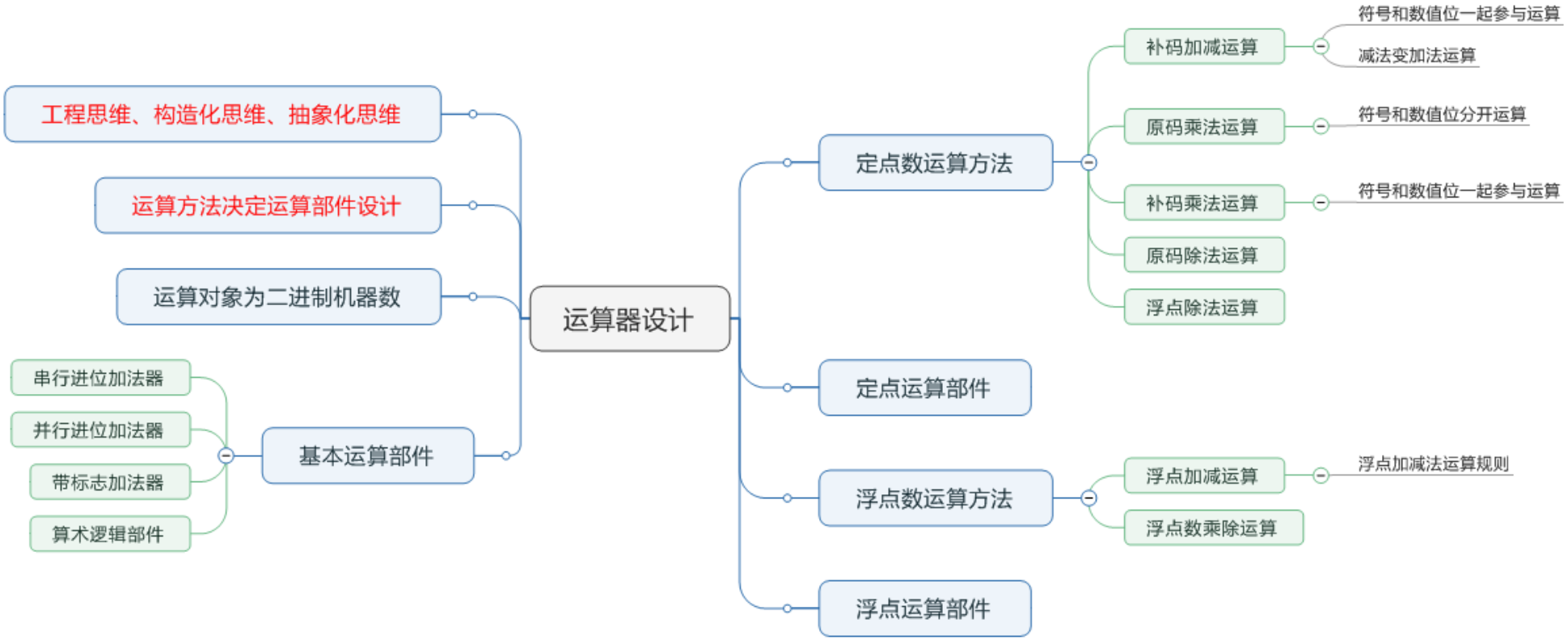
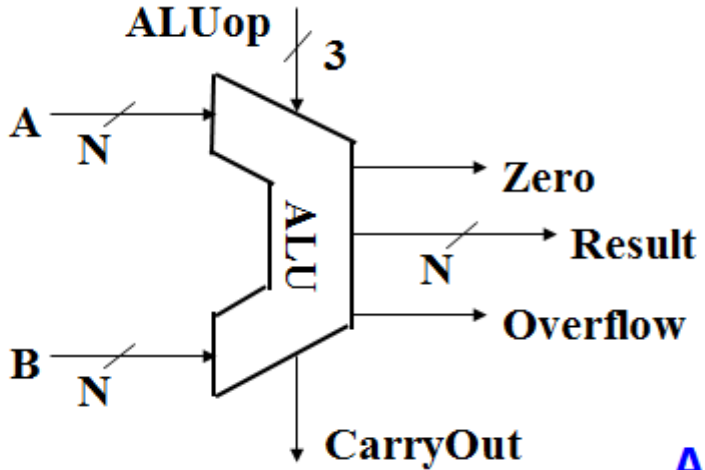
运算器设计


工程思维、构造化思维、抽象化思维
运算方法决定运算部件设计
运算对象为二进制机器数
串行进位加法器
并行进位加法器
带标志加法器
算术逻辑部件
基本运算部件
运算器设计
定点数运算方法
补码加减运算
符号和数值位一起参与运算
减法变加法运算
原码乘法运算
符号和数值位分开运算
补码乘法运算
符号和数值位一起参与运算
原码除法运算
浮点除法运算
定点运算部件
浮点数运算方法
浮点加减运算
浮点加减法运算规则
浮点数乘除运算
浮点运算部件

ALU的功能说明



ALU可进行基本的加/减算术运算和逻辑运算。其核心部件是加法器。

- ALU Control Lines (ALUop) Function
  - 000 And
  - 001 Or
  - 010 Add
  - 110 Subtract
  - 111 Set-on-less-than

n位整数加/减运算器

- 补码加减运算公式

$[A+B]_{补} = [A]_{补} + [B]_{补} \quad (\mathrm{mod}\ 2^n)$

$[A-B]_{补} = [A]_{补} + [-B]_{补} \quad (\mathrm{mod}\ 2^n)$

– 实现减法的主要工作在于：求$[-B]_{补}$

问题：如何求$[-B]_{补}$？

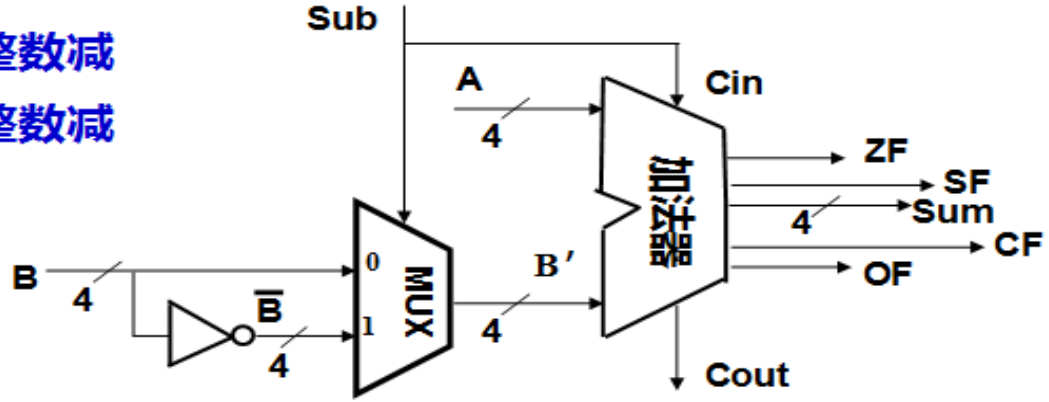
$[-B]_{补} = \overline{[B]}_{补} + 1$

- 利用带标志加法器，可构造整数加/减运算器，进行以下运算：

无符号整数加、无符号整数减

带符号整数加、带符号整数减

当Sub为1时，做减法
当Sub为0时，做加法



整数加/减运算部件

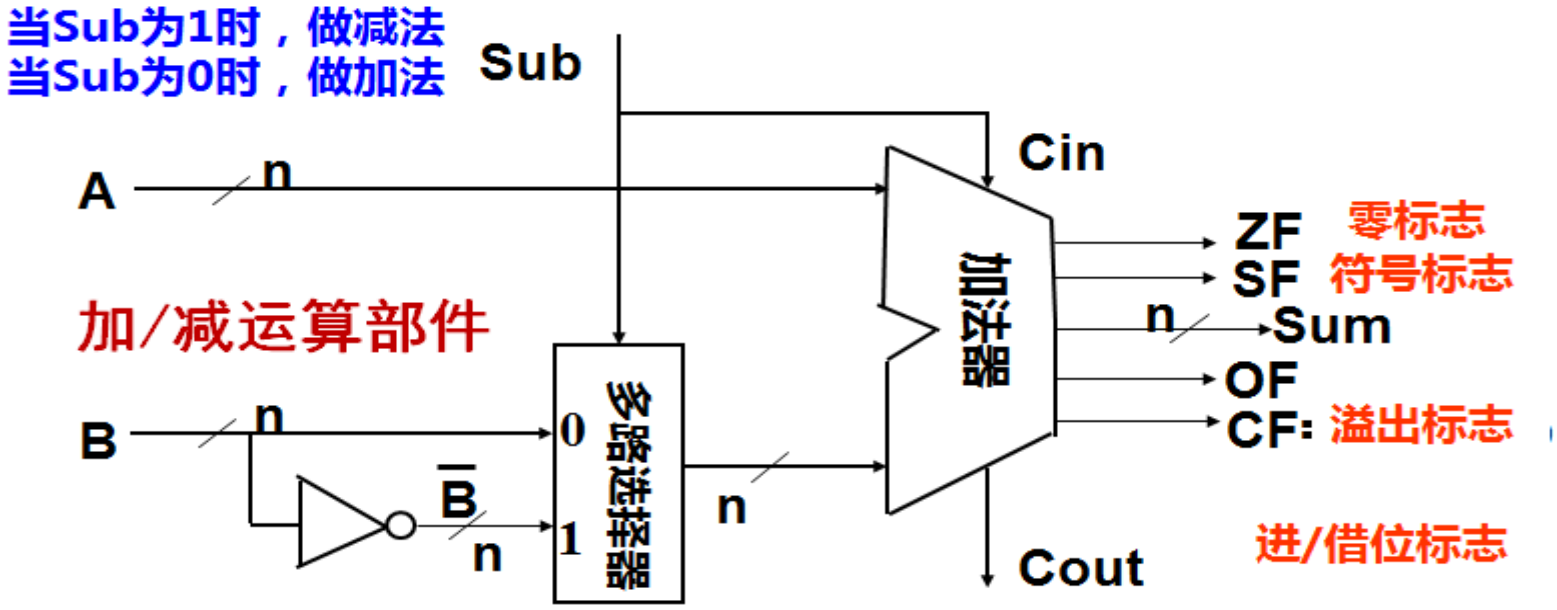
在整数加/减运算部件基础上，加上寄存器、移位器以及控制逻辑，就可实现ALU、乘/除运算以及浮点运算电路

所有运算电路的核心

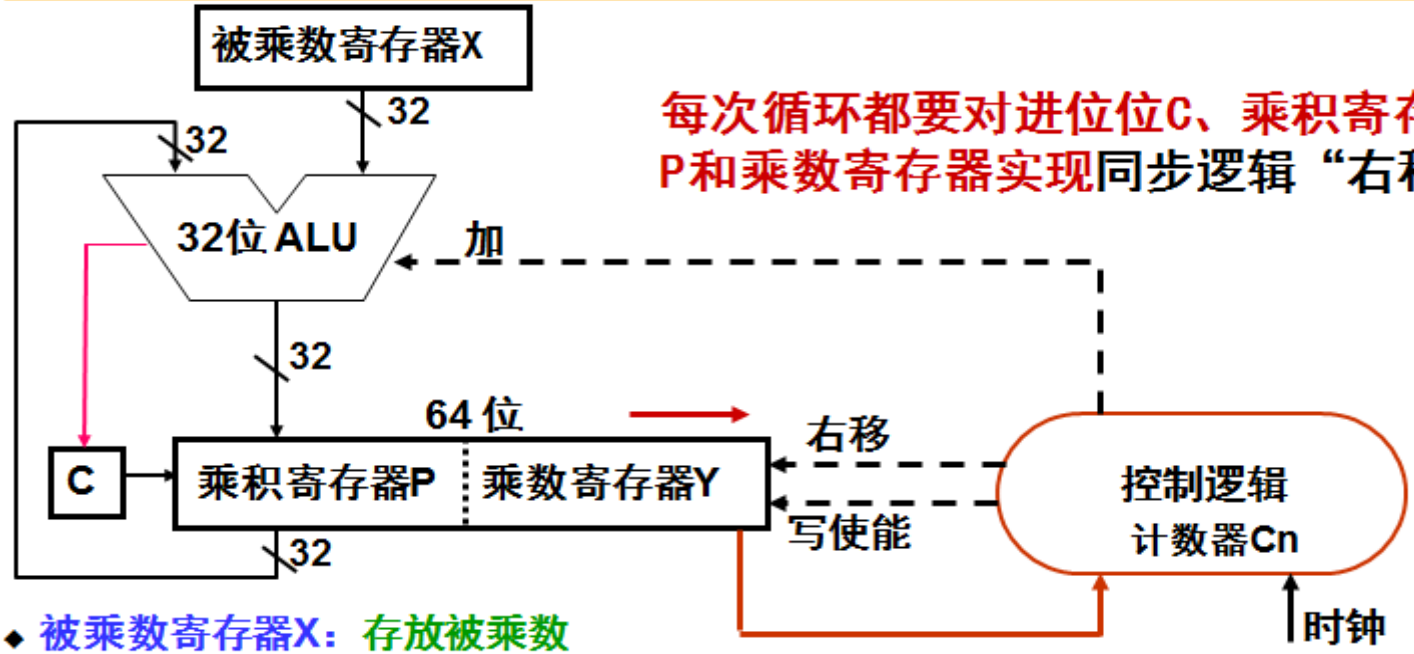
重要认识1：计算机中所有算术运算都基于加法器实现！

重要认识2：加法器不知道所运算的是带符号数还是无符号数。

重要认识3：加法器不判定对错，总是取低n位作为结果，并生成标志信息。



32位乘法运算的硬件实现



每次循环都要对进位位C、乘积寄存器P和乘数寄存器实现同步逻辑“右移”

- 被乘数寄存器X：存放被乘数
- 乘积寄存器P：开始置初始部分积P0 = 0；结束时，存放的是64位乘积的高32位
- 乘数寄存器Y：开始时置乘数；结束时，存放的是64位乘积的低32位
- 进位触发器C：保存加法器的进位信号
- 循环次数计数器Cn：存放循环次数。初值32，每循环一次，Cn减1，Cn=0时结束
- ALU：乘法核心部件。在控制逻辑控制下，对P和X的内容“加”，在“写使能”控制下运算结果被送回P，进位位在C中

Example：无符号整数乘法运算

举例说明：若需计算z=x*y；x、y和z都是unsigned类型。

设x=1110　y=1101　应用递推公式：$P_i=2^{-1}(x*y_i+P_{i-1})$

	C	乘积P	乘数R
	0	0000	1101
	+	1110	
	0	1110	1101
→	0	0111	0110
→	0	0011	1011
	+	1110	
	1	0001	1011
→	0	1000	1101
	+	1110	
	1	0110	1101
→	0	1011	0110

可用一个双倍字长的乘积寄存器；也可用两个单倍字长的寄存器。

部分积初始为0。

保留进位位。

右移时进位、部分积和剩余乘数一起进行逻辑右移。

验证：x=14, y=13, z=x*y=182

当z取4位时，结果发生溢出，因为高4位不为全0！

布斯算法举例

已知$[X]_{补}$ = 1 101，$[Y]_{补}$ = 0 110，计算$[X \times Y]_{补}$　　$[-X]_{补}$ = 0011

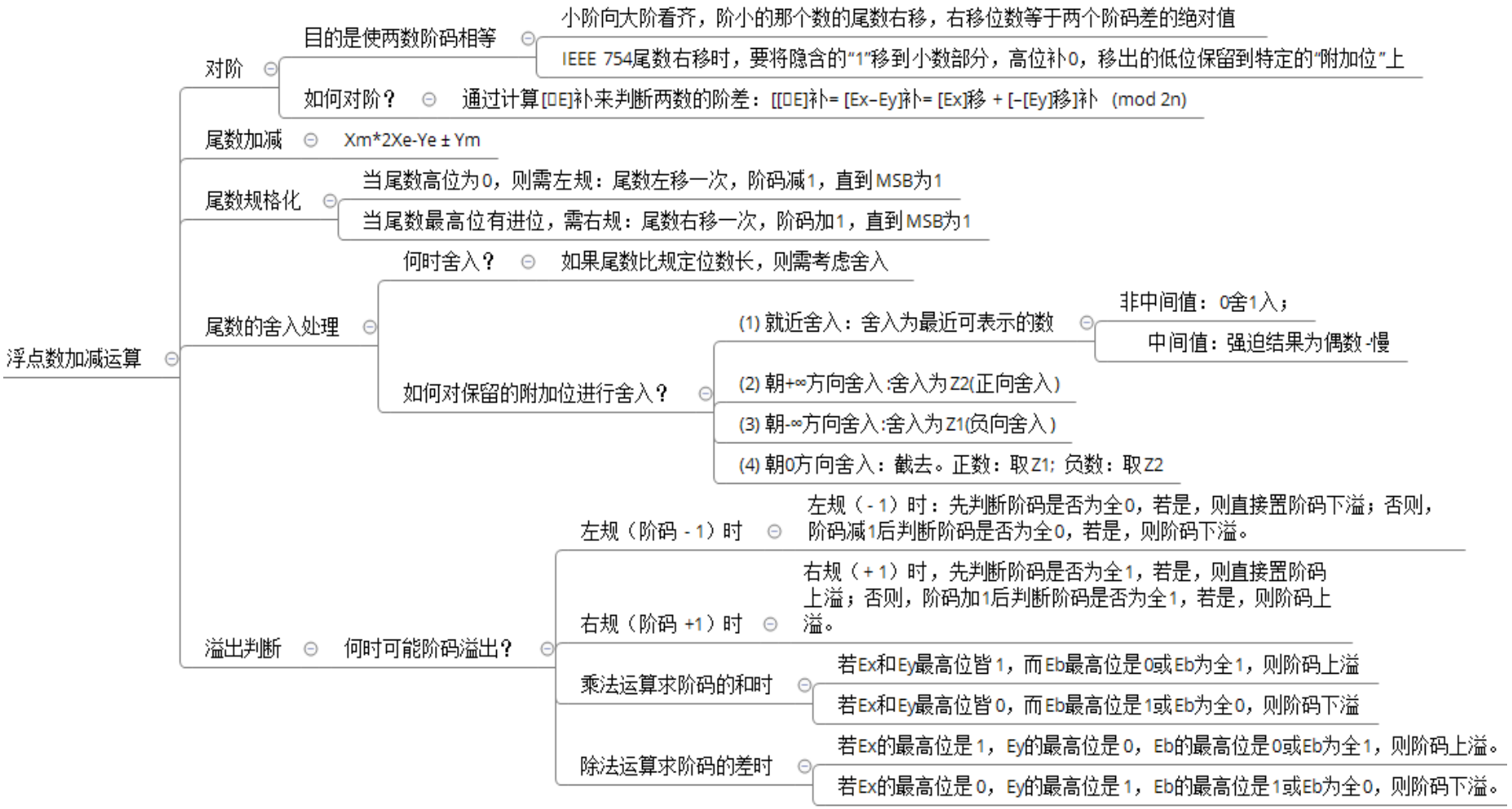
X = -3，Y = 6，X×Y=-18，$[X \times Y]_{补}$应等于11101110或结果溢出

P	Y	y_{-1}	说明
0 0 0 0	0 1 1 0	0	设 $y_{-1}=0$，$[P_0]_{补}=0$
	→ 1		$y_0\, y_{-1}=00$，P、Y 直接右移一位
0 0 0 0	0 0 1 1	0	得$[P_1]_{补}$
+ 0 0 1 1			$y_1\, y_0=10$，$+[-X]_{补}$
0 0 1 1	→ 1		P、Y 同时右移一位
0 0 0 1	1 0 0 1	1	得$[P_2]_{补}$
	→ 1		$y_2\, y_1=11$，P、Y 直接右移一位
0 0 0 0	1 1 0 0	1	得$[P_3]_{补}$
+ 1 1 0 1			$y_3\, y_2=01$，$+[X]_{补}$
1 1 0 1	→ 1		P、Y 同时右移一位
1 1 1 0	1 1 1 0	0	得$[P_4]_{补}$

如何判断结果是否溢出？

高4位是否全为符号位！

验证：当X×Y取8位时，结果 -0010010B = -18；为4位时，结果溢出！


浮点数加减运算
对阶
目的是使两数阶码相等
小阶向大阶看齐，阶小的那个数的尾数右移，右移位数等于两个阶码差的绝对值
IEEE 754尾数右移时，要将隐含的"1"移到小数部分，高位补0，移出的低位保留到特定的"附加位"上
如何对阶？
通过计算[ΔE]补来判断两数的阶差：[[ΔE]补= [Ex–Ey]补= [Ex]移 + [–[Ey]移]补 (mod 2n)
尾数加减
Xm*2Xe-Ye ± Ym
尾数规格化
当尾数高位为0，则需左规：尾数左移一次，阶码减1，直到MSB为1
当尾数最高位有进位，需右规：尾数右移一次，阶码加1，直到MSB为1
尾数的舍入处理
何时舍入？
如果尾数比规定位数长，则需考虑舍入
如何对保留的附加位进行舍入？
(1) 就近舍入：舍入为最近可表示的数
非中间值：0舍1入；
中间值：强迫结果为偶数 -慢
(2) 朝+∞方向舍入 :舍入为Z2(正向舍入)
(3) 朝-∞方向舍入:舍入为Z1(负向舍入)
(4) 朝0方向舍入：截去。正数：取Z1; 负数：取Z2
溢出判断
何时可能阶码溢出？
左规（阶码 - 1）时
左规（- 1）时：先判断阶码是否为全0，若是，则直接置阶码下溢；否则，阶码减1后判断阶码是否为全0，若是，则阶码下溢。
右规（阶码 +1）时
右规（+ 1）时，先判断阶码是否为全1，若是，则直接置阶码上溢；否则，阶码加1后判断阶码是否为全1，若是，则阶码上溢。
乘法运算求阶码的和时
若Ex和Ey最高位皆1，而Eb最高位是0或Eb为全1，则阶码上溢
若Ex和Ey最高位皆0，而Eb最高位是1或Eb为全0，则阶码下溢
除法运算求阶码的差时
若Ex的最高位是1，Ey的最高位是0，Eb的最高位是0或Eb为全1，则阶码上溢。
若Ex的最高位是0，Ey的最高位是1，Eb的最高位是1或Eb为全0，则阶码下溢。

定点数运算：由ALU + 移位器实现各种定点运算

- **移位运算**
  - 逻辑移位：对无符号数进行，左（右）边补0，低（高）位移出
  - 算术移位：对带符号整数进行，移位前后符号位不变，编码不同，方式不同。
  - 循环移位：最左（右）边位移到最低（高）位，其他位左（右）移一位。
- **扩展运算**
  - 零扩展：对无符号整数进行高位补0
  - 符号扩展：对补码整数在高位直接补符
- **加减运算**
  - 补码加/减运算：用于整数加/减运算。符号位和数值位一起运算，减法用加法实现。同号相加时，若结果的符号不同于加数的符号，则会发生溢出。
  - 原码加/减运算：用于浮点数尾数加/减运算。符号位和数值位分开运算，同号相加，异号相减；加法直接加；减法用加负数补码实现。
- **乘法运算：用加法和右移实现。**
  - 补码乘法：用于整数乘法运算。符号位和数值位一起运算。采用Booth算法。
  - 原码乘法：用于浮点数尾数乘法运算。符号位和数值位分开运算。数值部分用无符号数乘法实现。
- **除法运算：用加/减法和左移实现。**
  - 补码除法：用于整数除法运算。符号位和数值位一起运算。
  - 原码除法：用于浮点数尾数除法运算。符号位和数值位分开运算。数值部分用无符号数除法实现。

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。

ALU的实现

- 算术逻辑单元ALU：实现基本的加减运算和逻辑运算。
- 加法运算是所有定点和浮点运算（加/减/乘/除）的基础，加法速度至关重要
- 进位方式是影响加法速度的重要因素
- 并行进位方式能加快加法速度
- 通过“进位生成”和“进位传递”函数来使各进位独立、并行产生