第三章 运算方法和运算部件

第三章 运算方法和运算部件

3.8 浮点数四则运算

3.9 BCD码运算

3.10 运算部件组织

主要知识点

- 掌握定点数加减运算方法和实现的原理及溢出的判断方法
- 掌握定点数一位乘除运算方法和实现的原理
- 掌握浮点数四则运算方法
- 掌握运算部件组成原理

3.1 概述

1.运算器的分类

（1）串行运算器和并行运算器

（2）定点运算器和浮点运算器

（3）二进制运算器和十进制运算器

2.运算器的主要指标

（1）机器字长

3.1 概述

机器字长—运算器一次能处理的二进制位数称为机器字长。

（2）运算速度

① 普通法

是以完成一次加法或一次乘法运算所需的时间,或用每秒能完成算术运算的平均次数作为运算速度。

3.1 概述

② 吉布森 (Gibson)法

$$Tm = \sum_{i=1}^{n} fiti$$

fi为第i种指令的执行频度
ti为第i种指令的执行时间

3.1 概述

③基准法

用同一个程序，在不同的机器上运行所需的时间，为该机器的运算速度。

④MIPS

MIPS是Million Instructions Per Second的缩写，每秒处理的百万级的机器语言指令数。这是衡量CPU速度的一个指标。

3.1 概述

$$MIPS = \frac{CLOCK}{CPI \times 10^6}$$

CPI—每条指令的平均时钟周期数

3.2 定点加减运算

1.定点数的加减运算

计算机中基本采用补码，补码加减运算基本关系：

$(X+Y)_补=X_补+Y_补$

$(X-Y)_补=X_补+(-Y)_补$

运算规则：

（1）参与运算的操作数用补码表示，符号位作为数的一部分直接参与运算，运算结果为补码表示

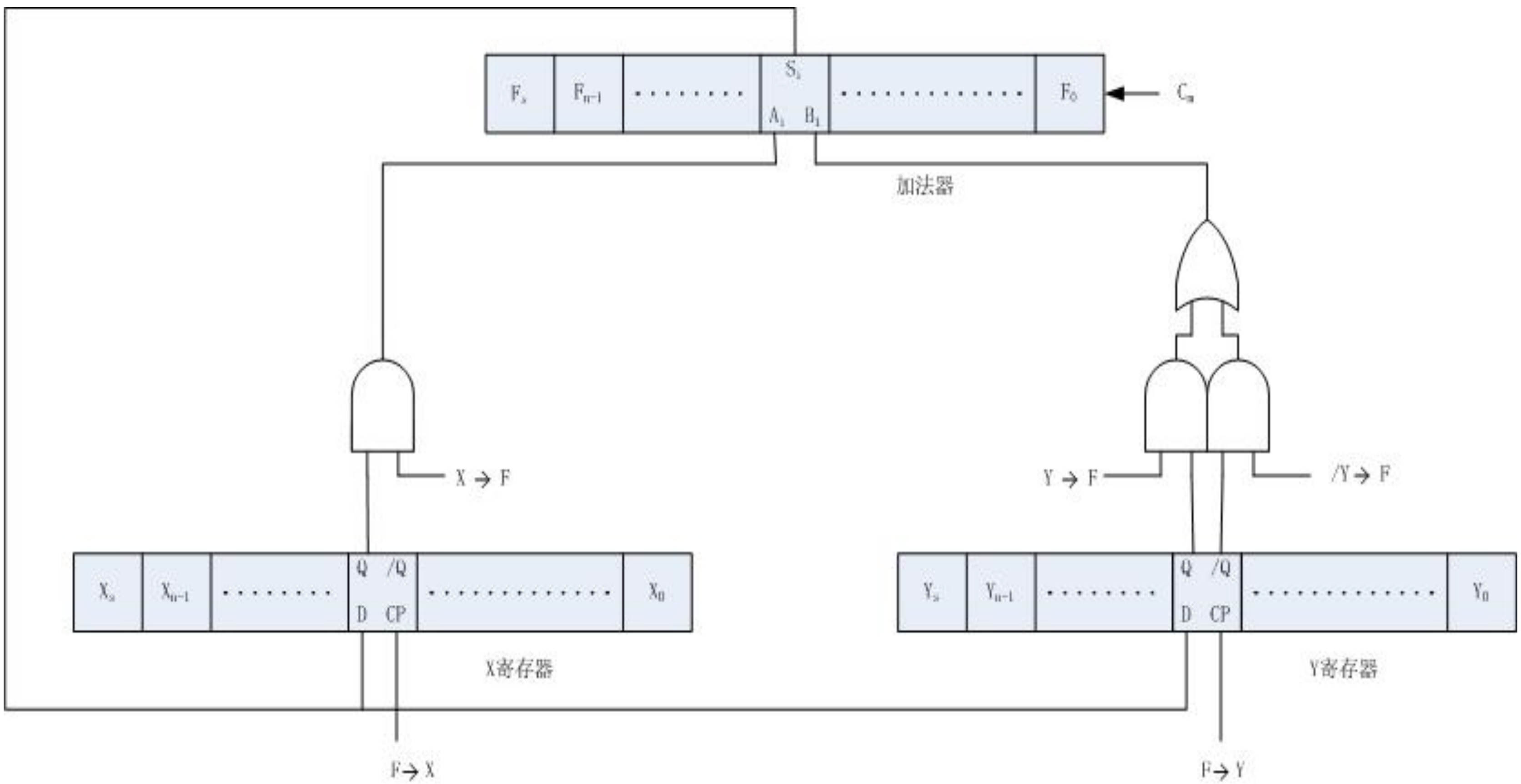
3.2 定点加减运算

（2）如操作为加，则两数直接相加。

（3）如操作为减，则将减数变补后再与被减数相加。

实现补码加减运算逻辑电路

3.2 定点加减运算



Fn
Fn-1
Si
Ai
Bi
F0
Cn
加法器
X → F
Y → F
/Y → F
Xn
Xn-1
Q /Q
D CP
X0
Yn
Yn-1
Y0
X寄存器
Y寄存器
F → X
F → Y

3.2 定点加减运算

2.溢出判别

当运算的结果超出允许的表示范围，则产生溢出，正超出称为正溢出，负超出称为负溢出。

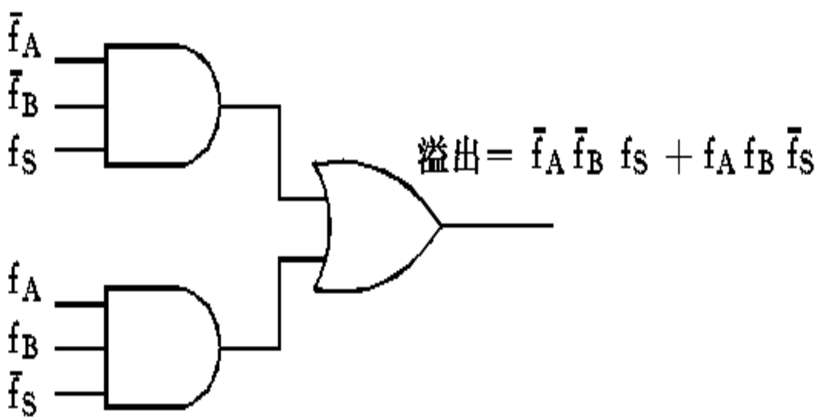
3.2 定点加减运算

（1）溢出判断逻辑一

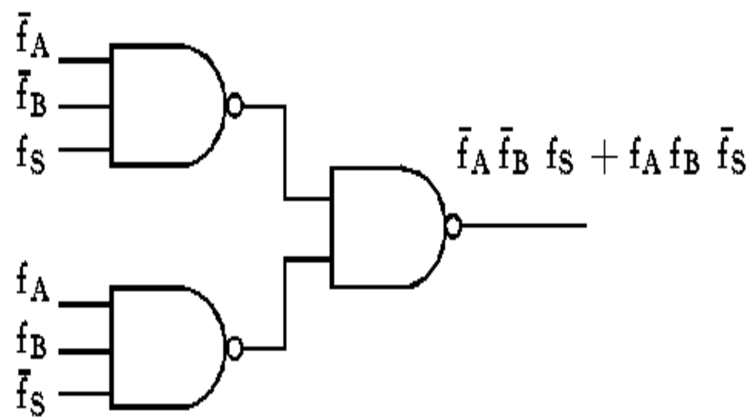
$$溢出 = \overline{F_a}\,\overline{F_b}F_s + F_aF_b\overline{F_s} = 1$$

Fa，Fb—加数，被加数符号位

Fs—运算和符号位



(a)



(b)

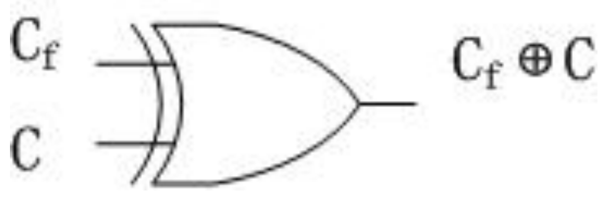
3.2 定点加减运算

（2）溢出判断逻辑二

溢出 $= C_f \oplus C = 1$

C_f—符号位运算产生的进位

C —最高有效位产生的进位



3.2 定点加减运算

（3）溢出判断逻辑三

单符号位的信息量只能表示两种可能，数为正或为负，如产生溢出，就会使符号位含义产生混乱。将符号位扩充到二位，采用变形补码（或称模4补码），就能通过符号位直接判断是否溢出。变形补码定义：

$$[X]_{变形补}=\begin{cases} X & 0\leqslant X\leqslant 2^{n-1}-1 \\ 2^{n+1}-|X| & -2^{n-1}\leqslant X<0 \end{cases}$$

3.2 定点加减运算

变形补码用00表示正，11表示负。

计算结果符号位为：

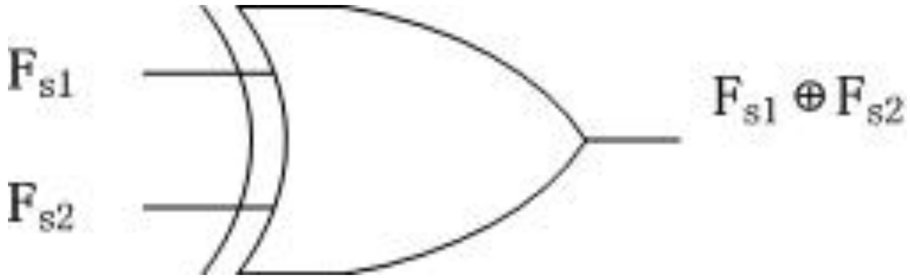
00—结果为正，无溢出；

11—结果为负，无溢出；

01—结果为正溢出；

10—结果为负溢出；

溢出 $= F_{s1} \oplus F_{s2}$



3.2 定点加减运算

注意：数据在主存中仍为单符号，运算时传送到运算器时扩充成双符号，运算结束后紧缩成单符号位存入主存中。

3.3 定点原码一位乘法

两个原码数相乘

●数值则为两数绝对值之积$P=|X|\times|Y|$

●其乘积的符号为相乘两数符号的异或值

$Sp=Sx\oplus Sy$

算法:

① 根据乘数最末一位判断

最末位是1,加被乘数,右移一位

最末位是0, 右移一位

② 如乘数为n位,则需要进行n次累加移位

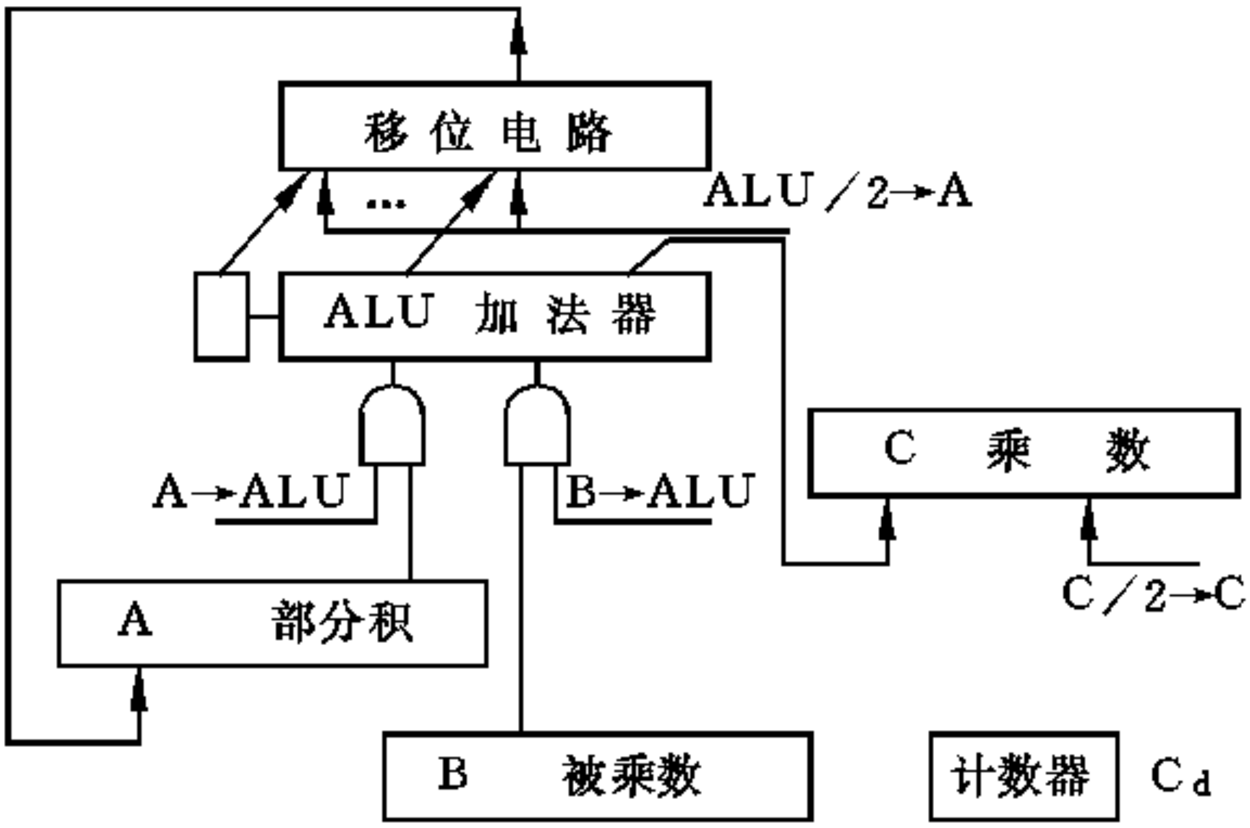
3.3 定点原码一位乘法

回忆算术中，竖式乘法的计算过程

$$
\begin{array}{r}
0.1011 \\
\times \quad 0.1101 \\
\hline
0.1011 \\
0.0000 \\
0.1011 \\
0.1011 \\
0.0000 \\
\hline
0.10001111
\end{array}
$$

3.3 定点原码一位乘法

原码一位乘法逻辑图



3.4 定点补码一位乘法

设 $[x]_补 = x_0.x_1x_2\ldots x_n, [y]_补 = y_0.y_1y_2\ldots y_n$ ，

$$[x \cdot y]_补 = [x]_补 (0.y_1y_2\cdots y_n) - [x]_补 \cdot y_0$$

3.4 定点补码一位乘法

运算规则（参考教材第48页）

① 符号位参加运算,被乘数是两个符号位,乘数是一个符号位。

② 在乘数的最后一位加个0

③ 判断乘数最末两位

00	部分积右移一位
11	部分积右移一位
01	部分积加被乘数右移一位
10	部分积减被乘数右移一位

最后一步只加减不移位

④ 移位按补码方式移位，即当部分积最高位为0时，移入0，为1时，移入1。

3.4 定点补码一位乘法

设 $[x]_{补} = x_0.x_1x_2...x_n, [y]_{补} = y_0.y_1y_2...y_n$ ，将符号位参与计算，运算数以补码表示且被乘数与部分积都取双符号位。

Booth算法的基本流程：

符号位参与与计算，运算数以补码表示且被乘数与部分积都取双符号位。

记部分积初值为0，乘数末位增加附加位 ，初值置0

根据 (y_n, y_{n+1}) 的取值，按下表循环执行n+1次，得到了n次对部分积的累加，并且最后一次部分积不右移，此时执行了n次右移

Yn	Yn+1	操作
0	0	+0，右移一位
0	1	+[x]补，右移一位
1	0	+[-x]补，右移一位
1	1	+0，右移一位

3.4 定点补码一位乘法

下面给出一个示例：设机器字长为5位，其中一位是符号位。且

$x=-0.1101, y=+0.1011$，利用Booth算法求 $P=x\cdot y$.

解：$[x]_补=11.0011$，$[-x]_补=00.1101$，$[y]_补=0.1011$。Booth 算法的求解过程如下。

	(高位部分积)	(低位部分积/乘数)	说明
	00.0000	0.1011 0 丢失位	起始情况
$+[-x]_补$	00.1101		$Y_4Y_5=10, Y_5-Y_4=-1$, 则+[-x]补
	00.1101		
右移	00.0110	10.101 10	右移部分积和乘数
+0	00.0000		$Y_3Y_4=11, Y_4-Y_3=0$, 则+0
	00.0110		
右移	00.0011	010.10 110	右移部分积和乘数
$+[x]_补$	11.0011		$Y_2Y_3=01, Y_3-Y_2=1$, 则+[x]补
	11.0110		
右移	11.1011	0010.1 0110	右移部分积和乘数
$+[-x]_补$	00.1101		$Y_1Y_2=10, Y_2-Y_1=-1$, 则+[-x]补
	00.1000		
右移	00.0100	00010. 10110	右移部分积和乘数
$+[x]_补$	11.0011		$Y_0Y_1=01, Y_1-Y_0=1$, 则+[x]补
	11.0111	构成$[x\cdot y]_补$	

所以$[x\cdot y]_补=1.01110001$，得 $x\cdot y=-0.10001111$。

3.4 定点补码一位乘法

原理

由于乘法计算的本质就是加法的累加，因此当乘数的二进制代码中“含1量”过高时，必然会出现大量频繁的加法计算，但事实上这并**不是必要的。**

回忆小学时，我们曾经做过如下的简便计算：

$$9 \times 99 = 9 \times (100 - 1) = 900 - 9 = 891$$

这就是一种化简方法，在二进制 中同样有类似的**化繁为简**的妙用，例如：

$00111110 \ = 010000 - 10$. 注意，这里不能看成是**用010000减去10**，而应该看成二进制的第二位**取-1**（负1）。

显然如果用010000-10作为乘数的话，比起前者（00111110）机器需要做5次累加来说，计算量减小到了2次。但是第一次是做**减法**而不是加法，因此对于**补码**来说才更容易实现。

那么问题又来了，我们该如何“一眼看出”00111110 可以化为010000-10呢？

3.4 定点补码一位乘法

我们记正乘数 $Y = (y_0 . y_1 y_2 \cdots y_n)_2$ ，所以有 $y_0 = 0$ ，因此:

$$\begin{aligned} Y &= y_1 2^{-1} + y_2 2^{-2} + \cdots + y_{n-1} 2^{-(n-1)} + y_n 2^{-n} \\ 2Y &= y_1 2^{0} + y_2 2^{-1} + y_3 2^{-2} + \cdots + y_n 2^{-(n-1)} \\ 2Y - Y = Y &= y_1 2^{0} + (y_2 - y_1) 2^{-1} + \cdots + (y_n - y_{n-1}) 2^{-(n-1)} + (0 - y_n) 2^{-n} \end{aligned}$$

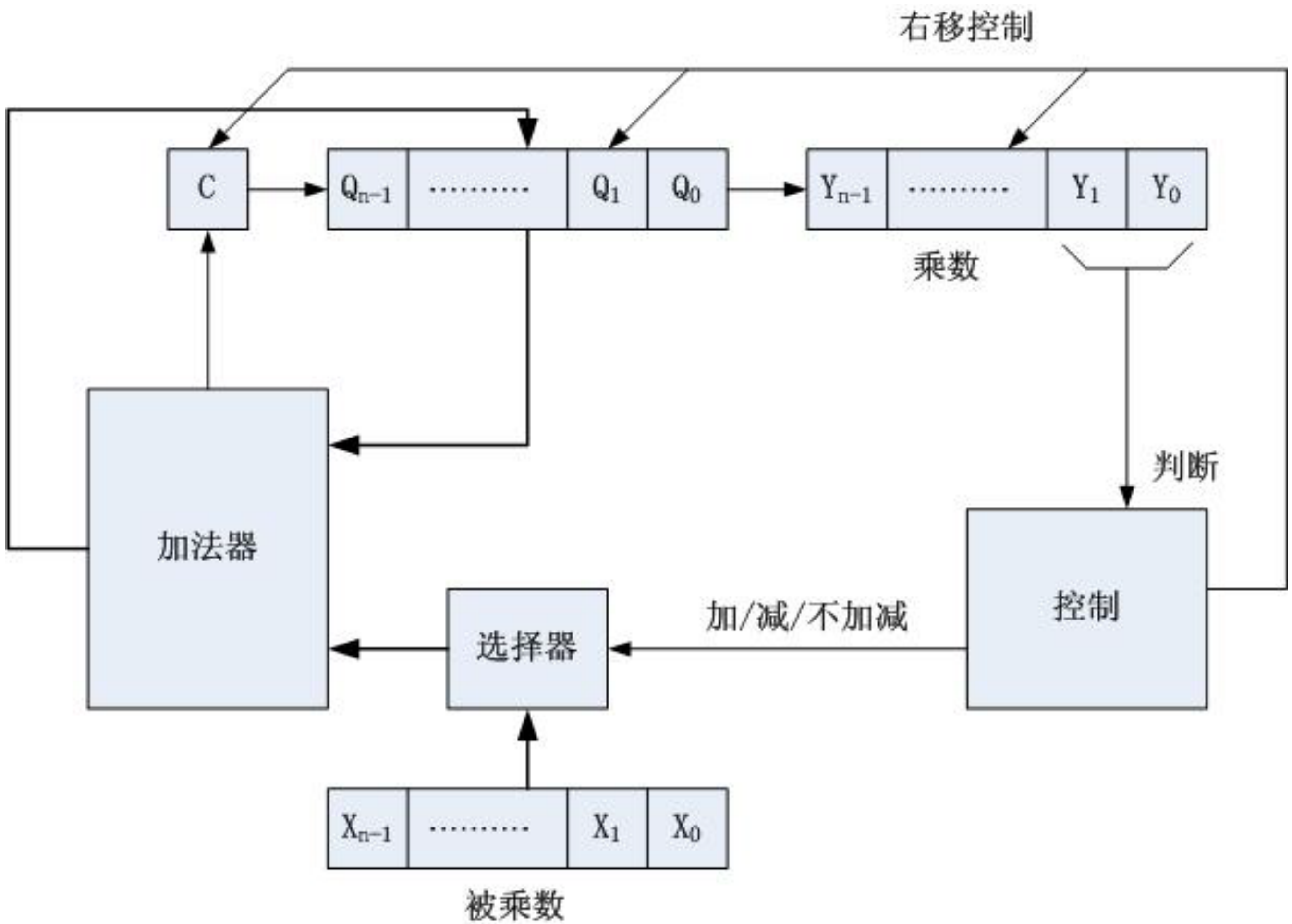
取 $y_{n+1} = y_0 = 0$, 则第一项和最后一项就分别为 $(y_1 - y_0)2^0, (y_{n+1} - y_n)2^{-n}$.

由于 $y_i \in \{0, 1\}$ ，所以有:

1. 当 $y_{i+1} = y_i$ 时，第i项取0
2. 当 $y_{i+1} > y_i$ 时，第i项取1
3. 当 $y_{i+1} < y_i$ 时，第i项取-1

Yn	Yn+1	操作
0	0	+0，右移一位
0	1	+[x]补，右移一位
1	0	+[-x]补，右移一位
1	1	+0，右移一位

3.4 定点补码一位乘法



3.4 定点补码一位乘法

课堂练习：

$x=-0.1101, y=-0.1011$，利用Booth算法求 $P=x\cdot y$.

3.5 定点原码一位除法

运算规则:

① 商的符号位同定点数原码乘法的处理方法：$Sp=Sx\oplus Sy$

② 两数的绝对值部分进行相除

3.5 定点原码一位除法

加减交替算法：

① 被除数先减除数，结果（余数）为正商上1；
结果（余数）为负商上0；

② 结果（余数）左移一位

③ 如上次商为1 时，则被除数减除数；
如上次商为0 时，则被除数加除数；
结果（余数）为正商上1；
结果（余数）为负商上0；

3.5 定点原码一位除法

设机器字长为5位（含1位符号位，$n=4$），$x=0.1011$，$y=0.1101$，求x/y

$(0.1011\times 2^4)\div(0.1101\times 2^4)$

$$
\begin{array}{r}
0.1101 \\
01101\,\overline{\big)\,01011} \\
00000 \\
\hline
10110 \\
01101 \\
\hline
10010 \\
01101 \\
\hline
01010 \\
00000 \\
\hline
10100 \\
01101 \\
\hline
0111
\end{array}
$$

$$
\begin{array}{l}
\quad\ \ 0.1101 \\
0.1101\,\overline{\big)\,0.1011} \\
\quad\ \ 0.0000 \\
\hline
\quad\ \ 0.10110 \\
\quad\ \ 0.01101 \\
\hline
\quad\ \ 0.010010 \\
\quad\ \ 0.001101 \\
\hline
\quad\ \ 0.0001010 \\
\quad\ \ 0.0000000 \\
\hline
\quad\ \ 0.00010100 \\
\quad\ \ 0.00001101 \\
\hline
\quad\ \ 0.00000111
\end{array}
$$

3.5 定点原码一位除法

设机器字长为5位（含1位符号位，n=4），x=0.1011，y=0.1101，采用原码恢复余数法求x/y

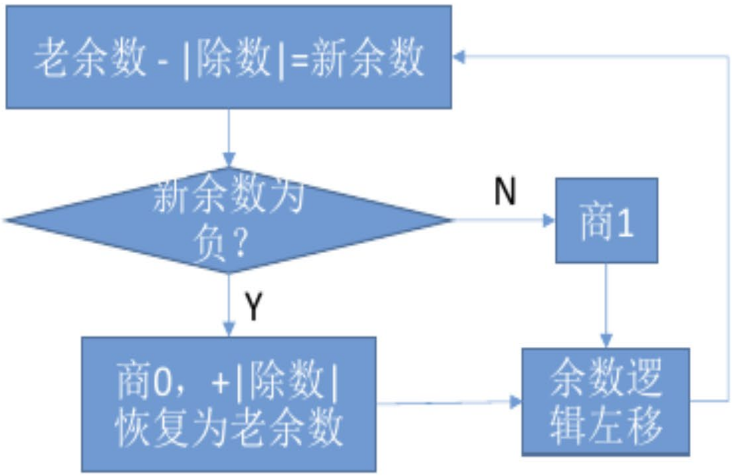
$|x|$=0.1011，$|y|$=0.1101，$[|y|]_{补}$=0.1101，$[-|y|]_{补}$=1.0011

	被除数/余数	商
	0.1011	
$+[-\lvert y\rvert]_{补}$	1.0011	
	1.1110	0
$+[\lvert y\rvert]_{补}$	0.1101	
	0.1011	
左移	1.0110	
$+[-\lvert y\rvert]_{补}$	1.0011	
	0.1001	01
左移	1.0010	
$+[-\lvert y\rvert]_{补}$	1.0011	
	0.0101	011
	…	…

余数为负，就要商0，并恢复余数

逻辑左移

余数为正，就要商1，不用恢复余数



左移n次，上商n+1次

最后一次上商余数不左移

3.5 定点原码一位除法

设机器字长为5位（含1位符号位，n=4），x=0.1011，y=0.1101，采用原码恢复余数法求x/y

$|x|$=0.1011，$|y|$=0.1101，$[|y|]_{补}$=0.1101，$[-|y|]_{补}$=1.0011

	被除数/余数		商
	0.1011		
$+[-\|y\|]_{补}$	1.0011		
	1.1110	a	0
$+[\|y\|]_{补}$	0.1101	b	
	0.1011	a+b	
左移	1.0110	(a+b)×2 = 2a + 2b	
$+[-\|y\|]_{补}$	1.0011	(a+b)×2 - b = 2a + 2b – b = 2a + b	
	0.1001		01
左移	1.0010		
$+[-\|y\|]_{补}$	1.0011		
	0.0101		011
	…		…

余数a为负

若余数为负，则可直接商0，并让余数左移1位再加上|除数|

3.5 定点原码一位除法

设机器字长为5位（含1位符号位，n=4），x=0.1011，y=0.1101，采用原码加减交替除法求x/y

$|x|$=0.1011，$|y|$=0.1101，$[|y|]_{补}$=0.1101，$[-|y|]_{补}$=1.0011

$Q_s=x_s \oplus y_s=0 \oplus 0=0$
得x/y=+0.1101
余0.0111×2^{-4}

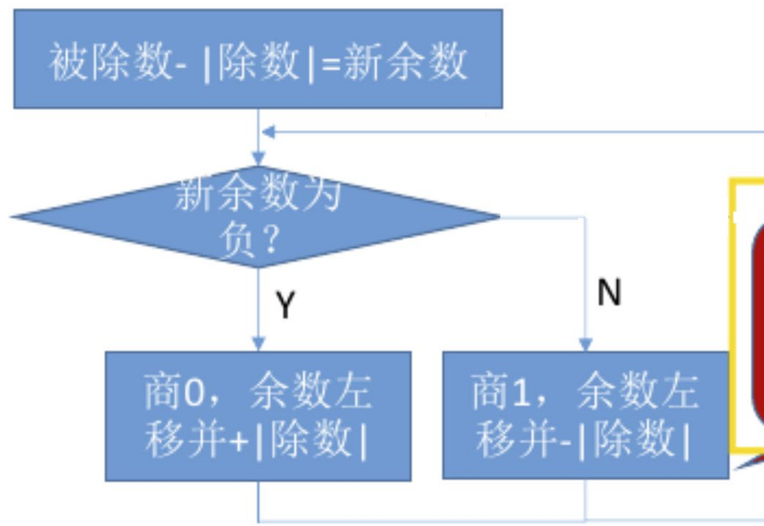
若余数为负，则可直接商0，让余数左移1位再加上|除数|，得到下一个新余数

若余数为正，则商1，让余数左移1位再减去|除数|，得到下一个新余数

	被除数/余数
	0.1011
$+[-|y|]_{补}$	1.0011
	$\boxed{1.1110}$
左移	1.1100
$+[|y|]_{补}$	0.1101
	$\boxed{0.1001}$
左移	1.0010
$+[-|y|]_{补}$	1.0011
	0.0101
左移	0.1010
$+[-|y|]_{补}$	1.0011
	1.1101
左移	1.1010
$+[|y|]_{补}$	0.1101
	0.0111

若余数为负，需商0，并$+[|y|]_{补}$得到正确余数

商	ACC	MQ
	01011	00000
0	11110	00000
	11100	00000



之后每次根据余数的正负性来确定加/减

3.6 定点补码一位除法

介绍加减交替法

运算规则：

如被除数与除数同号，先减除数
如被除数与除数异号，先加除数
然后：
① 余数和除数同号，商上1，余数左移一位，下次减除数
② 余数和除数异号，商上0，余数左移一位，下次加除数

百年同济
TONGJI UNIVERSITY

3.6 定点补码一位除法

设机器字长为5位（含1位符号位，n=4），x=+0.1000，y=-0.1011，采用补码加减交替除法求x/y

$[x]_{补}$=00.1000，$[y]_{补}$=11.0101，$[-y]_{补}$=00.1011　　$[x/y]_{补}$=1.0101，余0.0111×2^{-4}

	被除数/余数
	00.1000
+$[y]_{补}$	11.0101
	11.1101
左移	11.1010
+$[-y]_{补}$	00.1011
	00.0101
左移	00.1010
+$[y]_{补}$	11.0101
	11.1111
左移	11.1110
+$[-y]_{补}$	00.1011
	00.1001
左移	01.0010
+$[y]_{补}$	11.0101
	00.0111

ACC	MQ
001000	00000
111101	00001
111010	00010
000101	00010
001010	00100
111111	00101
111110	01010
001001	01010
010010	10100
000111	10101

精度误差不超过 2^{-n}

末位商恒置1

补码除法：
- 符号位参与运算
- 被除数/余数、除数采用双符号位

被除数和除数同号，则被除数减去除数；
异号则被除数加上除数。

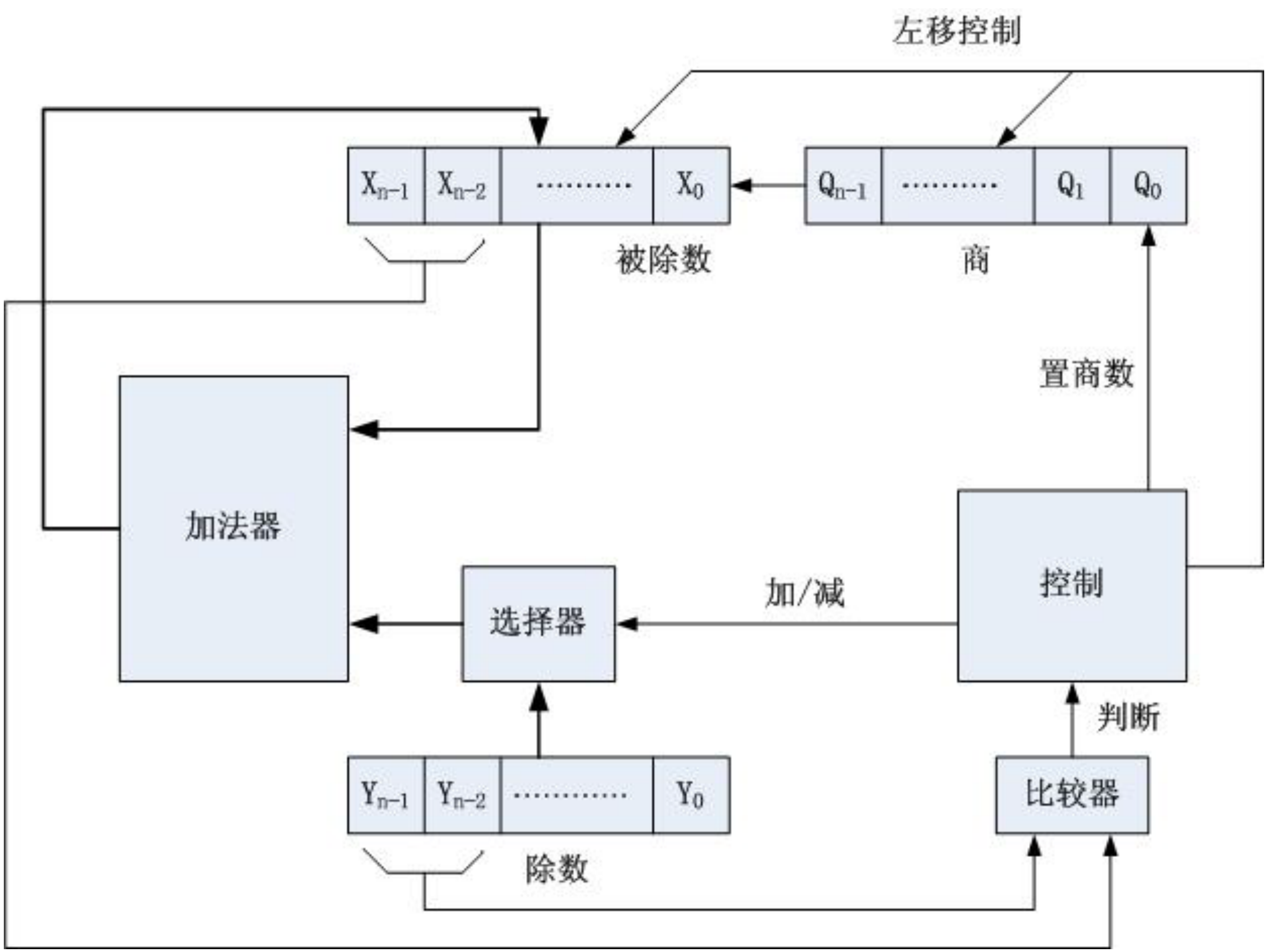
余数和除数同号，商1，余数左移一位减去除数；
余数和除数异号，商0，余数左移一位加上除数。
重复n次

3.6 定点补码一位除法

原码加减交替法和补码加减交替法对比：

<table>
<tr><th rowspan="2">除法类型</th><th rowspan="2">符号位参与运算</th><th rowspan="2">加减次数</th><th colspan="2">移位</th><th rowspan="2">上商、加减原则</th><th rowspan="2">说明</th></tr>
<tr><th>方向</th><th>次数</th></tr>
<tr><td>原码加减交替法</td><td>否</td><td>N+1或N+2</td><td>左</td><td>N</td><td>余数的正负</td><td>若最终余数为负，需恢复余数</td></tr>
<tr><td>补码加减交替法</td><td>是</td><td>N+1</td><td>左</td><td>N</td><td>余数和除数是否同号</td><td>商末位恒置1</td></tr>
</table>

3.6 定点补码一位除法



左移控制
X_{n-1}
X_{n-2}
X_0
Q_{n-1}
Q_1
Q_0
被除数
商
置商数
加法器
控制
加/减
选择器
判断
Y_{n-1}
Y_{n-2}
Y_0
比较器
除数

3.6 定点补码一位除法

课堂练习：

$$x=-0.1011\,,y=-0.1101$$

用补码加减交替法求 x/y

3.7 逻辑运算的实现

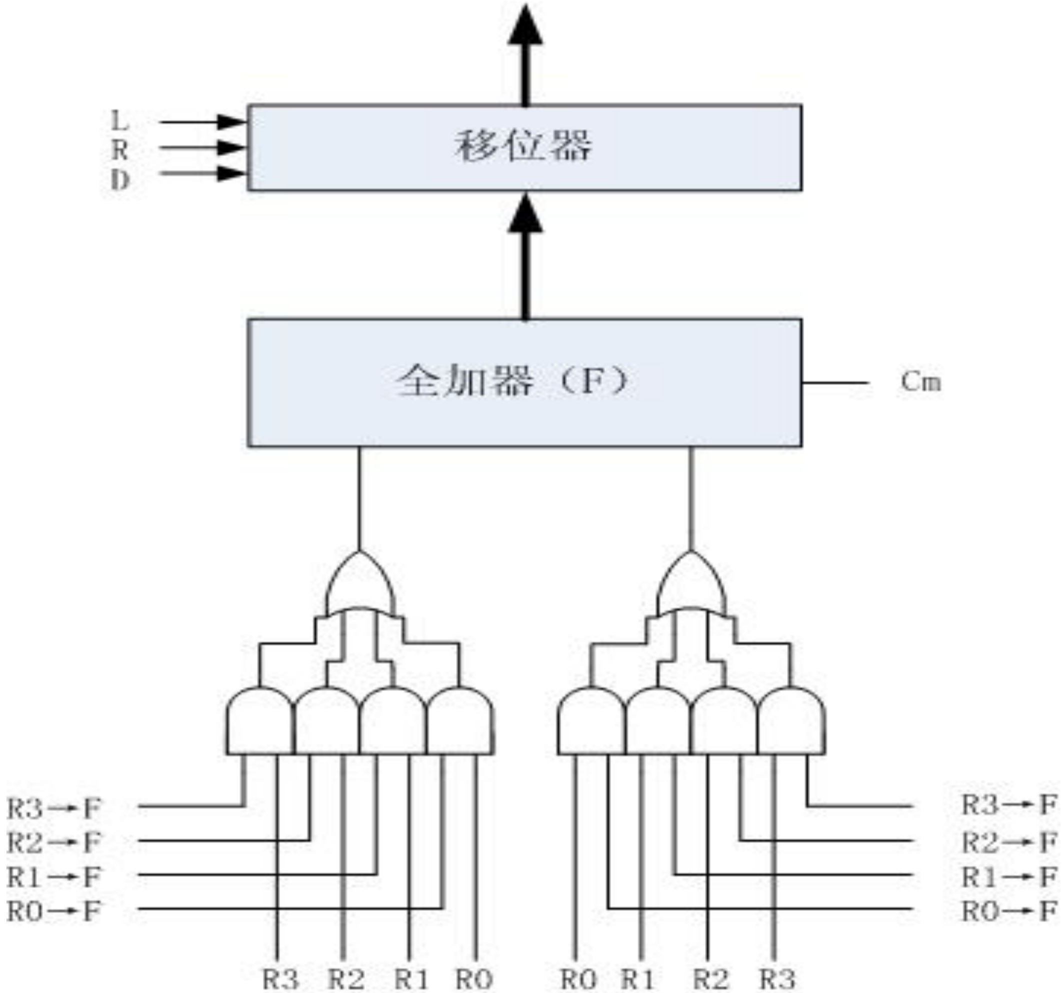
1.逻辑运算

基本的逻辑运算是求反、或、与、异或等，这些逻辑运算是通过在原有加法器上再附加部分线路实现的。

2.移位操作

一般由移位器实现，移位器放在全加器输出端。

3.7 逻辑运算的实现



L
R
D
移位器
全加器（F）
Cm
R3→F
R2→F
R1→F
R0→F
R3 R2 R1 R0
R0 R1 R2 R3
R3→F
R2→F
R1→F
R0→F

3.8 浮点运算

3.8.1 浮点数的加减法运算

设有两浮点数X、Y,实现X±Y运算，其中：$X = M_X \times 2^{Ex}$; $Y = M_Y \times 2^{Ey}$。均为规格化数。执行以下五步完成运算。

(1)“对阶”操作

对阶的规则：

阶码小的数向阶码大的数对齐，即：阶码小的数的尾数右移并相应增大阶码。

3.8 浮点运算

比较两浮点数阶码的大小，求出其差ΔE，并保留其大值E，$E=max(E_X,E_Y)$。

当ΔE≠0时，将阶码值小的数的尾数右移ΔE位，并将其阶码值加上ΔE，使两数的阶码值相等，这一操作称之为“对阶”。尾数右移时:

对原码表示的尾数，符号位不参加移位，尾数数值部分的高位补0；

对补码表示的尾数，符号位参加右移，并保持原符号位不变。

3.8 浮点运算

(2) 尾数的加/减运算

执行对阶后，两尾数进行加/减运算，得到两数之和/差。

(3) 规格化操作

规格化的目的是使尾数部分的绝对值尽可能以最大值的形式出现。设尾数M的数值部分有n位，规格化数的范围为：$1/2 \le |M_{原}| \le 1-2^{-n}$， $1/2 \le [M]_{补} \le 1-2^{-n}$(当M为正)，$1/2 \le |[M]_{补}| \le 1$(当M为负)。

3.8 浮点运算

当运算的结果(和/差)不是规格化数时，需将它转变成规格化数。

规格化操作的规则是：

① 如果结果的两个符号位的值不同

（01.XX····X ，10.XX····X），表示加/减运算尾数结果溢出，此时将尾数结果右移1位，阶码E+1，称为“向右规格化”，简称“右规”。

3.8 浮点运算

② **如果结果的两个符号位的值相同**，表示加/减运算尾数结果不溢出。但若最高数值位与符号位相同，此时尾数连续左移，直到最高数值位与符号位的值不同为止；同时从E中减去移位的位数，这称之为“向左规格化”，简称“左规”。

(4) 舍入

在执行右规或对阶时，尾数低位上的数值会移掉，使数值的精度受到影响，常用“0”舍“1”入法。

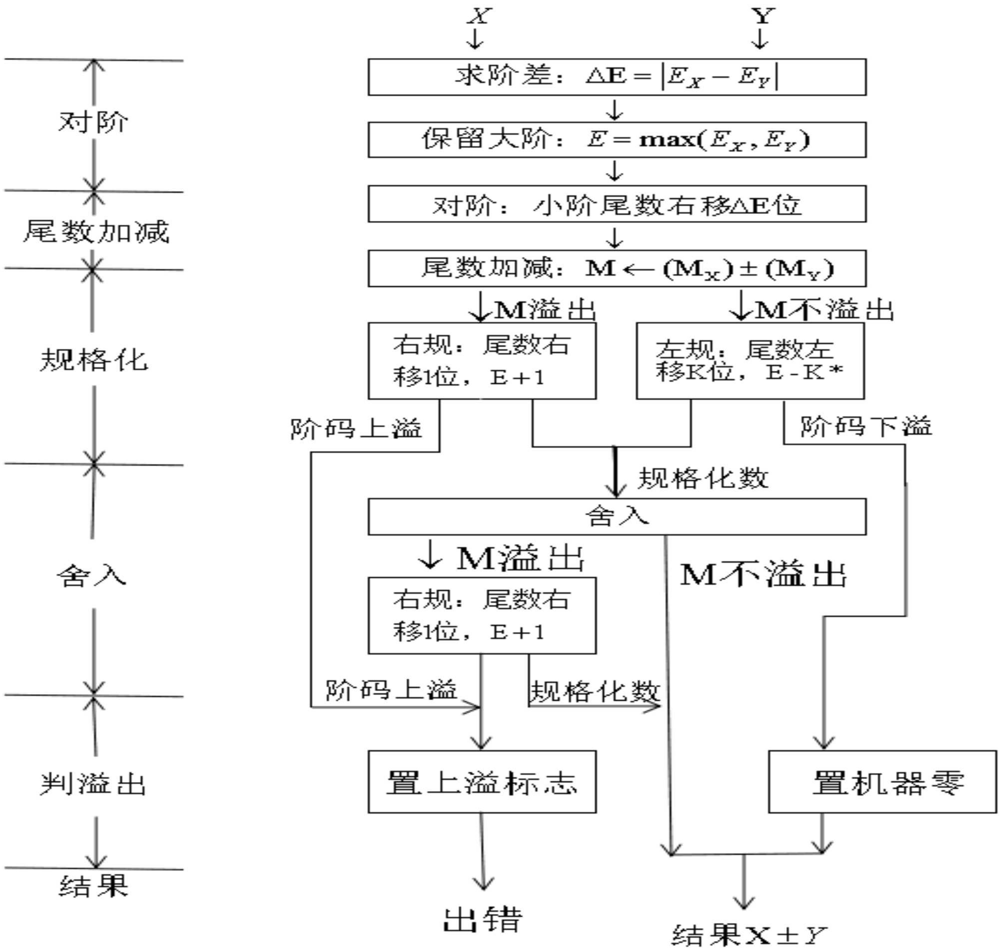
3.8 浮点运算

(5) 检查阶码是否溢出

阶码溢出表示浮点数溢出。在规格化和舍入时都可能发生溢出，若阶码正常，加/减运算正常结束。若阶码下溢，则置运算结果为机器零，若上溢，则置溢出标志。

下图为规格化浮点数加减运算流程。

3.8 浮点运算



*如果已为规格化数，则K＝0，不移位。

3.8 浮点运算

例：两浮点数相加，求X+Y。

已知： $X=0.11011011\times 2^{010}$

$Y=-0.10101100\times 2^{100}$

解：计算过程：

① 对阶操作

阶差$\Delta E=[E_X]_补+[-E_Y]_补$

$=00010+11100=11110$

X阶码小，M_X右移2位，保留阶码E=00100。

$[M_X]_补=00\ 00\ 110\ 110\ \underline{11}$

下划线上的数是右移出去而保留的附加位。

3.8 浮点运算

② 尾数相加

$[M_X]_{补}+[M_Y]_{补}$

$=00001101\underline{11}+1101010100=1110001010\ \underline{11}$

③ 规格化操作

左规，移1位，结果$=1100010101\ \underline{10}$;阶码减1，E=00011。

④ 舍入

附加位最高位为1，在所得结果的最低位+1，得新结果：$[M]_{补}=1100010110$,M=-0.11101010。

3.8 浮点运算

⑤ 判溢出

阶码符号位为00，故不溢出，最终结果为：

$X+Y=-0.11101010\times 2^{011}$

3.8 浮点运算

3.8.2 浮点数的乘除法运算

两浮点数相乘，其乘积的阶码为相乘两数阶码之和，其尾数应为相乘两数的尾数之积。两个浮点数相除，商的阶码为被除数的阶码减去除数的阶码得到的差，尾数为被除数的尾数除以除数的尾数所得的商。参加运算的两个数都为规格化浮点数。乘除运算都可能出现结果不满足规格化要求的问题，因此也必须进行规格化、舍入和判溢出等操作。规格化时要修改阶码。

3.8 浮点运算

1. 浮点数的阶码运算

阶码有+1，-1，两阶码求和以及两阶码求差四种运算，还要检查结果是否溢出。在计算机中，阶码通常用补码或移码形式表示。补码运算规则和判定溢出的方法，已在前面说明。这里讨论移码的运算规则和判定溢出的方法。

当阶码由1位符号位和n位数据组成时，其移码的定义为： $[X]_{移}=2^n+X \quad -2^n \le X < 2^n$

按此定义，则有 $[X]_{移}+[Y]_{移}$

$=2^n+X+2^n+Y=2^n+(2^n+(X+Y))=2^n+[X+Y]_{移}$

3.8 浮点运算

即直接用移码实现求阶码之和时，结果的最高位多加了个1，要得到移码形式的结果，需对结果的符号取反。

根据补码定义： $[Y]_{补}=2^{n+1}+Y \mod 2^{n+1}$

对同一个数值，移码和补码的数值位完全相同，而符号位正好相反。因此求阶码和(移码表示)可用如下方式完成：

$$[X]_{移}+[Y]_{补}=2^{n}+X+2^{n+1}+Y$$
$$=2^{n+1}+(2^{n}+(X+Y))=[X+Y]_{移} \mod 2^{n+1}$$

同理有 $[X]_{移}+[-Y]_{补}=[X-Y]_{移}$。

3.8 浮点运算

以上表明执行移码加或减时，取加数或减数符号位的反码进行运算。

如果阶码运算的结果溢出，上述条件则不成立。此时，使用双符号位的阶码加法器，并规定移码的第二个符号位，即最高符号位恒用0参加加减运算，则溢出条件是结果的最高符号位为1。此时，当低位符号位为0时，表明结果上溢，为1时，表明结果下溢。当最高符号位为0时，表明没有溢出，低位符号位为1，表明结果为正，为0时，表明结果为负。

3.8 浮点运算

2. 浮点数的舍入处理

在计算机中，浮点数的尾数是用确定的位数来表示的，但浮点数的运算结果却常常超过给定的位数。如加减运算过程中的对阶和右规处理，会使尾数低位部分的一位或多位的值丢失；乘除运算(无论是定点数或浮点数)也可有更多位数的结果，在这里讨论如何处理这些多出来的位上的值。处理的原则是使本次处理所造成的误差以及按此原则产生的累计误差都比较小。

3.8 浮点运算

① 无条件地丢掉正常尾数最低位之后的全部数值。这种办法被称为截断处理，其好处是处理简单，缺点是影响结果的精度。

② 运算过程中保留右移中移出的若干高位的值，然后再按某种规则用这些位上的值修正尾数。这种处理方法被称为舍入处理。较简单的舍入方法是：只要尾数最低位为1，或移出去的几位中有1，就把尾数的最低位置1，否则仍保持原有的0值。或者采用更简便的方法，即最低位恒置1的方法。

3.8 浮点运算

③ 0舍1入法(相当于十进制中的四舍五入法)，即当丢失的最高位的值为1时，把这个1加到最低数值位上进行修正，否则舍去丢失的各位的值，其缺点是要多进行一次加法运算。下面举例说明0舍1入情况。

3.8 浮点运算

例：设有5位数(其中有二附加位)，用原码或补码表示，舍入后保留4位结果。

设： $[X]_{原}=0.110110$ 舍入后 $[X]_{原}=0.1110$

$[X]_{原}=0.111001$ 舍入后 $[X]_{原}=0.1110$

$[X]_{补}=1.001010$ 舍入后 $[X]_{补}=1.0011$

$[X]_{补}=1.001001$ 舍入后 $[X]_{补}=1.0010$

舍入后产生了误差，但误差值小于末位的权值。

3.8 浮点运算

3. 浮点乘法运算步骤

① 检测操作数是否为0；

② 两数阶码相加，求积的阶码；

③ 两数尾数相乘，求积的尾数；

④ 尾数乘积规格化

⑤ 判断阶码有无溢出。

3.8 浮点运算

例： 阶码4位(移码)，尾数8位(补码，含1符号位)，阶码以2为底。运算结果仍取8位尾数。

设： $X=2^{-5}\times 0.1110011$, $Y=2^{3}\times(-0.1110010)$

X，Y为真值，此处阶码用十进制表示，尾数用二进制表示。运算过程中阶码取双符号位。

3.8 浮点运算

(1) 求乘积的阶码。乘积的阶码为两数阶码之和。

$[E_X+E_Y]_{移} = [E_X]_{移} + [E_Y]_{补}$

$=00011+00011=00110$

(2) 尾数相乘。用定点数相乘的办法，

$[X \times Y]_{补}=1.0011001\ 1001010$ (尾数部分)

(3) 规格化处理。本例尾数已规格化，不需要再处理。如未规格化，需左规。

3.8 浮点运算

(4) 舍入。尾数(乘积)低位部分的最高为1，需要舍入，在乘积高位部分的最低位加1，因此：

$[X\times Y]_{补}=1.0011010$ (尾数部分)

(5) 判溢出。阶码未溢出，故结果为正确。

$X\times Y=2^{-2}\times(-0.1100110)$

在求乘积的阶码(即两阶码相加)时，有可能产生上溢或下溢的情况；在进行规格化处理时，有可能产生下溢。

3.8 浮点运算

4. 浮点数乘法运算(阶码的底为8或16)

为了用相同位数的阶码表示更大范围的浮点数，在一些计算机中也有选用阶码的底为8或16的。此时浮点数N被表示成：

$$N=8^{E}\cdot M \quad 或 \quad N=16^{E}\cdot M$$

阶码E和尾数M还都是用二进制表示的，其运算规则，与阶码以2为底基本相同，但关于对阶和规格化操作有新的相应规定。

3.8 浮点运算

当阶码以8为底时，只要尾数满足1/8≤M<1或-1≤M<-1/8就是规格化数。执行对阶和规格化操作时，每当阶码的值增或减1，尾数要相应右移或左移三位。

当阶码以16为底时，只要尾数满足1/16≤M<1或-1≤M<-1/16就是规格化数。执行对阶和规格化操作时，阶码的值增或减1，尾数必须移四位。

判别为规格化数或实现规格化操作，均应使数值的最高三位(以8为底)或四位(以16为底)中至少有一位与符号位不同。

3.8 浮点运算

5. 浮点数除法运算步骤：

① 检测操作数是否为0；

② 尾数调整，检测被除数的尾数的绝对值是否小于除数的尾数的绝对值，如果不是，将被除数的尾数右移一位，并相应调整阶码;

③ 被除数阶码减除数阶码;

④ 被除数尾数除以除数尾数;

⑤ 尾数规格化。

3.9 BCD码运算

1.BCD码加法运算

用BCD码进行相加时，按二进制方法进行，但要对“和”进行加6修正。

修正条件：

① 某位和大于9时，加6修正。

② 低位向高位产生进位时，加6修正。

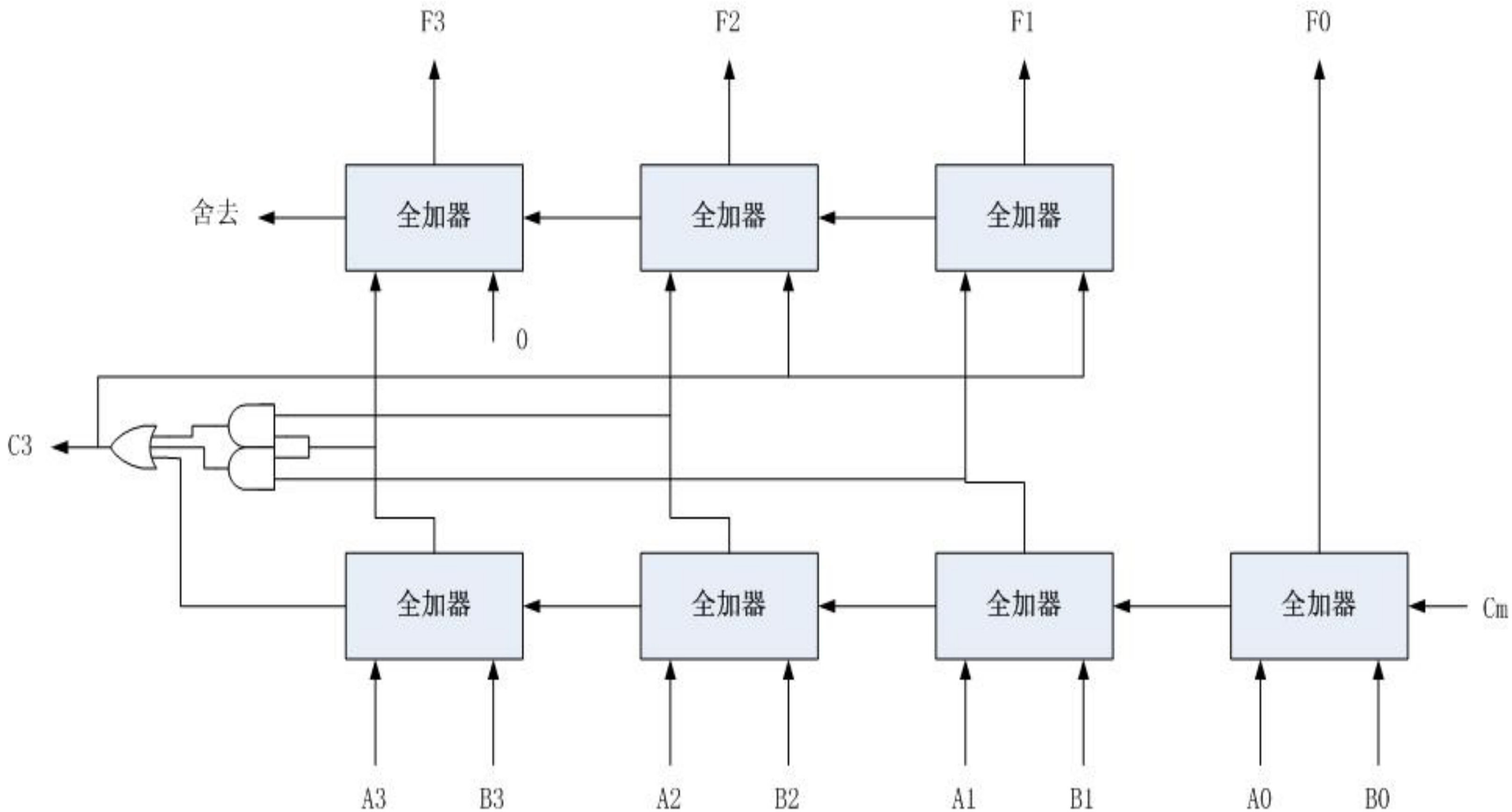
2.BCD码减法运算

用BCD码进行相减时，按二进制方法进行，但要对“差”进行减6修正。

修正条件：

低位向高位产生借位时，减6修正。

3.9 BCD码运算


F3
F2
F1
F0
舍去
全加器
全加器
全加器
0
C3
全加器
全加器
全加器
全加器
Cm
A3
B3
A2
B2
A1
B1
A0
B0

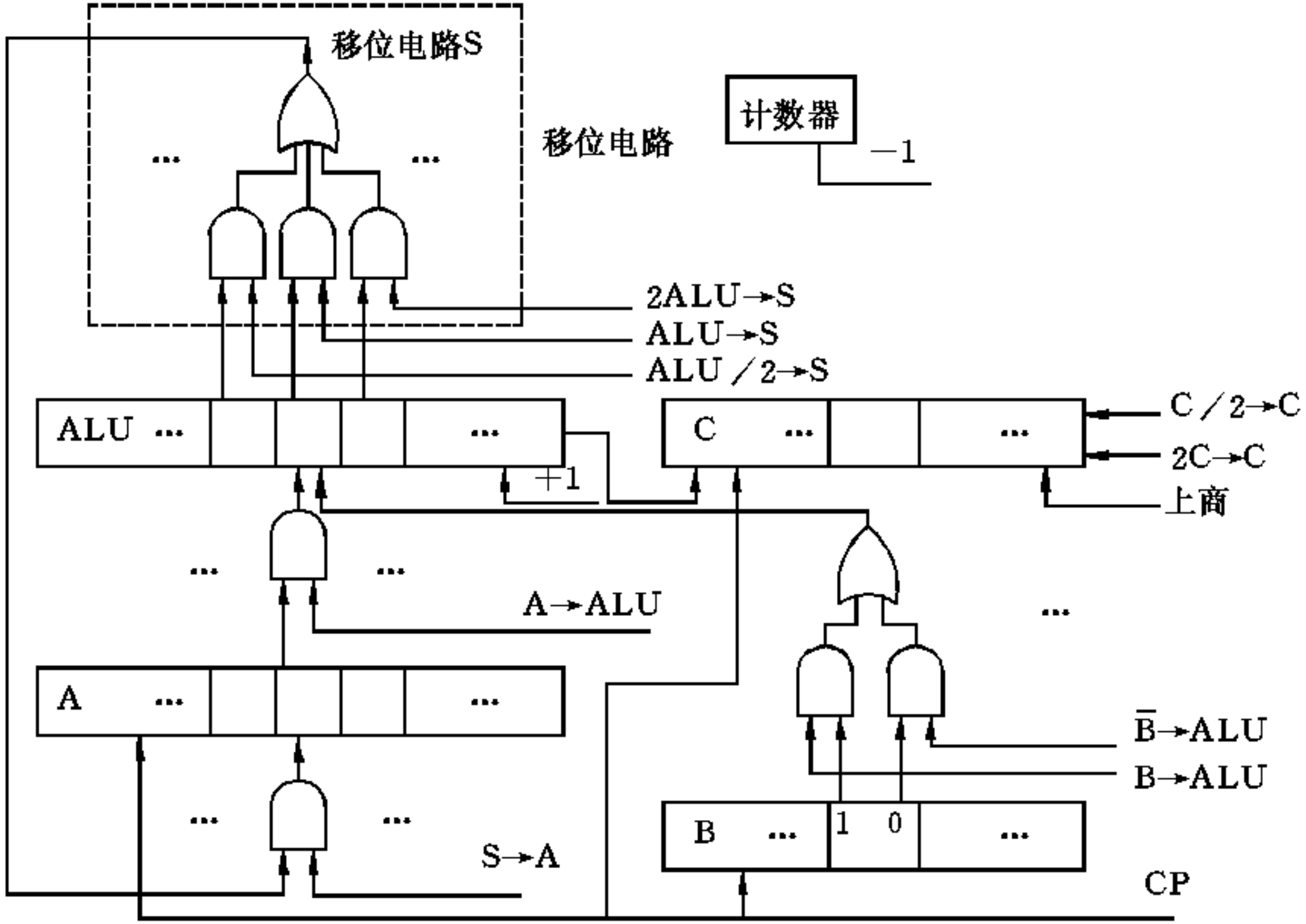
3.10 运算部件组织

1. 定点运算部件

定点运算部件由算术逻辑运算部件ALU、若干个寄存器、移位电路、计数器、门电路等组成。ALU部件主要完成加减法算术运算及逻辑运算，其中还应包含有快速进位电路。

下图为定点运算部件的框图，图中仅有三个寄存器(A，B，C)，而目前一般的运算部件都设置有数量较多的寄存器，可任意放置操作数和运算结果等，称之为通用寄存器。

3.10 运算部件组织



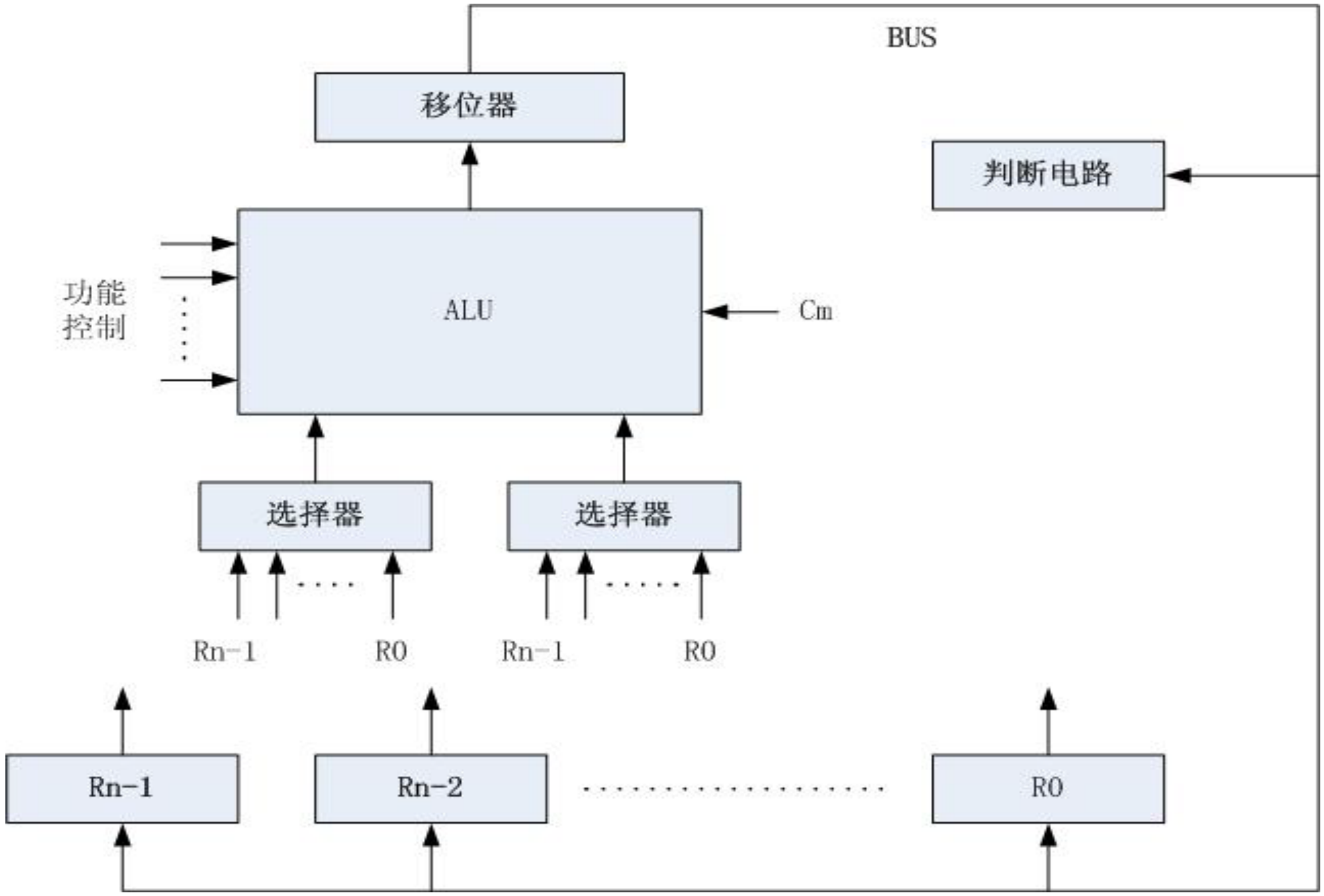
移位电路S
移位电路
计数器
−1
2ALU→S
ALU→S
ALU／2→S
ALU
C
C／2→C
2C→C
+1
上商
A→ALU
A
B̄→ALU
B→ALU
B
1
0
S→A
CP

3.10 运算部件组织

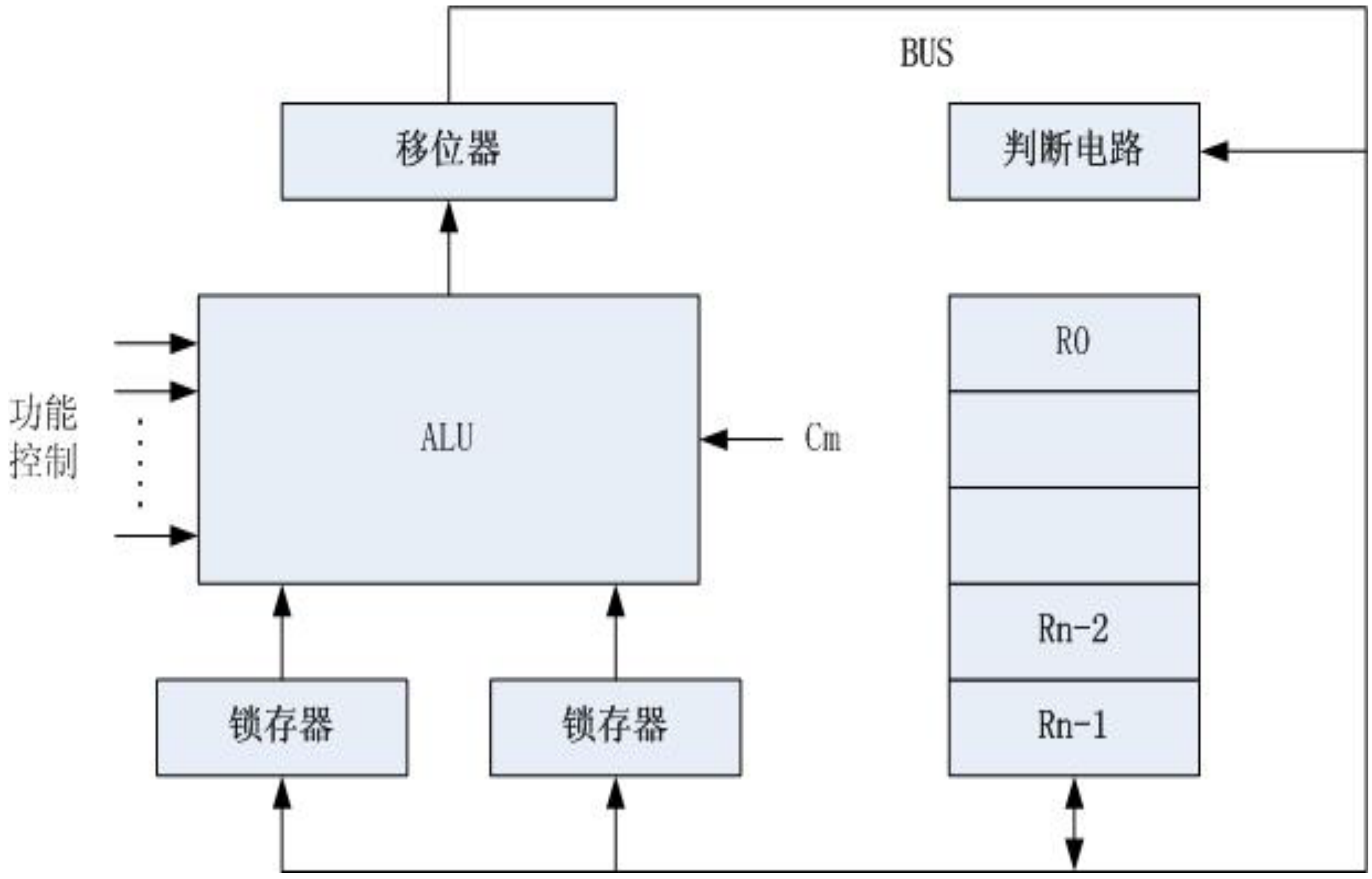
A，B，C寄存器的作用

运算	A	B	C
加法	被加数 运算结果	加数	无用
减法	被减数 运算结果	减数	无用
乘法	部分积 乘积高位	被乘数	乘数，乘积低位
除法	被除数 余数	除数	商

3.10 运算部件组织



3.10 运算部件组织

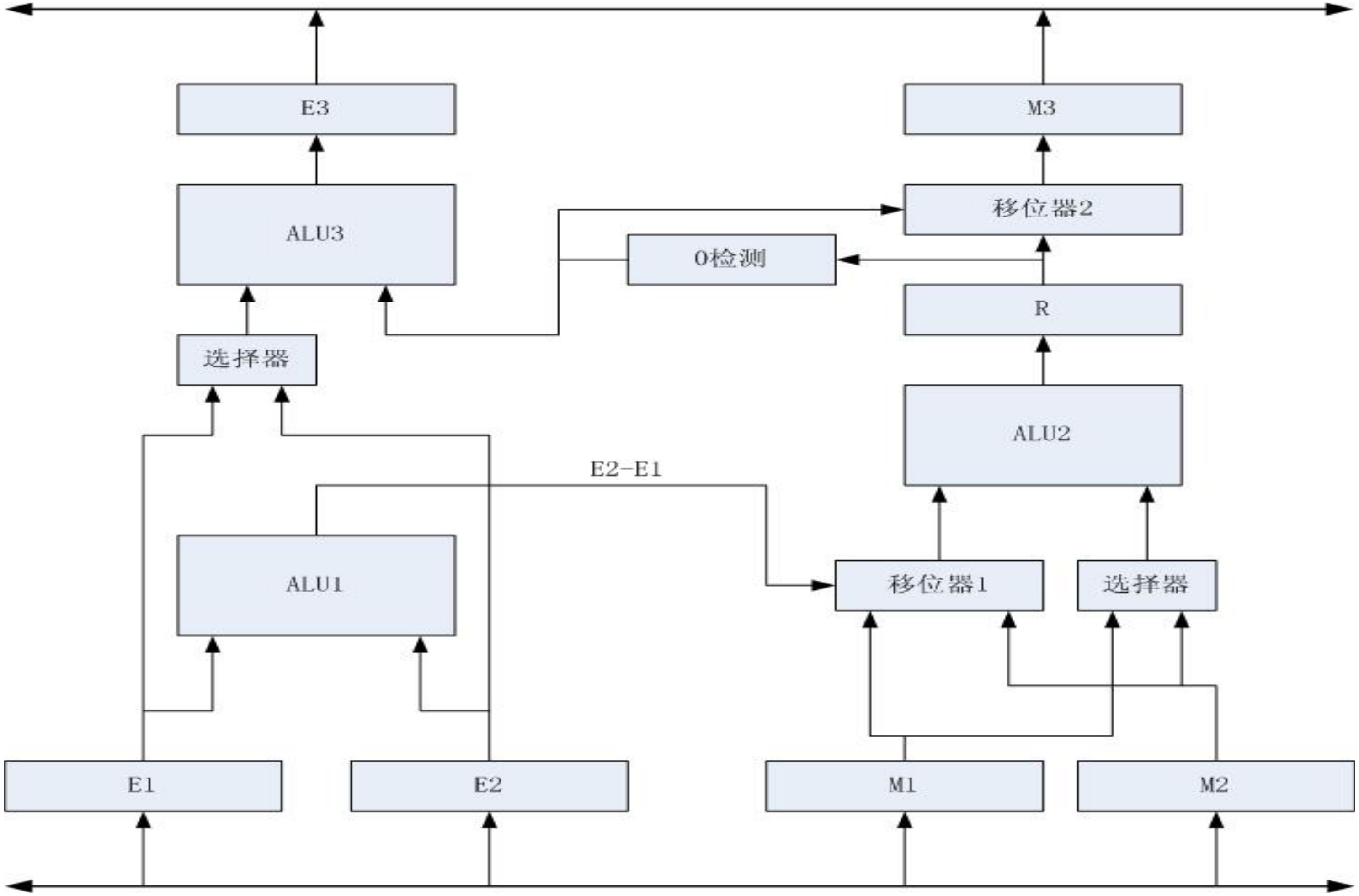


3.10 运算部件组织

2. 浮点运算部件

通常由阶码运算部件和尾数运算部件组成，其各自的结构与定点运算部件相似。但阶码部分仅执行加减法运算。其尾数部分则执行加减乘除运算，左规时有时需要左移多位。为加速移位过程，有的机器设置了可移动多位的电路（桶形移位器）。

3.8 浮点运算



习 题

P67

习题：16，17，18，21，22，23，24