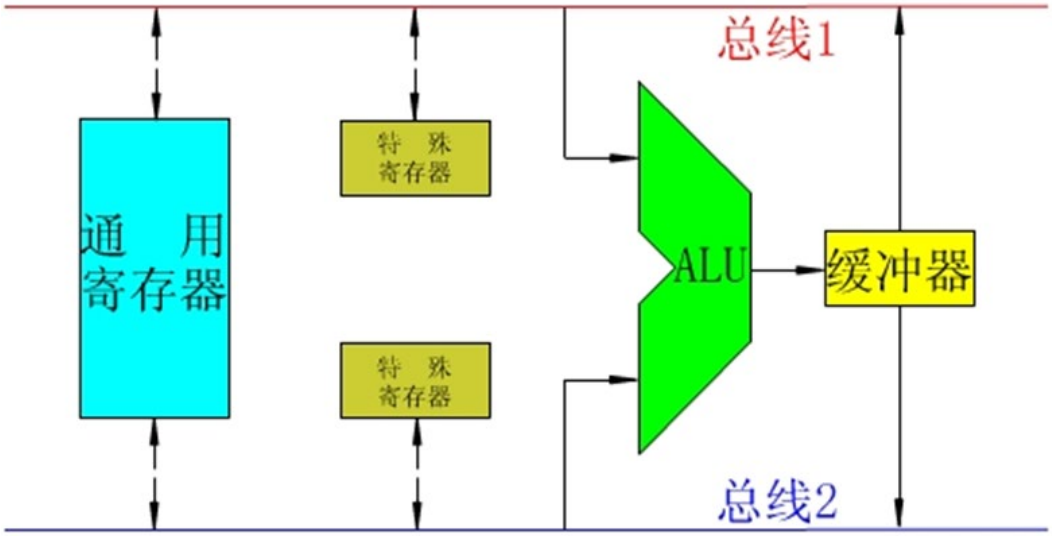
单总线结构运算器

- 特点：所有部件连接到同一总线上，控制简单
- 性能
  - 同一时刻仅允许一个操作数出现在总线上
  - 数据存入A、B寄存器：2个时钟周期
  - 结果输出：1个时钟周期



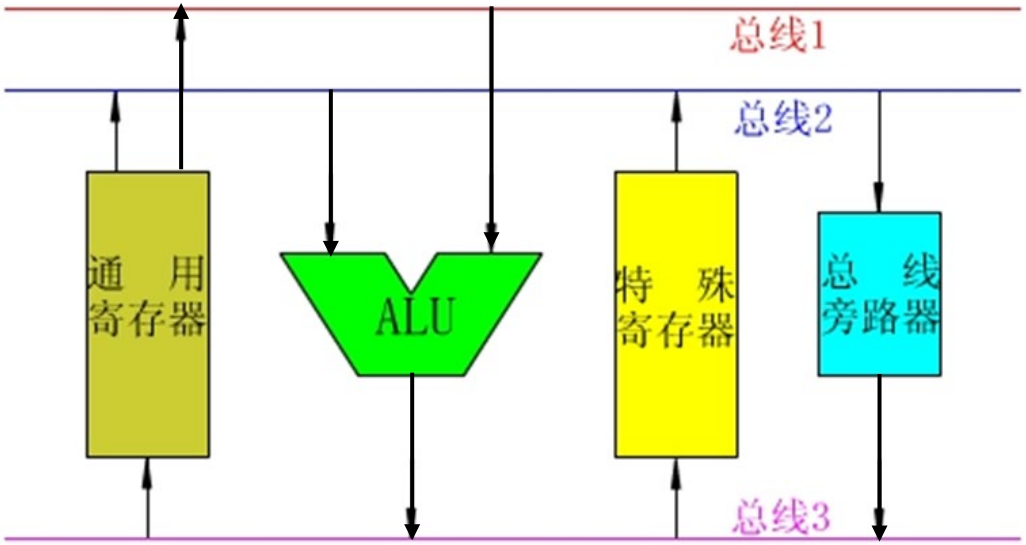
双总线结构运算器

- 特点：ALU输入端由不同总线连接
- 性能
  - 两个操作数可同时送入ALU
  - 数据输入：1个时钟周期
  - 结果输出：1个时钟周期



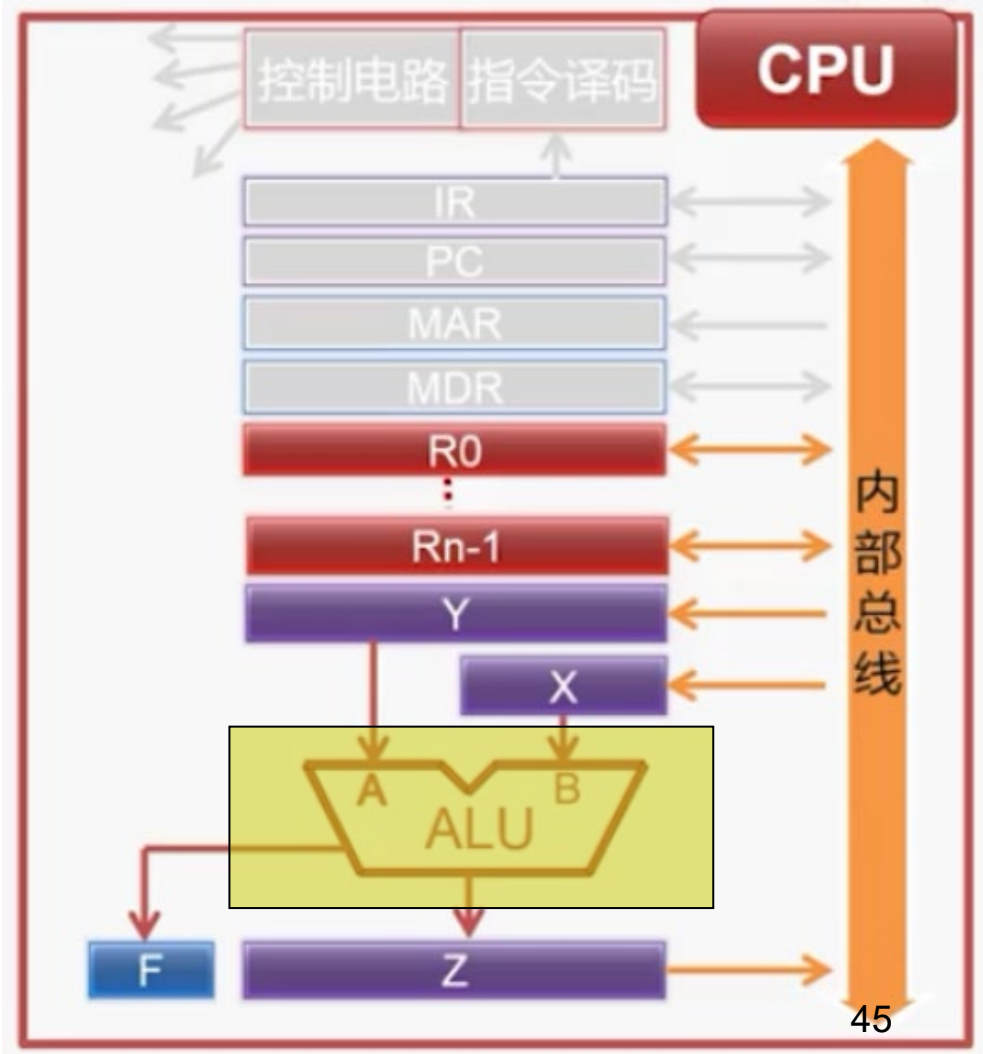
三总线结构运算器

- 特点：ALU输入端、输出端由不同总线连接
- 性能
  - 1个时钟周期，进行输入输出
  - 选通脉冲，考虑ALU延迟
  - 总线旁路器：操作数不需要修改

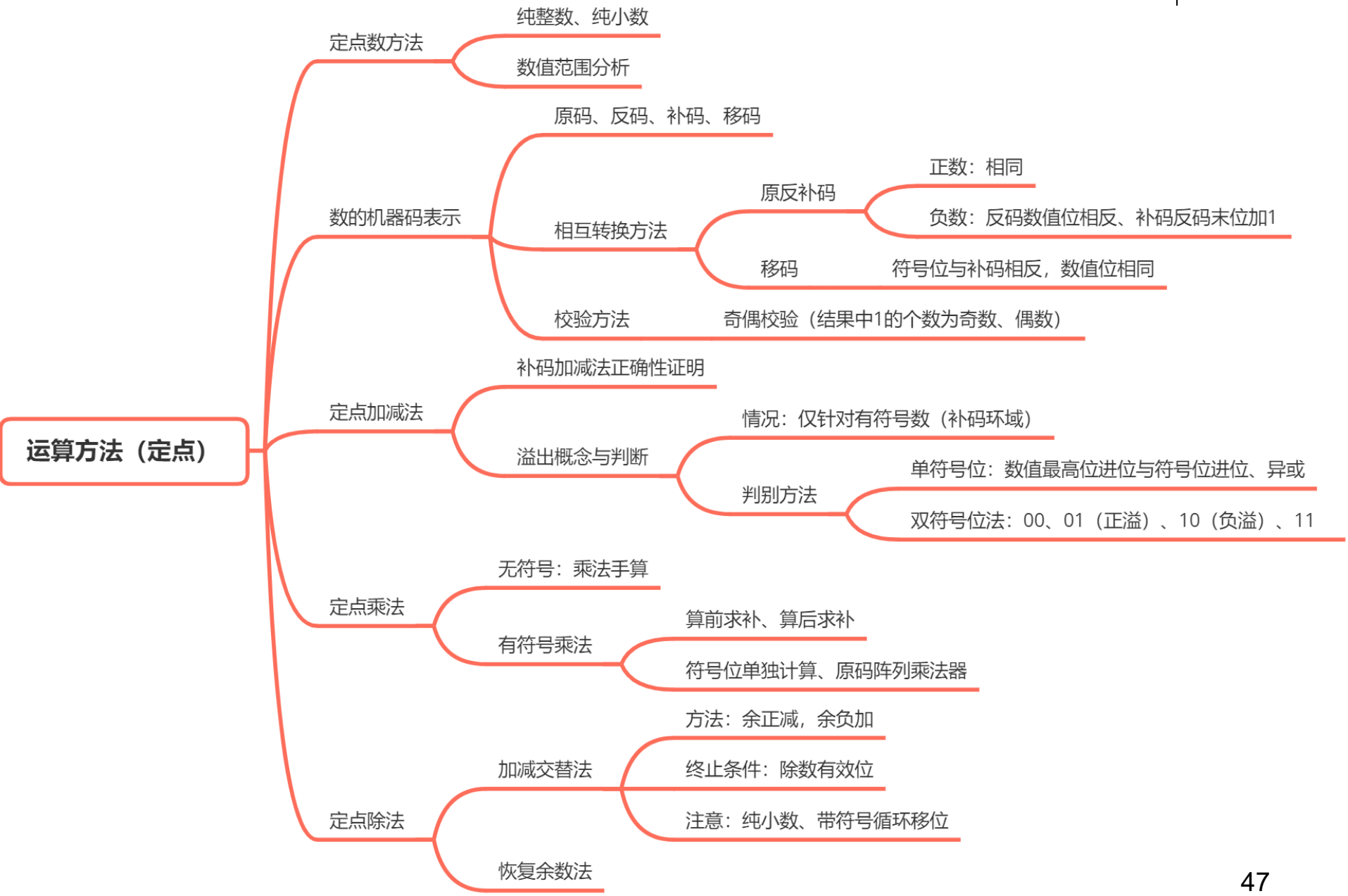


第二章总结

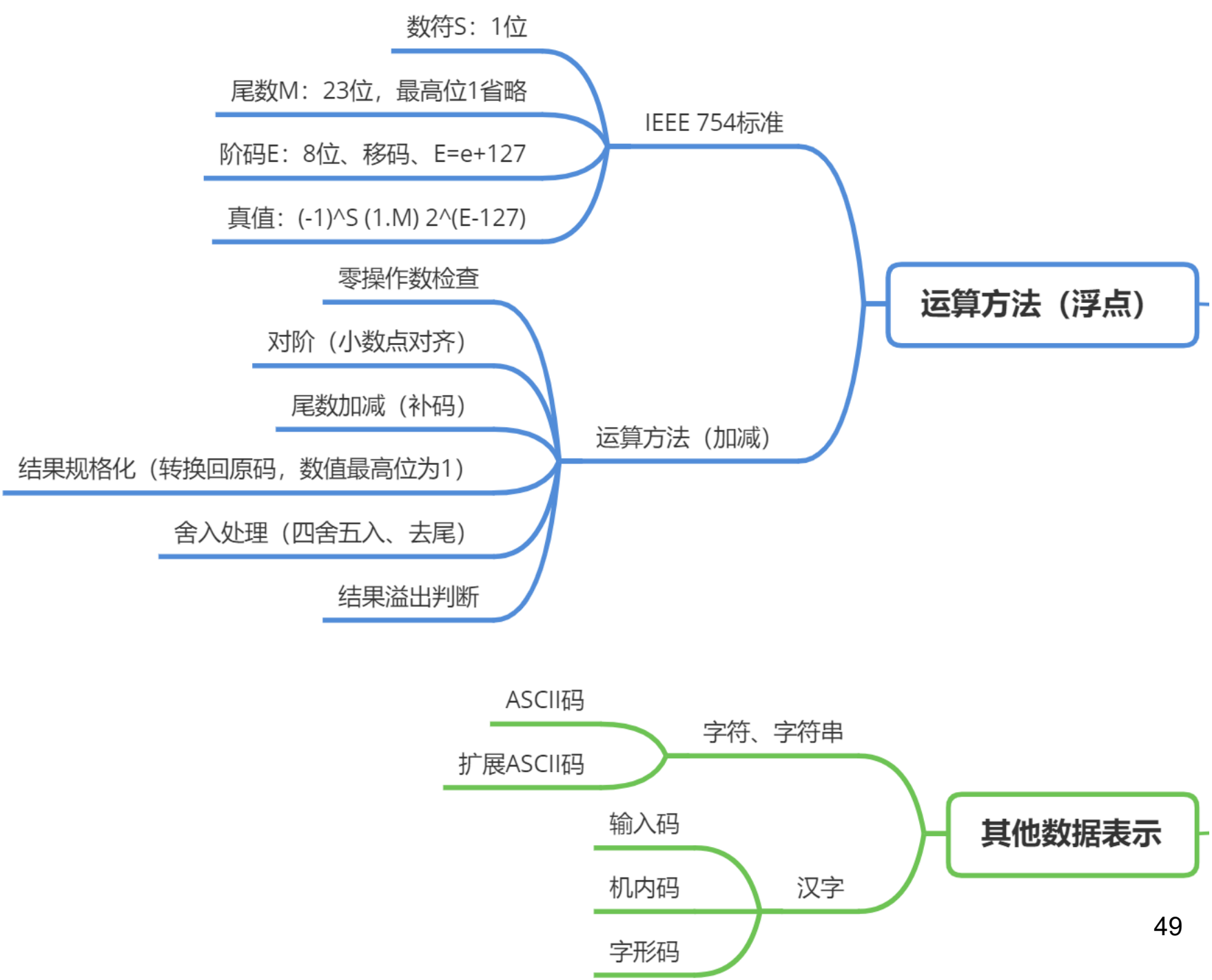
- 硬件部分：
  - 延迟分析/超前进位
  - 加减乘除法器设计
  - 总线结构
- 理论部分：
  - 原、反、补、移
  - IEEE 754 浮点数
  - 逻辑运算
  - 溢出判断
  - 定点加减乘除法
  - 浮点加减法



运算方法与运算器


运算方法（定点）
定点数方法
纯整数、纯小数
数值范围分析
数的机器码表示
原码、反码、补码、移码
相互转换方法
原反补码
正数：相同
负数：反码数值位相反、补码反码末位加1
移码
符号位与补码相反，数值位相同
校验方法
奇偶校验（结果中1的个数为奇数、偶数）
定点加减法
补码加减法正确性证明
溢出概念与判断
情况：仅针对有符号数（补码环域）
判别方法
单符号位：数值最高位进位与符号位进位、异或
双符号位法：00、01（正溢）、10（负溢）、11
定点乘法
无符号：乘法手算
有符号乘法
算前求补、算后求补
符号位单独计算、原码阵列乘法器
定点除法
加减交替法
方法：余正减，余负加
终止条件：除数有效位
注意：纯小数、带符号循环移位
恢复余数法

- **运算器（定点）**
  - 逻辑运算
  - 算术运算（加减法器）
    - 半加器、全加器
    - 行波进位、超前进位（信号产生）
    - 时延分析：关键路径
    - 加减法统一：补码（异或）
    - 溢出判断：异或（单符号位）
  - ALU
    - 函数信号发生器、74181算数逻辑单元
    - 内部总线方式：单总线、双总线、三总线
  - 定点乘除法器
    - 串行乘法、阵列乘法电路
    - 对二求补电路：（负数）从右往左扫描，最低位1左侧（除符号位）全部取反
    - 加减可控单元（CAS）、并行除法器


运算方法（浮点）
IEEE 754标准
数符S：1位
尾数M：23位，最高位1省略
阶码E：8位、移码、E=e+127
真值：(-1)^S (1.M) 2^(E-127)
运算方法（加减）
零操作数检查
对阶（小数点对齐）
尾数加减（补码）
结果规格化（转换回原码，数值最高位为1）
舍入处理（四舍五入、去尾）
结果溢出判断
其他数据表示
字符、字符串
ASCII码
扩展ASCII码
汉字
输入码
机内码
字形码