计算机组成与体系结构

黄倩
huangqian@hhu.edu.cn
勤学楼4203

第2章 运算方法和运算器

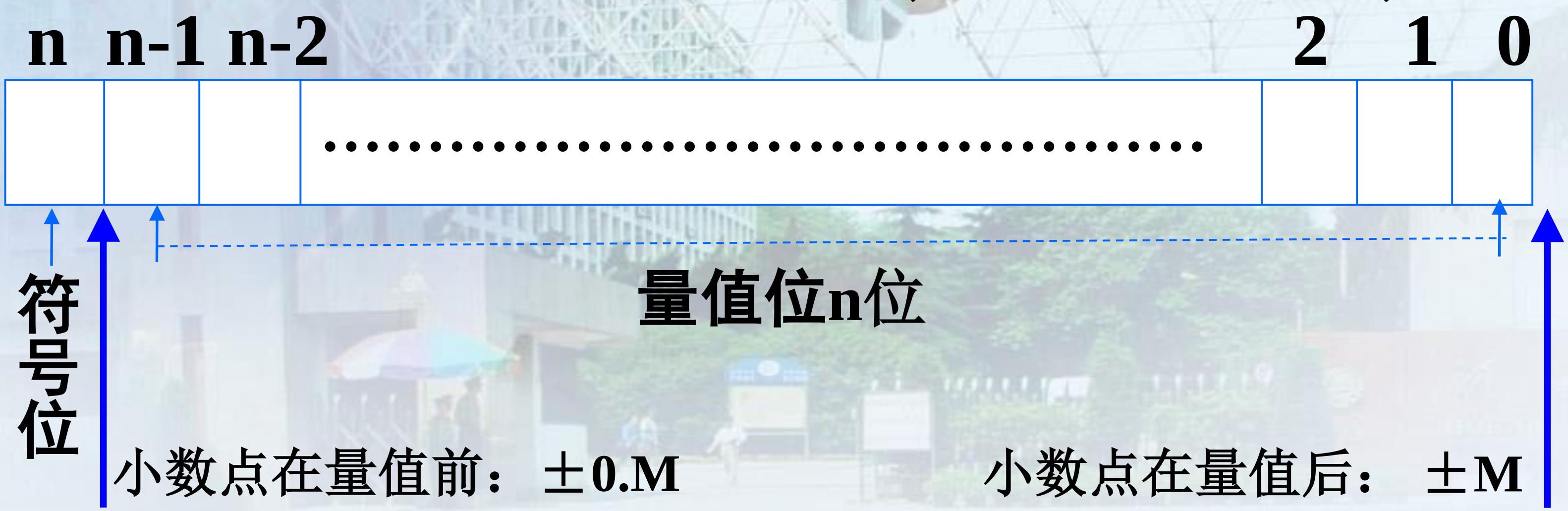
- 数据与文字的表示方法
- 定点加减法
- 定点乘法
- 定点除法
- 定点运算器的组成
- 浮点运算方法和浮点运算器

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 计算机中数的表示
    - 考虑因素
      - 要表示的数的类型（小数、整数、实数、复数）
      - 可能遇到的数值范围
      - 数值精确度
      - 数据存储和处理所需的硬件代价
    - 常用格式
      - 定点格式：小数点的位置固定不变
        - » 小数点不需要用“.”表示
      - 浮点格式：小数点的位置在一定范围内可以自由浮动
        - » 小数点不需要用“.”表示

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 定点格式：定点数由符号、量值两部分组成；小数点的位置固定不变，但并不真正储存小数点
    - 通常采用纯小数或纯整数格式；符号位0表示正，1表示负



- 无符号数：定点正整数，小数点在量值最后

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 浮点格式：浮点数由尾数、指数、基数三部分组成，能表示更大的数值范围
    - R进制数$N = M \times R^E$：M是尾数、E是指数、R为基数
    - 早期计算机：

小数点位置

阶符	阶码	数符	数码
E_s	$E_{m-1}E_{m-2}\cdots E_0$	M_s	$M_{n-1}M_{n-2}\cdots M_0$

【例】假设某机器中，浮点数的尾数用纯小数表示，则

0　00010011　1　1001011101表示$-0.1001011101 \times 2^{+00010011}$

假设某机器中，浮点数的尾数应表示为1.M的格式，则

0　00010011　1　1001011101表示$-1.1001011101 \times 2^{+00010011}$

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 浮点格式
    - IEEE754标准：

小数点位置

符号	指数	尾数
E_s	$E_{m-1}E_{m-2}\ldots E_0$	$M_{n-1}M_{n-2}\ldots M_0$

- 符号：1位，0正1负
- 指数：用移码表示，隐含符号；32位浮点数用8位表示指数，64位浮点数用11位表示指数
- 尾数： 32位浮点数用23位表示尾数，64位浮点数用52位表示尾数
- 真值：十进制值

32位：$(-1)^s\times(\mathbf{1.}M)\times 2^{E-127}$；64位：$(-1)^s\times(\mathbf{1.}M)\times 2^{E-1023}$

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 浮点格式
    - 浮点数的规格化表示：每台机器根据实际需要，对浮点数所采用的唯一表示方式
    - 【例】浮点数$(1.75)_{10}$可表示为1.11×2^0，0.111×2^1等多种形式，但如果机器遵循IEEE754标准，则该数只能表示为1.11×2^0的形式
  - 定点格式与浮点格式的选择
    - 根据计算机的使用条件确定
      - 高档微机：同时采用定点、浮点表示，由使用者进行选择
      - 单片机：多采用定点表示

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 【例1】若浮点数x的IEEE754标准存储格式为$(41360000)_{16}$，求其浮点数的十进制数值
  - 解：

$(41360000)_{16}$ = (0100 0001 0011 0110 0000 0000 0000 0000)$_2$

对照IEEE754标准的32位浮点数格式，有：

s=0, E=100 0001 0, M=011 0110 0000 0000 0000 0000

则十进制数值为$(-1)^s \times (1.M) \times 2^{E-127} = +1011.011$

$= (11.375)_{10}$

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 【例2】将数$(20.59375)_{10}$转换成IEEE754标准的32位浮点数的二进制存储格式
  - 解：

$$(20.59375)_{10} = (+10100.10011)_2$$
$$= (+1.010010011 \times 2^4)_2$$
$$= (+1.010010011 \times 2^{131-127})_2$$

对照IEEE754标准的32位浮点数格式，二进制存储格式为

0	10000011	010010011	00000000000000
符号	指数	尾数	**补14个0**

即$(01000001101001001100000000000000)_2=(41A4C000)_{16}$

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 十进制数串的表示形式——BCD码
    - BCD：Binary-Coded Decimal，即用4位二进制数来表示十进制数中的0~9这10个数码
    - 常用的BCD码
      - 8421码：最基本和最常用的BCD码，它和4位自然二进制码相似，从高到低各位的权值为8、4、2、1
      - 5421码/2421码：各位的权值分别为5、4、2、1和2、4、2、1
      - 余3码：在8421码的基础上加3
      - 余3格雷码：也称余3循环码，其特性是任何相邻的两组代码中，仅有一个数位不同，因而又叫单位距离码

2.1 数据与文字的表示方法

• 2.1.1 数据格式：常见BCD码的比较

十进制数	8421码	5421码	2421码	余3码	余3格雷码
0	0000	0000	0000	0011	0010
1	0001	0001	0001	0100	0110
2	0010	0010	0010	0101	0111
3	0011	0011	0011	0110	0101
4	0100	0100	0100	0111	0100
5	0101	1000	1011	1000	1100
6	0110	1001	1100	1001	1101
7	0111	1010	1101	1010	1111
8	1000	1011	1110	1011	1110
9	1001	1100	1111	1100	1010

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 十进制数串的表示形式
    - 字符串形式（非压缩的BCD码）：十进制数的符号位和每个数位都用一个字节表示，因此每个十进制数需要连续的多个字节才能表示
      - 问题1：每个字节中的高4位都浪费了
      - 问题2：运算时不能直接进位

【例】十进制数98用二进制可表示为01100010，如果用字符串形式则为 00001001　00001000；十进制数12用二进制可表示为00001100，用字符串形式则为00000001　00000010；两者相加会得到一个三字节的字符串，进位处理不方便

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 十进制数串的表示形式
    - 压缩的BCD码：一个字节存放两个十进制数位
      - 空间浪费少
      - 运算时可以直接进位

【例】十进制数98用字符串形式表示为 00001001 00001000，用压缩的BCD码则表示为10011000

  - 符号的表示方法
    - » 符号位占半个字节，用2^4-10=6种冗余状态中的两种来表示正和负。例如，可以用12（C）表示正号、13（D）表示负号

2.1 数据与文字的表示方法

- 2.1.1 数据格式
  - 十进制数串的表示形式
    - 压缩BCD码的运算：可能需要对结果进行修正
    - 【例】用压缩的BCD码计算98+74=172
      - BCD码运算
        - » 低四位的运算结果是1010-1111，则+06H
        - » 低四位向高四位有进位（半进位），则+06H
        - » 高四位的运算结果是1010-1111，则+60H
        - » 高四位向前有进位，则+60H
      - 二进制运算

```
  0110 0010
+ 0100 1010
----------------
  1010 1100
```

```
  1001 1000
+ 0111 0100
----------------
1 0000 1100
+        0110
----------------
1 0001 0010
+ 0110
----------------
1 0111 0010
```

2.1 数据与文字的表示方法

- 2.1.2 数的机器码表示
  - 原码
  - 反码
  - 补码
  - 移码

2.1.2.1 原码表示法

- 定点小数
  - 小数点固定在符号位后边
    - 如果$0 \le X < 1$， $[X]_{原} = 0. X1X2\ldots Xn = X$
    - 如果$-1 < X \le 0$， $[X]_{原} = 1. X1X2\ldots Xn$

$$= 1 + 0. X1X2\ldots Xn$$
$$= 1 + |X|$$
$$= 1 - X$$

2.1.2.1 原码表示法

- 定点整数
  - 小数点固定在最末位后边
    - 如果$0 \le X < 2^n$， $[X]_{原} = 0\ X1X2 \ldots Xn = X$
    - 如果$-2^n < X \le 0$，$[X]_{原} = 1\ X1X2 \ldots Xn$

$$= 2^n + X1X2 \ldots Xn$$

$$= 2^n + |X|$$

$$= 2^n - X$$

2.1.2.1 原码表示法

- 原码的优缺点
  - 表示简单，直接在前面加符号位即可
  - 运算复杂
    - 当两数相加时，如果是同号则数值相加；如果是异号，则要进行减法。而在进行减法时还要比较绝对值的大小，然后大数减去小数，最后还要给结果选择符号
  - 有“+0”和“-0”之分
    - 表示范围少了一个值
  - 解决方案：引入补码

2.1.2.2 反码表示法

- 定点小数
  - 小数点固定在符号位后边
    - 如果$0 \le X < 1$， $[X]_{反} = 0.X1X2\ldots Xn = X = [X]_{原}$
    - 如果$-1 < X \le 0$， $[X]_{反} = (2-2^{-n}) + X$

$$= (2-2^{-n}) - |X|$$

$$= 1.11\ldots1 - 0.X1X2\ldots Xn$$

$$= \overline{[|X|]}_{原}$$

2.1.2.2 反码表示法

- 定点整数
  - 小数点固定在最末位后边
    - 如果$0 \le X < 2^n$， $[X]_{反} = 0\ X1X2\ldots Xn = X = [X]_{原}$
    - 如果$-2^n < X \le 0$，$[X]_{反} = (2^{n+1} - 1) + X$

$$= 1\ 11\ldots111 - 0\ X1X2\ldots Xn$$

$$= \overline{[|X|]}_{原}$$

2.1.2.2 反码表示法

- 反码的优缺点
  - 将减法运算转换为加法运算，从而简化了运算规则
  - 有“+0”和“-0”之分
    - 表示范围少了一个值
  - 解决方案：引入补码

2.1.2.3 补码表示法

- 补码的引入
  - 以钟表对时为例说明补码的概念
    - 假设现在的标准时间为4点正；而有一只表已经7点了，为了校准时间，可以采用两种方法：一是将时针退 7-4=3格；一是将时针向前拨12-3=9格
  - “取模”的启示
    - 计算机中由于机器字长有限，因此运算是有模的
    - 负数用补码表示时，可以把减法转化为加法，在计算机中实现方便

2.1.2.3 补码表示法

- 补码的模
  - 全1再加1，则溢出，该数即为模数
    - 定点小数的模一定为2

小数点位置

$X_{n-1}\quad X_{n-2}\ \dots\ X_2\quad X_1\quad X_0$

符号位

$$
\begin{array}{r}
1.\ 111\ \dots\ 111 \\
+\qquad\qquad\qquad 1 \\
\hline
10.\ 000\ \dots\ 000
\end{array}
$$

2.1.2.3 补码表示法

- 补码的模
  - 全1再加1，则溢出，该数即为模数
    - 定点整数的模为2^n，这里n是机器字长

小数点位置

$X_{n-1}\ \ X_{n-2} \ldots X_2\ \ X_1\ \ X_0$

符号位

```
   1  111 ... 111
+               1
------------------
  10  000 ... 000
```

河海大学

2.1.2.3 补码表示法

- 定点小数的补码表示
  - 如果$0 \le X < 1$， $[X]_{补} = 0.\ X1X2\ldots Xn = X = [X]_{原}$
  - 如果$-1 < X \le 0$， $[X]_{补} = 2 + X$

$$= 10.\ 00\ldots0 - 0.\ X1X2\ldots Xn$$

$$= \overline{[|X|]}_{原} + 0.00\ldots01$$

- 定点整数的补码表示
  - 如果$0 \le X < 2^{n}$， $[X]_{补} = 0X1X2\ldots Xn = X = [X]_{原}$
  - 如果$-2^{n} \le X \le 0$， $[X]_{补} = 2^{n+1} + X = \overline{[|X|]_{原}} + 1$

2.1.2.4 移码表示法

- 移码的求法
  - 移码的数值部分与补码相同，符号位相反
- 移码的应用
  - 移码表示法通常用于浮点数的阶码表示，因此主要讨论定点整数
    - 设机器字长为n+1位，则：
      - $[X]_{移} = 2^n + X = 1\ 00...0 + X$（相当于在符号位加1）

2.1 数据与文字的表示方法

- 2.1.3 字符与字符串的表示方法
  - 现代计算机不仅处理数值问题，也处理大量非数值问题，因此必然要引入文字、字母以及某些专用符号；然而数字计算机只能处理二进制数据，因此需要用二进制格式表示字符与字符串信息
  - 字符（Char）
    - 目前国际上普遍采用的字符系统是七单位的IRA（International Reference Alphabet）码，其美国版本为ASCII（American Standard Code for Information Interchange）码，见下页表格

2.1 数据与文字的表示方法

- 2.1.3 字符与字符串的表示方法
  - ASCII字符编码表

高三位　　通常用1个字节表示，最高位用做奇偶校验

低四位

	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P		p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	DEL	ETB		7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N		n	~
1111	SI	US	/	?	O	_	o	DEL

数字0-9： 30H-39H
大写字母A-Z： 41H-5AH
小写字母a-z： 61H-7AH
空格： 20H
回车、换行： 0DH, 0AH

2.1 数据与文字的表示方法

- 2.1.3 字符与字符串的表示方法
  - 字符串（String）
    - 字符串是指连续的一串字符，通常方式下，它们占用主存中连续的多个存储单元，每个字节存一个字符
    - 例：将字符串IF ␣ A>B ␣ THEN ␣ READ(C)从高位字节到低位字节依次存在主存中
    - 解：设主存存储单元长度由4个字节组成

字符	ASCII码	存储单元（32位）
IF␣A	73、70、32、65	01001001 01000110 00100000 01000001
>B␣T	62、66、32、84	00111110 01000010 00100000 01010100
HEN␣	72、69、78、32	01000111 01000101 01001110 00100000
READ	82、69、65、68	01010010 01000101 01000001 01000100
(C)␣	40、67、41、32	00101000 01000011 00101001 00100000

2.1 数据与文字的表示方法

- 2.1.4 汉字的表示方法
  - 汉字的计算机处理过程
    - 汉字国标码@1981.5：与ASCII码共存时会产生二义性
    - 汉字内码：在汉字国标码每个字节最前面加1

两个字节表示一个汉字，最高位均为0

为与ASCII区分，两字节的最高位变为1

2.1 数据与文字的表示方法

- 2.1.5 校验码
  - 背景：元件故障、噪声干扰等各种因素常常导致计算机在处理信息过程中出现错误
  - 措施：通常的方法是，在每个数据字上添加一些校验位，用来检错或纠错
    - 检错码
      - 奇偶校验

数　据	偶校验编码 C	奇校验编码 C
1 0 1 0 1 0 1 0	1 0 1 0 1 0 1 0 0	1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 0	0 1 0 1 0 1 0 0 1	0 1 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 0	1 1 1 1 1 1 1 1 1

2.1 数据与文字的表示方法

- 2.1.5 校验码
  - 纠错码
    - Hamming Code（海明码/汉明码）
    - 课后作业：举一个4位信息码的例子，描述用1位海明码检错和纠错的过程。

河海大学

2017年图灵奖

- John L. Hennessy & David A. Patterson
  - 主要贡献：RISC。现在每年生产的160亿个微处理器中，99%是RISC处理器，所有的智能手机、平板电脑、物联网设备中都有他们的技术
  - 人物简介
    - Hennessy：曾任斯坦福大学校长，2018任Alphabet董事长
    - David A. Patterson：曾任ACM会长，2016年加盟谷歌TPU
  - 教科书
    - 圣经：1990，《计算机体系结构：量化研究方法》
    - 《计算机组成与设计——硬件/软件接口》

2017年图灵奖

- David A. Patterson口述成长历程
  - 家里第一个大学生，数学专业，上了编译课后决定改行
  - 毕业后，在实验室当网络员
  - 在实验室读研，因为其他人都读博，所以也读了一个博士
  - 1977，加入UC Berkeley，研究并行结构
  - 1979，申请学术休假加入DEC，思考学术界的优势和劣势
  - 回到UC Berkeley，启动RISC项目
    - VLSI论文被IEEE Computer拒稿：这是一种愚蠢的设计计算机的方法
  - 写书动机
    - 要当系主任了，感觉学术生涯要结束了，打算做最后一个贡献

2.2 定点加减法

- 补码加法
- 补码减法
- 溢出概念与检测方法
- 基本的二进制加法/减法器

2.2 定点加减法

- 2.2.1 补码加法
  - 运算规则

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$（证明略）

  - 补码加法的特点
    - 1. 符号位和数值位一起参加运算，即**符号位数值化**，不需要判断或特别处理符号位；
    - 2. 在有模运算时，进位丢掉即可

2.2 定点加减法

- 2.2.1 补码加法
  - [例11] x ＝+1001， y ＝+0101，求 x ＋ y

[解] $[x]_{补}=0\ 1001$， $[y]_{补}=0\ 0101$

$$
\begin{array}{lr}
[x]_{补} & 0\ 1\ 0\ 0\ 1 \\
\underline{+[y]_{补}} & \underline{0\ 0\ 1\ 0\ 1} \\
[x+y]_{补} & 0\ 1\ 1\ 1\ 0
\end{array}
$$

所以 x ＋ y ＝＋1110 **（验证9+5=14）**

2.2 定点加减法

- 2.2.1 补码加法
  - [例12] x ＝+1011， y ＝-0101，求 x ＋ y

[解] $[x]_{补}$＝0 1011， $[y]_{补}$＝1 1011

$$
\begin{array}{lr}
[x]_{补} & 0\,1\,0\,1\,1 \\
+[y]_{补} & \underline{1\,1\,0\,1\,1} \\
[x+y]_{补} & 1\;0\,0\,1\,1\,0
\end{array}
$$

所以 x ＋ y ＝＋0110 **（验证11-5=6）**

2．2．2 补码减法

特点：

1 减法转换为加法

2 符号位数值化

3 有模运算，进位丢掉

运算规则：

$$[X-Y]_{补} = [X+(-Y)]_{补}$$

$$= [X]_{补} + [-Y]_{补}$$

$$= [X]_{补} + [[Y]_{补}]_{求补}$$

$$= [X]_{补} + \overline{[Y]}_{补} + 1$$

如果机器字长为n+1位，则
对于定点小数，末尾加1就相当于加2^{-n}
对于定点整数，末尾加1就相当于加1

2.2.2 补码减法

[例13] 已知 $x1=-1110$， $x2=+1101$，求：

$[x1]_{补}$，$[-x1]_{补}$，$[x2]_{补}$，$[-x2]_{补}$。

[解]

$$[x1]_{补}=1\ 0010$$

$$[-x1]_{补}=0\ 1101+1=0\ 1110$$

$$[x2]_{补}=0\ 1101$$

$$[-x2]_{补}=1\ 0010+1=1\ 0011$$

2．2．2 补码减法

[例14] $x=+1101$，$y=+0110$，求 $x-y$。

[解]

$[x]_{补}=0\ 1101$

$[y]_{补}=0\ 0110$，　　$[-y]_{补}=1\ 1010$

$$
\begin{array}{lr}
 [x]_{补} & 0\ 1\ 1\ 0\ 1 \\
 +[-y]_{补} & 1\ 1\ 0\ 1\ 0 \\
\hline
 [x-y]_{补} & 1\quad 0\ 0\ 1\ 1\ 1
\end{array}
$$

所以 $x-y=+0111$　　（验证13-6=7）

2．2．3 溢出概念与检测方法

1、溢出概念

由于机器字长是有限的，有模运算，必然会出现结果比所能表示的最小值还小，或者比最大值还大，也就是说超过有限字长的表示范围。

2．2．3 溢出概念与检测方法

[例15] $x = +1011$， $y = +1001$，求 $x + y$。

[解] $[x]_{补} = 0\ 1011$ $[y]_{补} = 0\ 1001$

$$
\begin{array}{rr}
[x]_{补} & 0\ 1\ 0\ 1\ 1 \\
+\ [y]_{补} & 0\ 1\ 0\ 0\ 1 \\
\hline
[x+y]_{补} & 1\ 0\ 1\ 0\ 0
\end{array}
$$

11+9=20，两个正数相加，结果是负数，显然错误。实际上是超过了所能表示的最大值（机器字长为5位，表示范围-16至+15）。

2.2.3 溢出概念与检测方法

[例16] x＝－1101， y＝－1011，求 x＋y。

[解] $[x]_{补}$＝1 0011 $[y]_{补}$＝1 0101

$$
\begin{array}{lr}
 & [x]_{补} \quad 1\ 0011 \\
+ & [y]_{补} \quad 1\ 0101 \\
\hline
 & [x+y]_{补} \quad 1\ 0\ 1000
\end{array}
$$

-13-11=-24，两个负数相加，结果是正数，显然错误。实际上是低于所能表示的最小值（-16至+15）。

2．2．3 溢出概念与检测方法

结论：

之所以发生错误，是因为运算结果产生了溢出。按照补码加法运算规则，出现错误有二种情况：

（1）两个正数相加，结果大于机器所能表示的最大正数，称为上溢、正溢。

（2）两个负数相加，结果小于机器所能表示的最小负数，称为下溢、负溢。

或者统一为：两个同符号数相加、而结果的符号相反，肯定错误，即溢出。

2．2．3 溢出概念与检测方法

2、 检测方法：单符号位法

1） 单符号位法

2）双符号位法（变形补码、模4补码）

2. 2. 3 溢出概念与检测方法

2、 检测方法：单符号位法

（1）根据被加数、加数、结果的符号位判断

A_f	B_f	C_f	V
0	0	1	1
1	1	0	1

$$V=\overline{A_f}\ \overline{B_f}\ C_f + A_f\ B_f\ \overline{C_f}$$

（与或表达式，采用与或非门即可实现）

2．2．3 溢出概念与检测方法

2、 检测方法：单符号位法

（2）根据符号位、最高数值位向前的进位判断

C_f	C_0	V
0	1	1
1	0	1

$V = C_f \oplus C_o$　异或门可以实现

2.2.3 溢出概念与检测方法

3、检测方法：双符号位法

双符号位表示：

$S_{f1}\ S_{f2} = 00$ 表示正数

$S_{f1}\ S_{f2} = 11$ 表示负数

运算方法：

双符号位都数值化，最高符号位上产生的进位丢掉。

2. 2. 3 溢出概念与检测方法

3、 检测方法：双符号位法

根据结果的双符号位进行判断：

S_{f1} S_{f2} = 00 结果为正，结果正确

S_{f1} S_{f2} = 11 结果为负，结果正确

S_{f1} S_{f2} = 01 正数相加，结果为负

S_{f1} S_{f2} = 10 负数相加，结果为正

$V = S_{f1} \oplus S_{f2}$ 异或门可以实现

2．2．3 溢出概念与检测方法

[例17] x ＝＋1100， y ＝＋1000，求 x ＋ y 。

[解] $[x]_{补}$＝00 1100， $[y]_{补}$＝00 1000

$$
\begin{array}{lr}
 & [x]_{补} \quad 00\ 1100 \\
+ & [y]_{补} \quad 00\ 1000 \\
\hline
 & 01\ 0100
\end{array}
$$

两个符号位出现“01”，表示已溢出，即结果大于＋15。

2．2．3 溢出概念与检测方法

[例15] x ＝-1100， y ＝-1000，求 x ＋ y 。

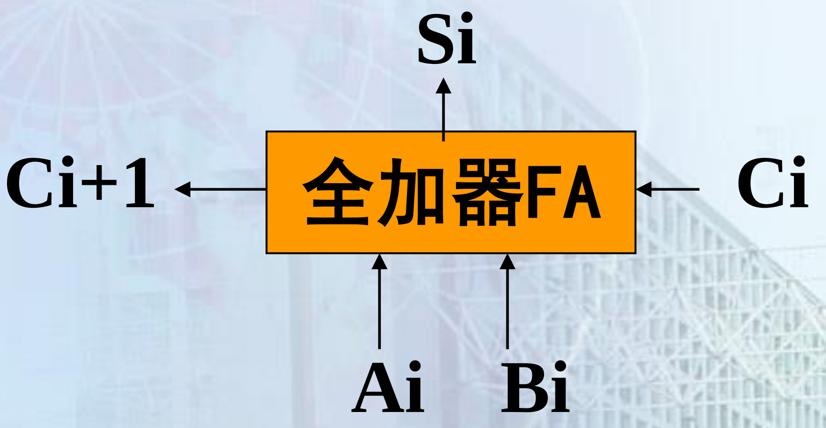
[解] $[x]_{补}$＝11 0100， $[y]_{补}$＝11 1000

$$
\begin{array}{lr}
 & [x]_{补} \quad 11\ 0100 \\
+ & [y]_{补} \quad 11\ 1000 \\
\hline
 & 1\ 10\ 1100
\end{array}
$$

两个符号位出现“10”，表示已溢出，即结果小于－16。

2．2．4 基本的二进制加法/减法器

1、基本加法单元（一位全加器）



A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S_i = (A_i \oplus B_i) \oplus C_i$　奇数个1

$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$　两个以上1

$= A_iB_i + (A_i \oplus B_i) C_i$

2.2.4 基本的二进制加法/减法器

1、基本加法单元（一位全加器）

$$S_i = (A_i \oplus B_i) \oplus C_i$$

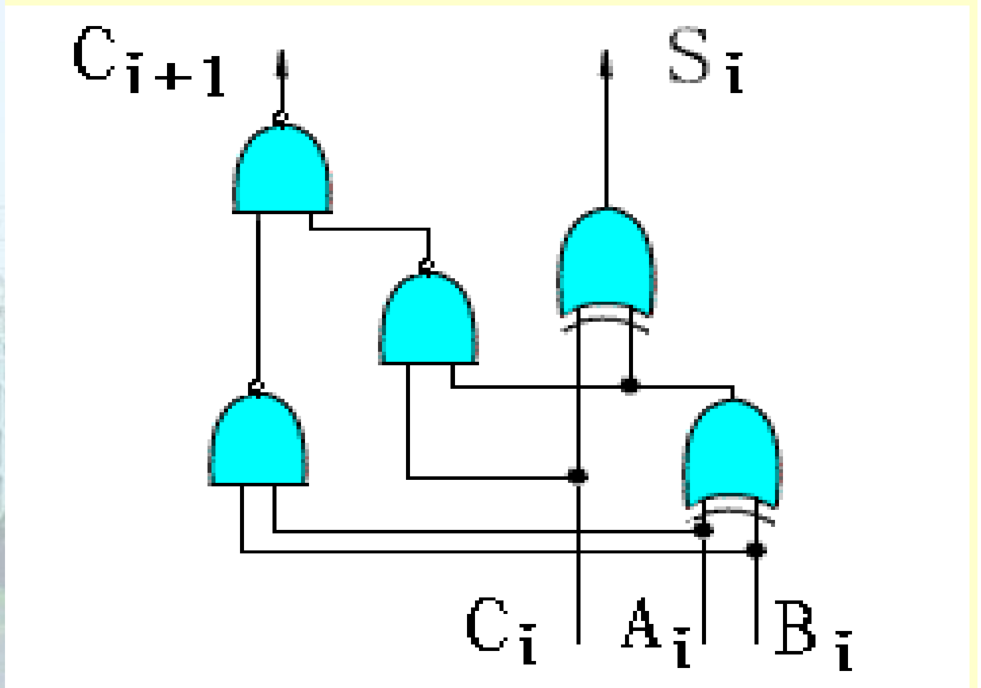
$$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$$

$$= A_iB_i + (A_i \oplus B_i)C_i$$

$$= \overline{\overline{A_iB_i + (A_i \oplus B_i)C_i}}$$

$$= \overline{\overline{A_iB_i} \cdot \overline{(A_i \oplus B_i)C_i}}$$

异或门、与非门实现



2．2．4 基本的二进制加法/减法器

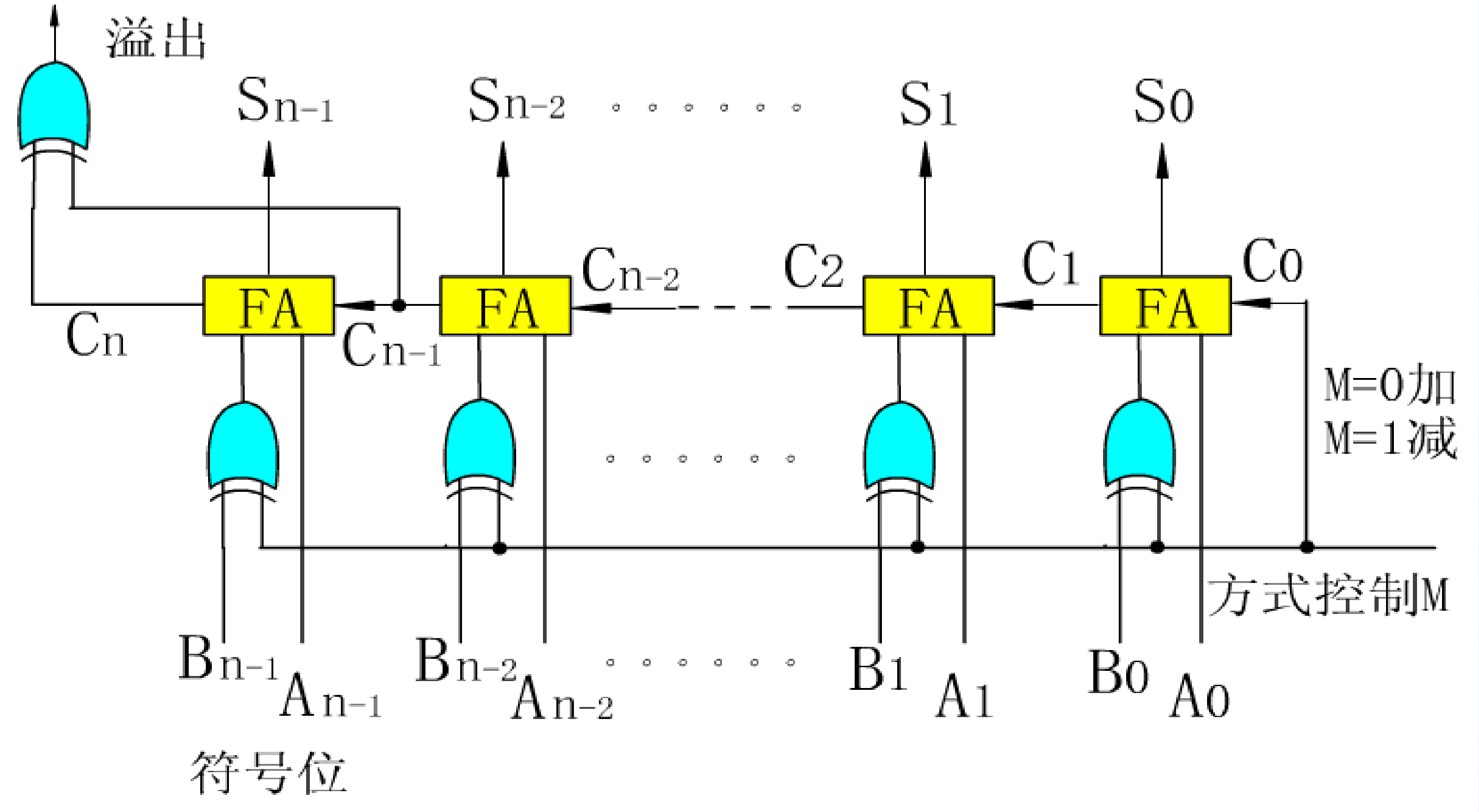
2、串行进位的补码加法/减法器

（1）n个1位的全加器(FA)可级联成一个n位的串行进位加减器。

（2）所谓串行进位是指高位全加器必须依赖低位的进位到来之后，才能进行运算。显然效率较低，后面会介绍并行进位的加法器。

2．2．4 基本的二进制加法/减法器

溢出 $V = C_{n-1} \oplus C_n$



运算级

取反级

M=0：A+B

M=1：A+B+1 = A-B

2.3 定点乘法运算

- 乘法器实现
  - 早期
    - 软件迭代：移位、累加
  - 现代
    - 硬件实现

2.3.1 原码并行乘法

- 1、人工算法与机器算法的同异性
  - 被乘数 $[x]_{原}=x_f\, x_{n-1}\cdots x_1x_0$
  - 乘数　 $[y]_{原}=y_f\, y_{n-1}\cdots y_1y_0$
  - 乘积　 $[z]_{原}=(x_f \oplus y_f) + (x_{n-1}\cdots x_1x_0) * (y_{n-1}\cdots y_1y_0)$
    - 符号部分：由两数符号位异或而得
    - 数值部分：与十进制乘法类似

```
            0.  1  1  0  1   (x)
×           0.  1  0  1  1   (y)
---------------------------------
                1  1  0  1
             1  1  0  1
          0  0  0  0
+      1  1  0  1
---------------------------------
0.  1  0  0  0  1  1  1  1   (z)
```

2.3.1 原码并行乘法

- 1、人工算法与机器算法的同异性
  - 软件迭代步骤
    - S1: 保存被乘数、乘数，乘法结果置为0
    - S2: 在每个时钟周期，判断乘法的最低位。如果为1，则将被乘数加到乘法结果；如果为0，则不进行加法操作。然后乘数右移1位，被乘数左移1位，进入下一时钟周期。
    - S3: 迭代执行，得到正确结果。

```
          0. 1 1 0 1   (x)
×         0. 1 0 1 1   (y)
-------------------------
             1 1 0 1
           1 1 0 1
         0 0 0 0
+      1 1 0 1
-------------------------
0. 1 0 0 0 1 1 1 1     (z)
```

2.3.1 原码并行乘法

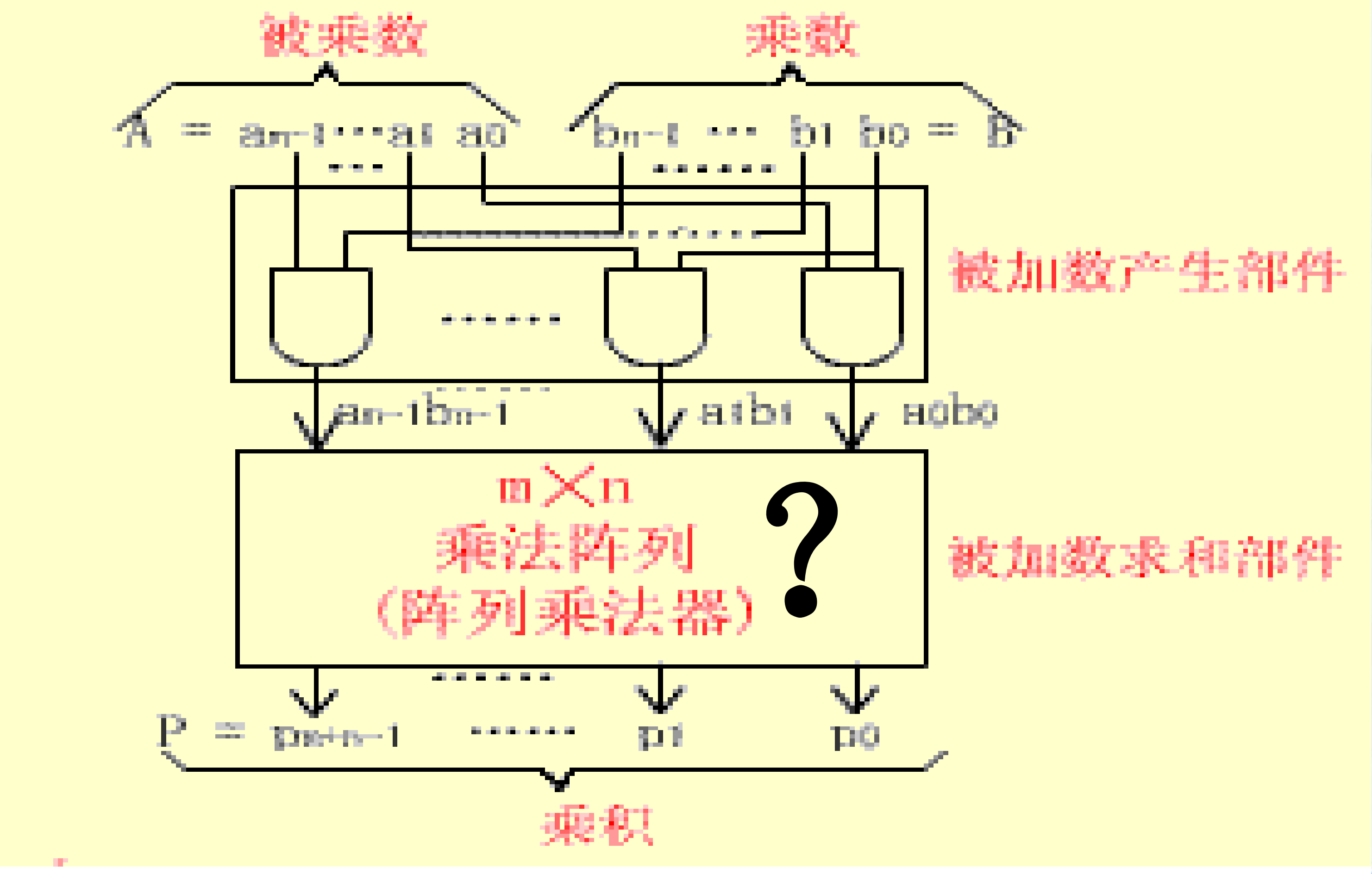
2、不带符号的阵列乘法器

$$
\begin{array}{rrrrrrrrr}
 & & & & a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 = A \\
\times) & & & & & b_{n-1} & \cdots & b_1 & b_0 = B \\
\hline
 & & & & a_{m-1}b_0 & a_{m-2}b_0 & \cdots & a_1b_0 & a_0b_0 \\
 & & & a_{m-1}b_1 & a_{m-2}b_1 & \cdots & a_1b_1 & a_0b_1 & \\
 & & \cdot & & & & \cdot & & \\
 & \cdot & & & & \cdot & & & \\
+) & a_{m-1}b_{n-1} & a_{m-2}b_{n-1} & \cdots & a_1b_{n-1} & a_0b_{n-1} & & & \\
\hline
p_{m+n-1} & p_{m+n-2} & p_{m+n-3} & \cdots & p_{n-1} & \cdots & p_1 & p_0 & = P
\end{array}
$$

所有的a_ib_j，实际上可以用$m \times n$个与门一次性并行产生，关键是a_ib_j如何相加的问题?

2．3．1 原码并行乘法

2、不带符号的阵列乘法器



2．3．1 原码并行乘法

2、不带符号的阵列乘法器

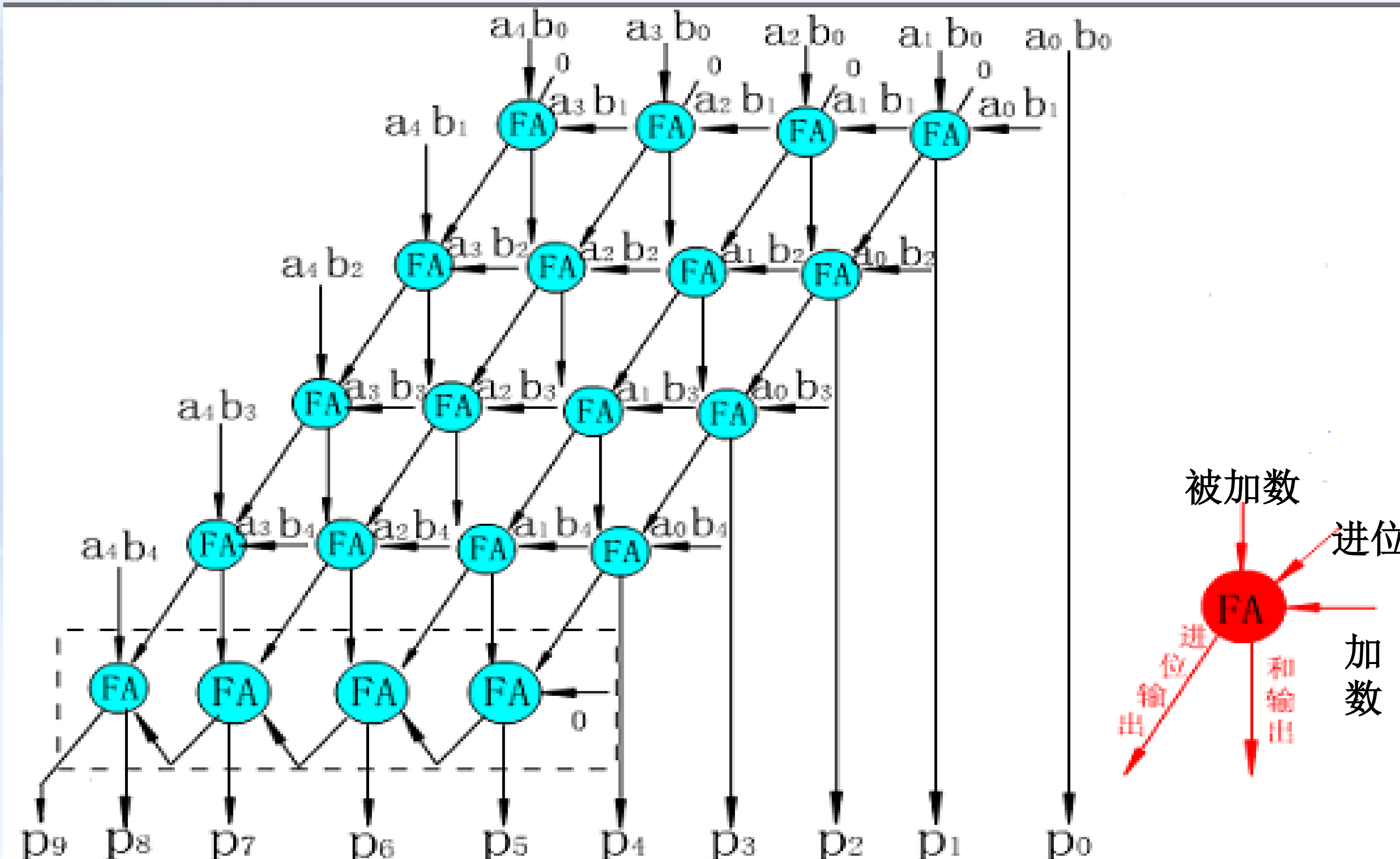
5bits * 5bits的乘法过程描述（A4A3A2A1A0 * B4B3B2B1B0)

					A4B0	A3B0	A2B0	A1B0	A0B0	1
				A4B1	A3B1	A2B1	A1B1	A0B1		2
			A4B2	A3B2	A2B2	A1B2	A0B2			3
		A4B3	A3B3	A2B3	A1B3	A0B3				4
+	A4B4	A3B4	A2B4	A1B4	A0B4					5
进位 P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	

可以这样操作：第1行+第2行、然后加第3行、加第4行、加第5行。

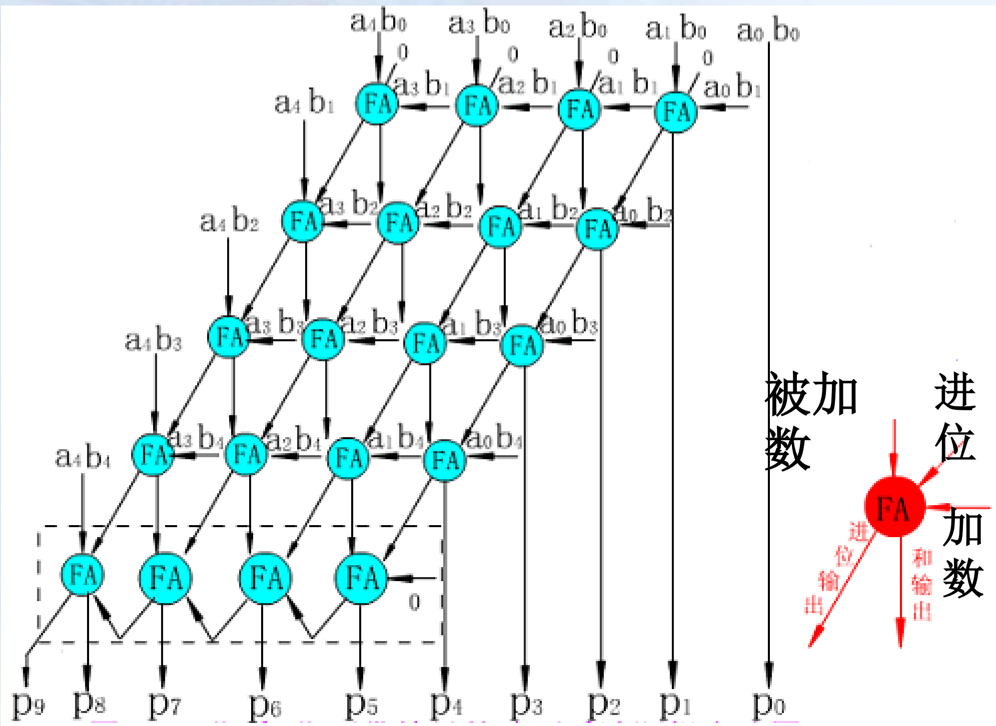
2．3．1 原码并行乘法

2、不带符号的阵列乘法器



2．3．1 原码并行乘法

2、不带符号的阵列乘法器



第0步：5X5个与门同时工作求a_ib_j；

第1步：第1排4个全加器同时工作

初始进位都为0

全加器的进位到第2排；

第2步：第2排4个全加器同时工作

全加器的进位到第3排；

……

第5步：第5排全加器串行工作

其中第1个的加数为0

2．3．1 原码并行乘法

3、 带符号的阵列乘法器

（1）求补电路

求补：从低位开始往高位找，找到第一个1，保留该1和低位的0，高位的所有位取反（0变1、1变0）。

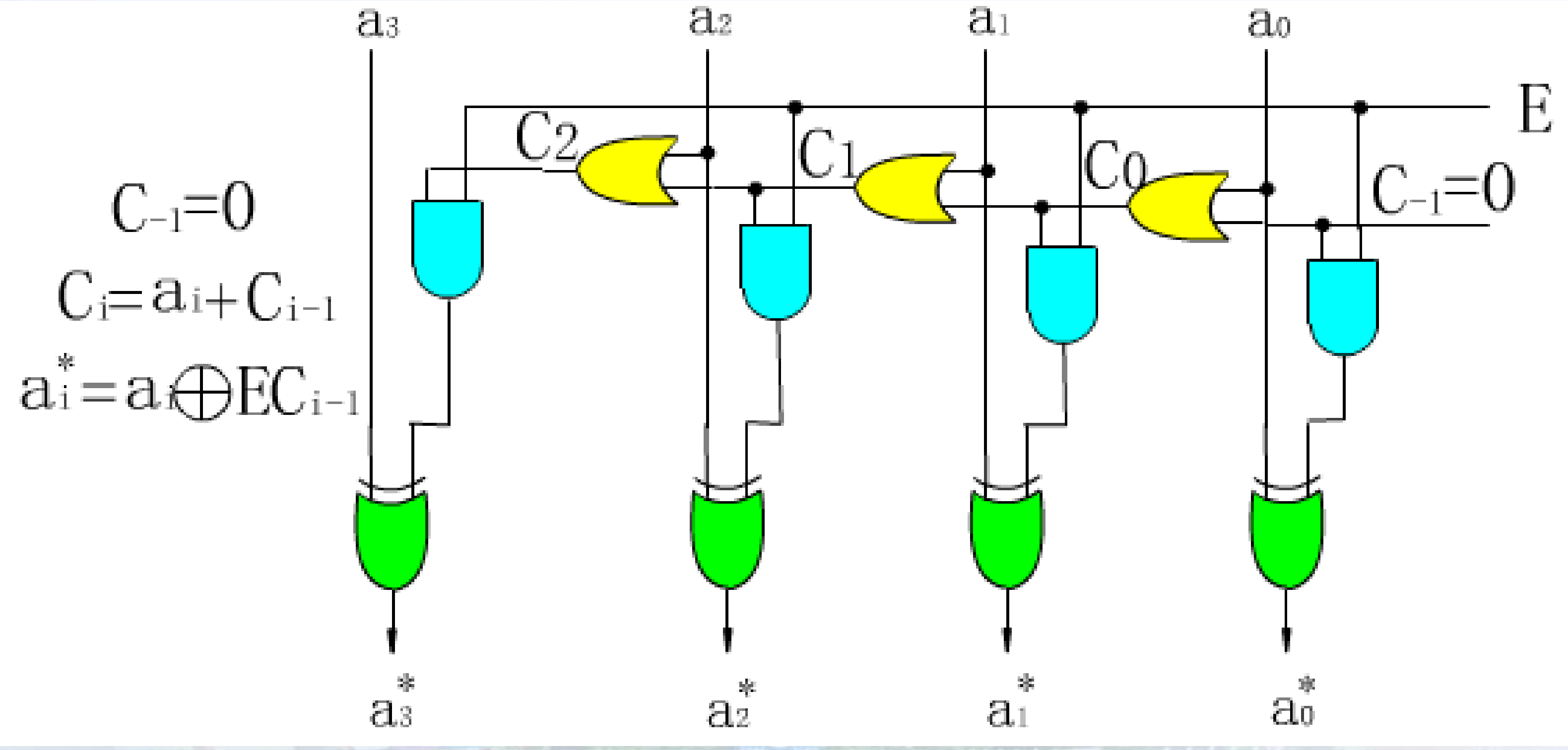
例：1 1 0 0 1 0 1 1 1 0 0 0

求补：0 0 1 1 0 1 0 0 1 0 0 0

2．3．1 原码并行乘法

[例] $a_3a_2a_1a_0$=1010

从低位往高位找到第一个1，该1通过或门往高位传递，按位取反。



E：数的符号位；

E=0，所有位原样输出，**$Ea_3a_2a_1a_0$= 0 1010**，则**a_3*a_2*a_1*a_0*= 1010**

E=1，求补， **$Ea_3a_2a_1a_0$= 1 1010**，则**a_3*a_2*a_1*a_0*= 0110**

2．3．1 原码并行乘法

3、 带符号的阵列乘法器

（1）求补电路

引入求补电路的意义：求负数补码的尾数部分的原码，以便将带符号补码数转换为原码再相乘。

2．3．1 原码并行乘法

4、 带符号数的阵列乘法器

（2）**(n+1) * (n+1)**带符号原码乘法器

1）符号位进行异或运算，就是乘积的符号位；

2）尾数部分用**n*n**不带符号数的阵列乘法器实现；

2．3．1 原码并行乘法

4、 带符号数的阵列乘法器

（3）(n+1) * (n+1)带符号的补码乘法器

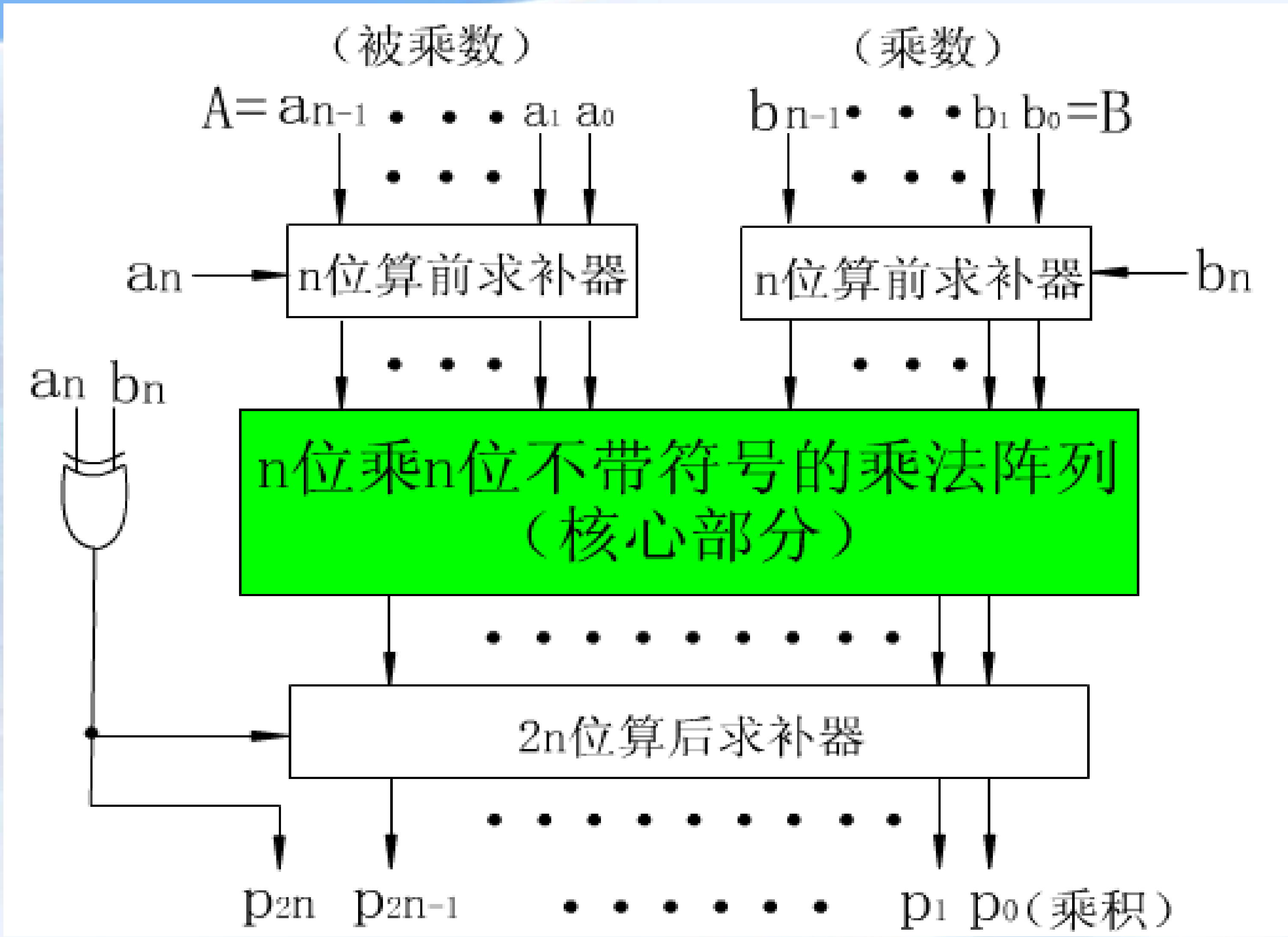
1）符号位进行异或运算，就是乘积的符号位；

2）用算前求补器将补码尾数转换为原码尾数，E控制线正好用数的符号位；

3）采用n*n无符号数阵列乘法器求乘积；

4）用算后求补器将乘积的原码尾数转换为补码尾数，E控制线正好用乘积的符号位；

2. 3. 1 原码并行乘法


（被乘数）
（乘数）
A= a_{n-1} • • • a_1 a_0
b_{n-1} • • • b_1 b_0=B
a_n
n位算前求补器
n位算前求补器
b_n
a_n b_n
n位乘n位不带符号的乘法阵列
（核心部分）
2n位算后求补器
p_{2n} p_{2n-1} • • • • • • p_1 p_0（乘积）

2. 5 定点运算器的组成

逻辑运算

多功能算术/逻辑运算单元

内部总线

定点运算器的基本结构

2．5．1 逻辑运算

[例24] x_1=01001011, x_2=11110000, 求$\overline{x}_1$， $\overline{x}_2$

[解]　非运算

$$\overline{x}_1 = 10110100$$

$$\overline{x}_2 = 00001111$$

2.5.1 逻辑运算

[例25]　x =10100001，y =10011011，求 x ∨ y 。

[解]　或运算

$$
\begin{array}{cccccccccc}
 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & x \\
\vee & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & y \\
\hline
 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & z
\end{array}
$$

即 x ∨ y ＝ 10111011

2.5.1 逻辑运算

[例26] x =10111001, y =11110011, 求 x ∧ y 。

[解] 与运算

$$
\begin{array}{rcccccccccl}
 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & & x \\
\wedge & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & & y \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & & z
\end{array}
$$

即 x ∧ y ＝ 10110001

2.5.1 逻辑运算

[例27]　x ＝10101011, y ＝11001100, 求 x ⊕ y 。

[解]　异或运算

$$
\begin{array}{cccccccccc}
 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & x \\
\oplus & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & y \\
\hline
 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & z
\end{array}
$$

即 x ⊕ y ＝ 01100111

2．5．2 多功能算术/逻辑运算单元

1、基本思想

将A_i和B_i先组合成由控制参数S_0，S_1，S_2，S_3控制的组合函数X_i和Y_i，然后再将X_i、Y_i、C_i通过全加器进行全加。

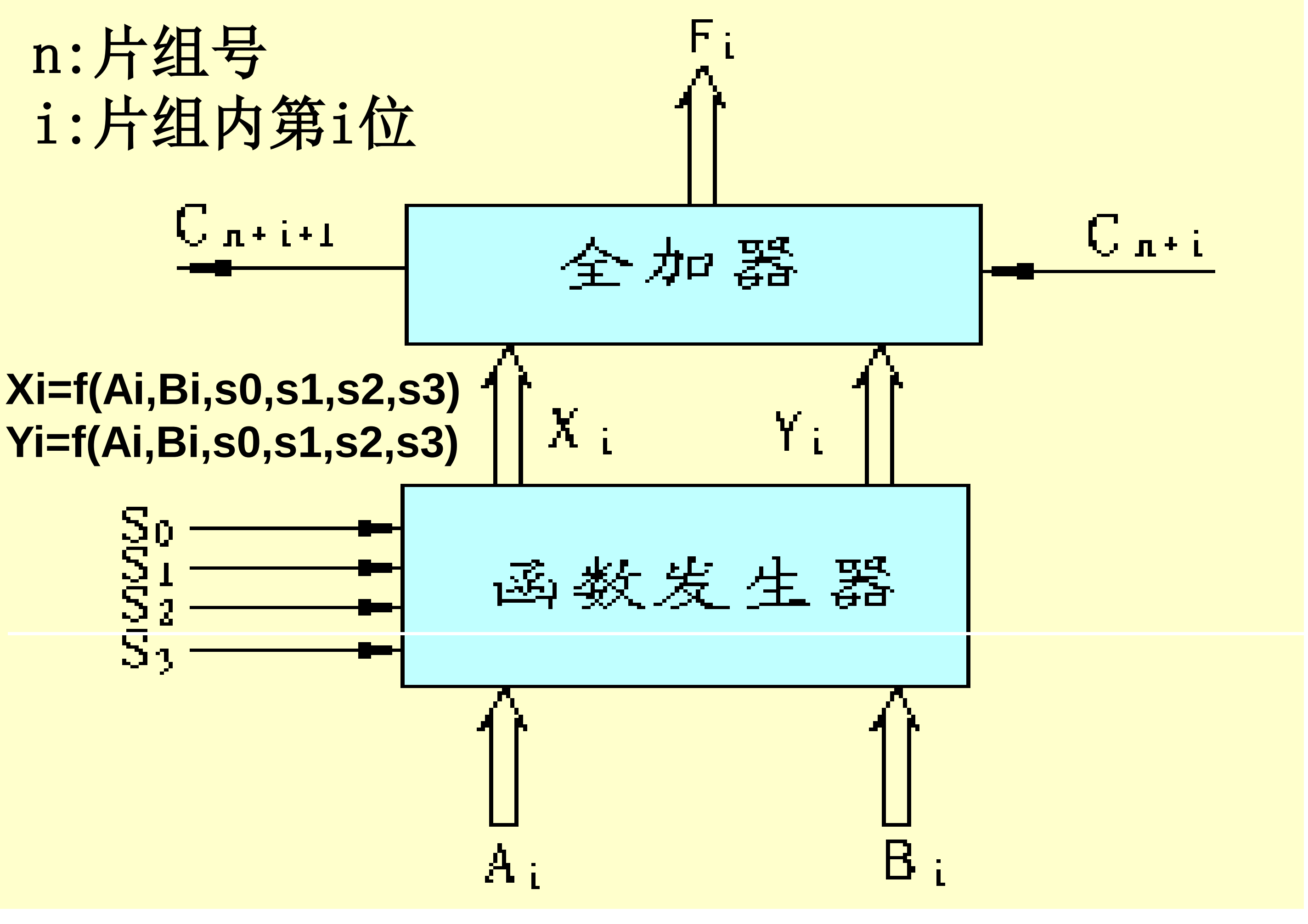
不同控制参数可以得到不同的组合函数，因而能够实现多种运算。

如下图所示。

2．5．2 多功能算术/逻辑运算单元

$$F_i = X_i \oplus Y_i \oplus C_{n+i}$$

$$C_{n+i+1} = X_iY_i + Y_iC_{n+i} + C_{n+i}X_i$$



2．5．2 多功能算术/逻辑运算单元

1、基本思想

X_i, Y_i 与控制参数和输入量的关系

$S_0\ S_1$	Y_i	$S_2\ S_3$	X_i
0 0	$\bar{A}_i$	0 0	1
0 1	$\bar{A}_i B_i$	0 1	$\bar{A}_i + \bar{B}_i$
1 0	$\bar{A}_i \bar{B}_i$	1 0	$\bar{A}_i + B_i$
1 1	0	1 1	$\bar{A}_i$

$$X_i = \bar{S}_2\bar{S}_3 + \bar{S}_2 S_3（\bar{A}_i + \bar{B}_i） + S_2\bar{S}_3（\bar{A}_i + B_i） + S_2S$$

$$Y_i = \bar{S}_0\bar{S}_1\bar{A}_i + \bar{S}_0 S_1 \bar{A}_i B_i + S_0\bar{S}_1\bar{A}_i\bar{B}_i + S_0S_1$$

2. 5. 2 多功能算术/逻辑运算单元

1、基本思想

进一步化简，可以得到：

$$X_i=\overline{S_3A_iB_i + S_2A_i\overline{B_i}}$$

$$Y_i=\overline{A_i+S_0B_i+S_1\overline{B_i}}$$

$$F_i=Y_i \oplus X_i \oplus C_{n+i}$$

$$C_{n+i+1}=Y_i+X_iC_{n+i}$$

Xi、Yi只与**Si、Ai、Bi**有关，可以根据**A、B、S**直接求得。

2．5．2 多功能算术/逻辑运算单元

2、并行进位

可以看出：

Cn+1、 Cn+2、 Cn+3、 Cn+4只与A、B、S、 Cn有关，都可以同时直接求出来，不需要串行传递，这就是并行进位。

Cn是第n-1片组向第n片组的进位。

第0位向第1位的进位：

$$C_{n+1}=Y_0+X_0C_n$$

第1位向第2位的进位：

$$C_{n+2}=Y_1+X_1C_{n+1}=Y_1+Y_0X_1+X_0X_1C_n$$

第2位向第3位的进位：

$$C_{n+3}=Y_2+X_2C_{n+2}=Y_2+Y_1X_2+Y_0X_1X_2+X_0X_1X_2C_n$$

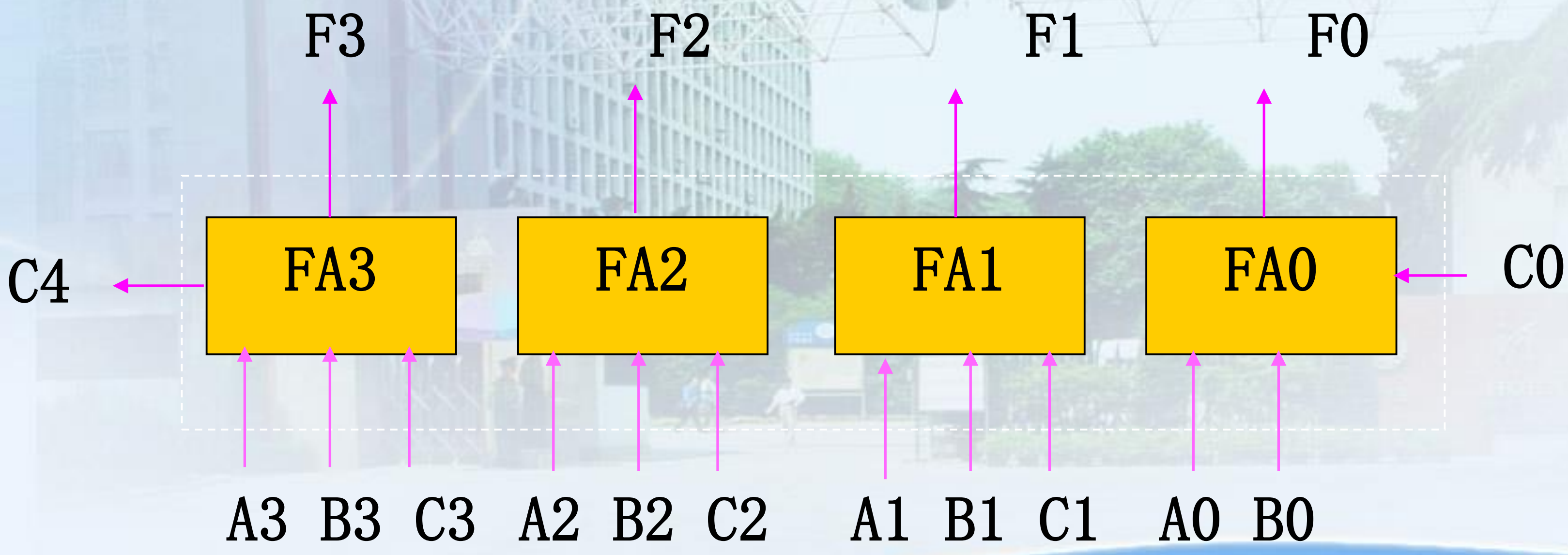
第3位向第4位的进位：

$$C_{n+4}=Y_3+X_3C_{n+3}=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3+X_0X_1X_2X_3C_n$$

2．5．2 多功能算术/逻辑运算单元

2、并行进位

下面不考虑函数发生器，直接从全加器F=A+B推导，也可以得出同样的结论。



2. 5. 2 多功能算术/逻辑运算单元

2、并行进位

$C_{i+1} = A_iB_i + (A_i + B_i) * C_i$

设 $G_i = A_iB_i$、$P_i = A_i + B_i$，

则 $C_{i+1} = G_i + P_i C_i$

Gi：表示第**i**位的两个数都是**1**，自然有进位，称为进位产生函数。

Pi：表示第**i**位的两个数中有一个**1**，当低位来的进位是$C_i = 1$的时候，也会有进位，称为进位传递函数。

2.5.2 多功能算术/逻辑运算单元

2、并行进位

第0位向第1位的进位：

$$C_1 = G_0+P_0C_0$$

第1位向第2位的进位：

$$C_2 = G_1+P_1C_1$$

$$= G_1+P_1*(G_0+P_0C_0)=G_1+P_1G_0+P_1P_0C_0$$

可解释

第2位向第3位的进位：

$$C_3 = G_2+P_2C_2= G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$$

第3位向第4位的进位：

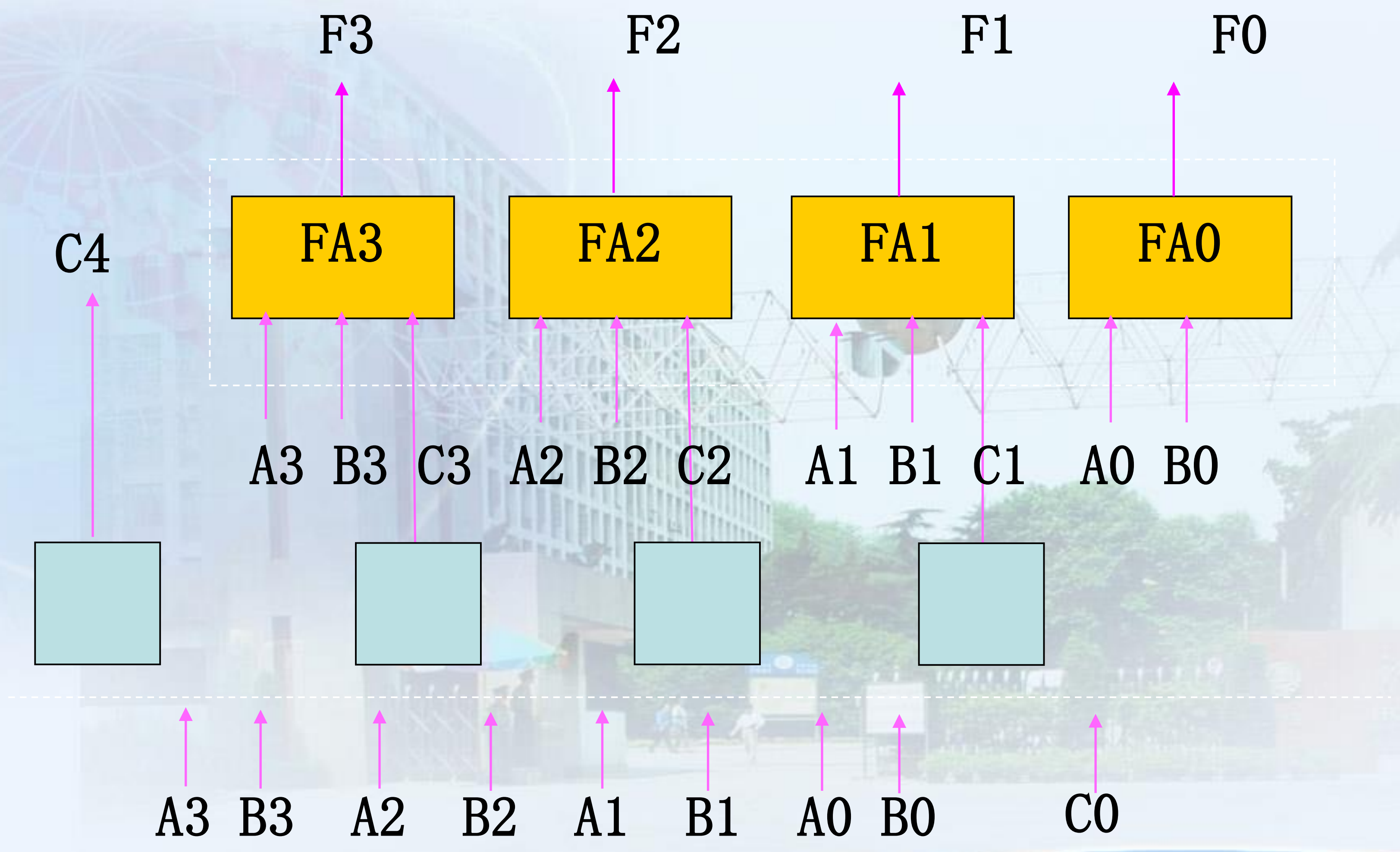
$$C_4 = G_3+P_3C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$$

可以看出：

C1、C2、C3、C4只与A、B、C0有关，都可以同时直接求出来，不需要串行传递，这就是并行进位。

2．5．2 多功能算术/逻辑运算单元

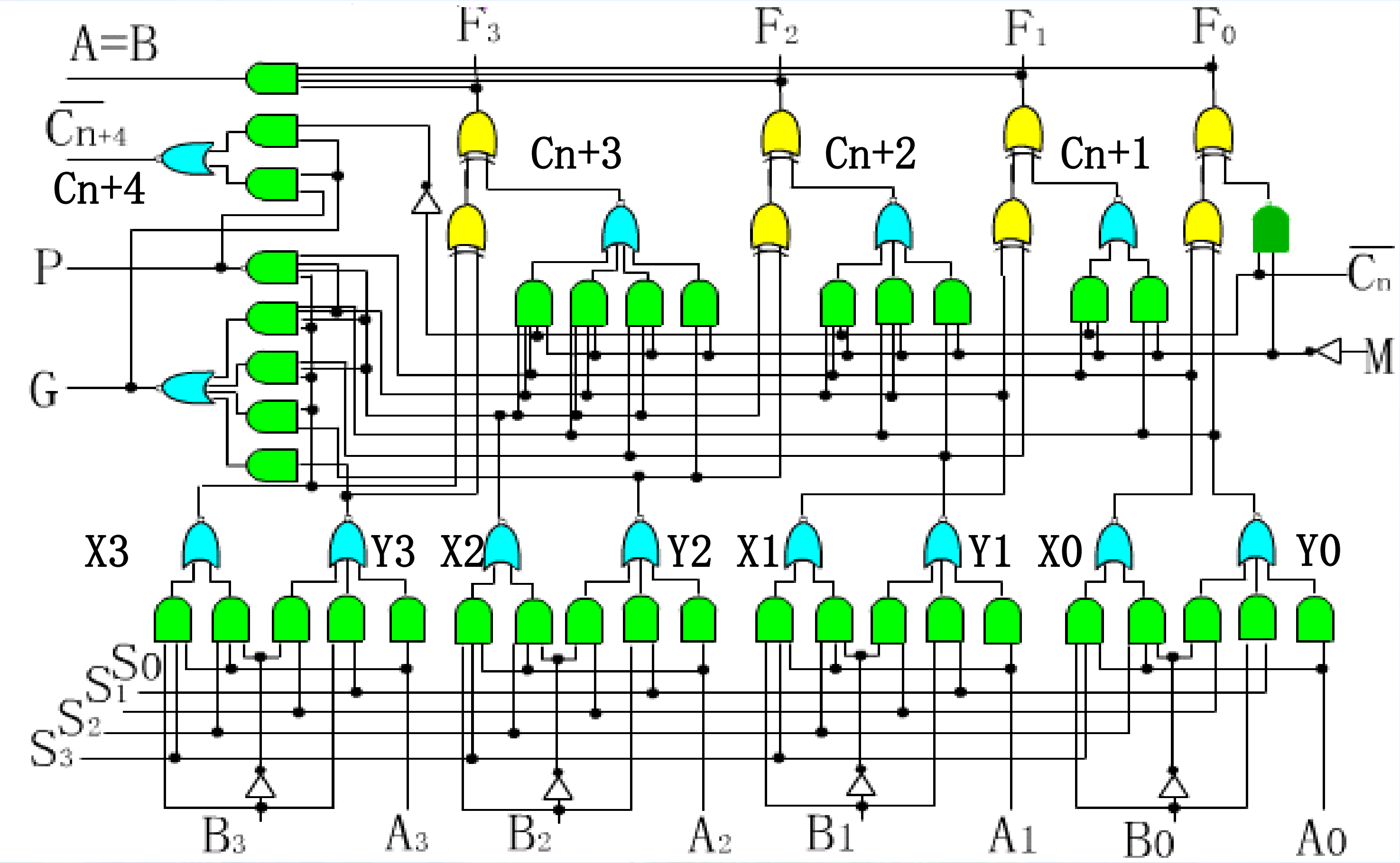
2、并行进位



2.5.2 多功能算术/逻辑运算单元

3、算术逻辑运算的实现

全加器
并行进位电路



函数发生器
A、B变换X、Y

M=1，Fi=Xi ⊕Yi，可变换为各种逻辑运算

M=0，Fi=Xi ⊕Yi ⊕Ci，算术运算

2．5．3 内部总线

1、总线结构

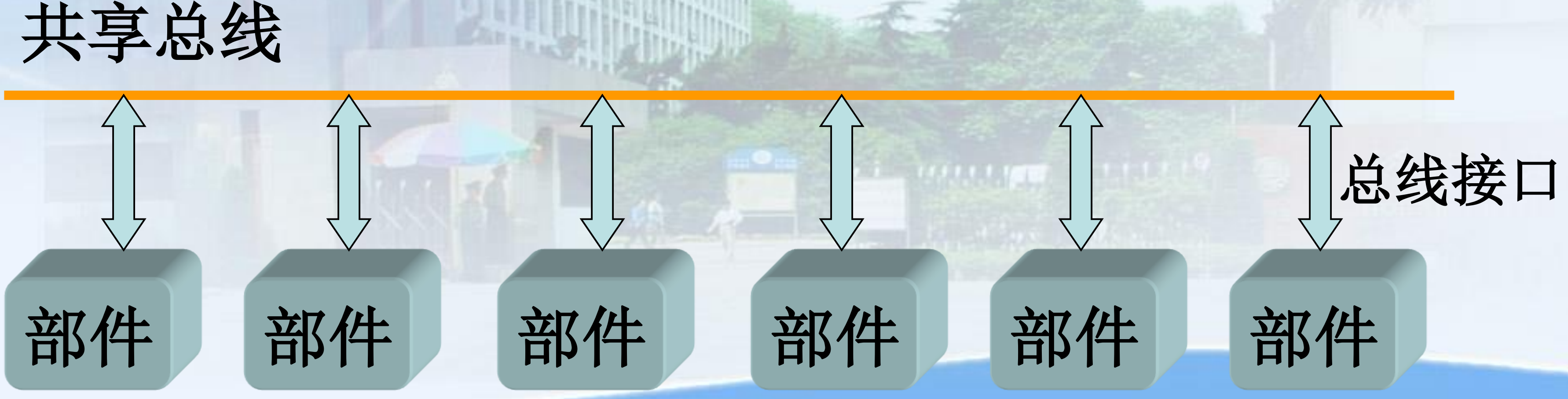
计算机各部件之间的数据传送非常频繁，如果任何两个部件之间都建立自己的数据传输通路，则传送线太多、控制复杂、扩展不方便。

为了减少传送线并便于控制，通常将部件之间的传送线加以归并，建立公用的共享的信息传输通路，组成总线结构，使不同信息源可以在总线结构上分时传送。

2.5.3 内部总线

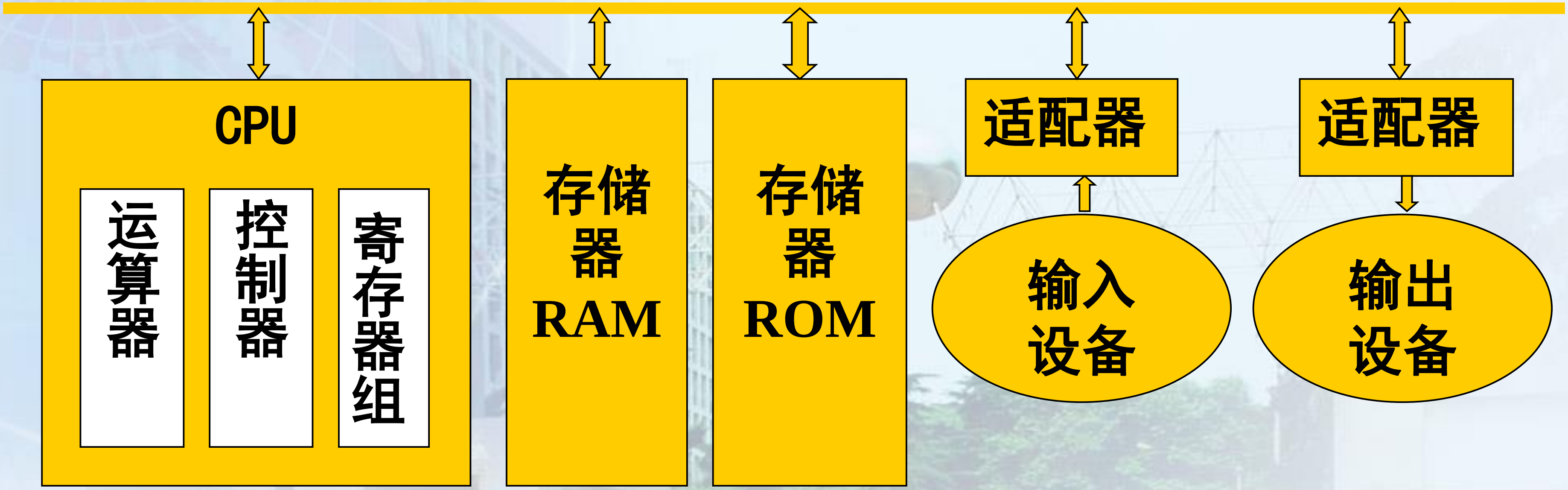
1 总线结构

定义：总线是一组能为多个部件分时共享的信息传送通路，用来连接多个部件并为之提供信息传输交换服务。



2.5.3 内部总线

1 总线结构



易扩展性：总线结构中，增加、删除部件方便。

2.5.3 内部总线

1 总线结构

共享性：总线所连接的所有部件都可以通过它传递信息。

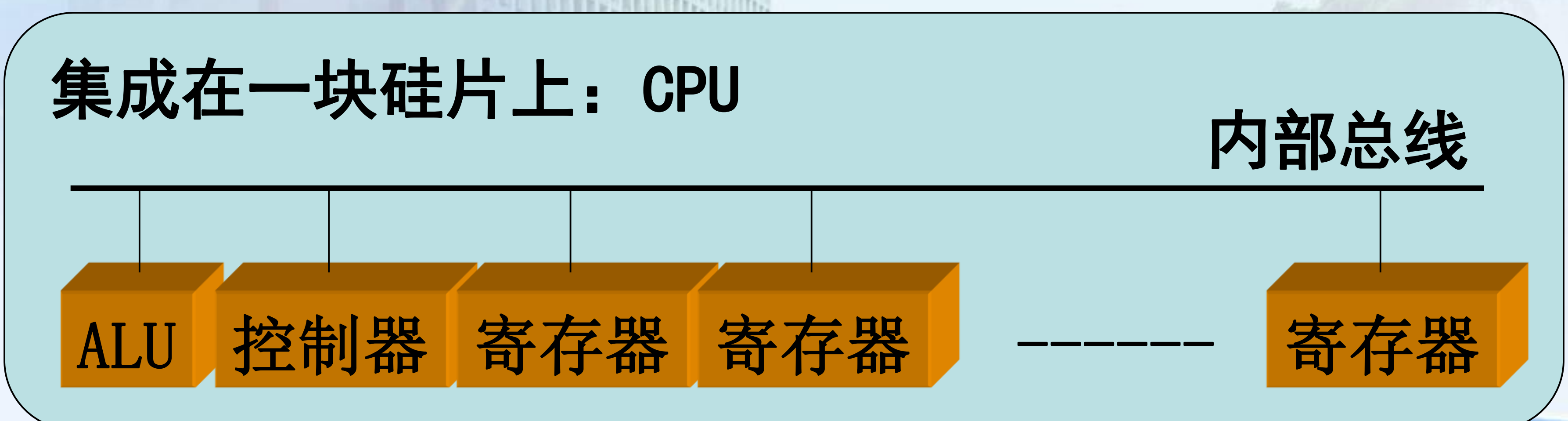
分时性：在某一个时刻总线只允许一个部件发送信息到总线上。显然，共享是分时实现的。

总线协议：总线不仅是一组传输线路，同时连接到总线上的所有部件都必须共同遵守一组规则和约定，称为总线协议（Protocol）。它一般包括信号线定义、数据格式、时序关系、信号电平、控制逻辑等。

2．5．3 内部总线

2、内部总线、外部总线

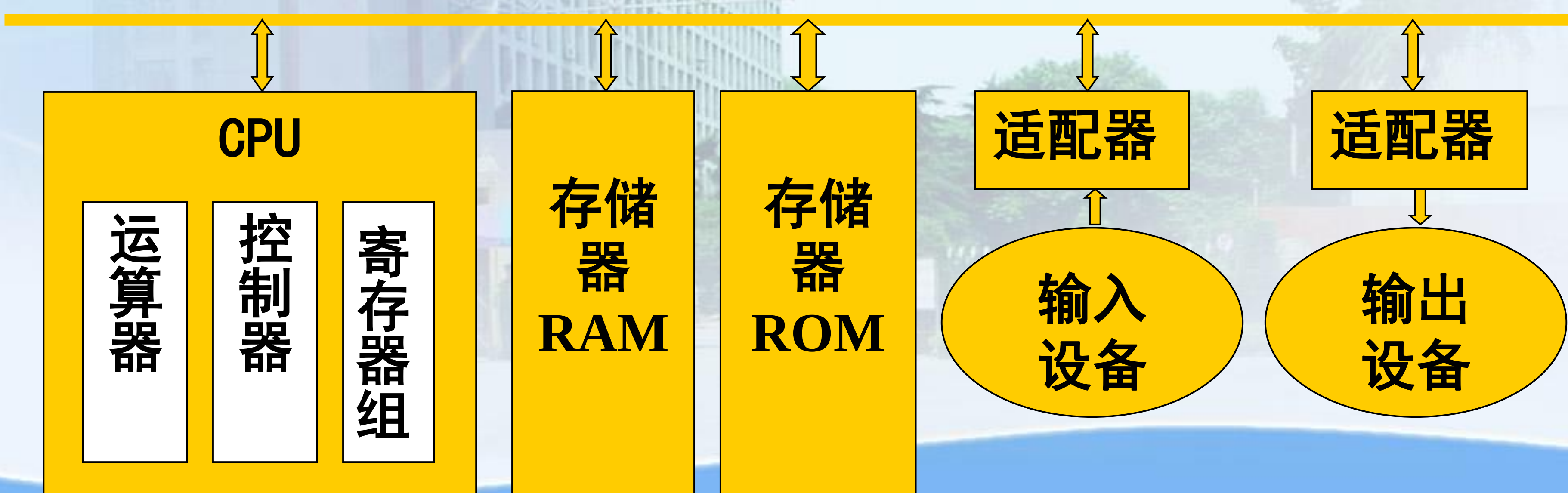
内部总线：CPU内部各部件（算术/逻辑运算单元ALU、控制器、寄存器）之间的连线，即公共信息交换通路。



2．5．3 内部总线

2、内部总线、外部总线

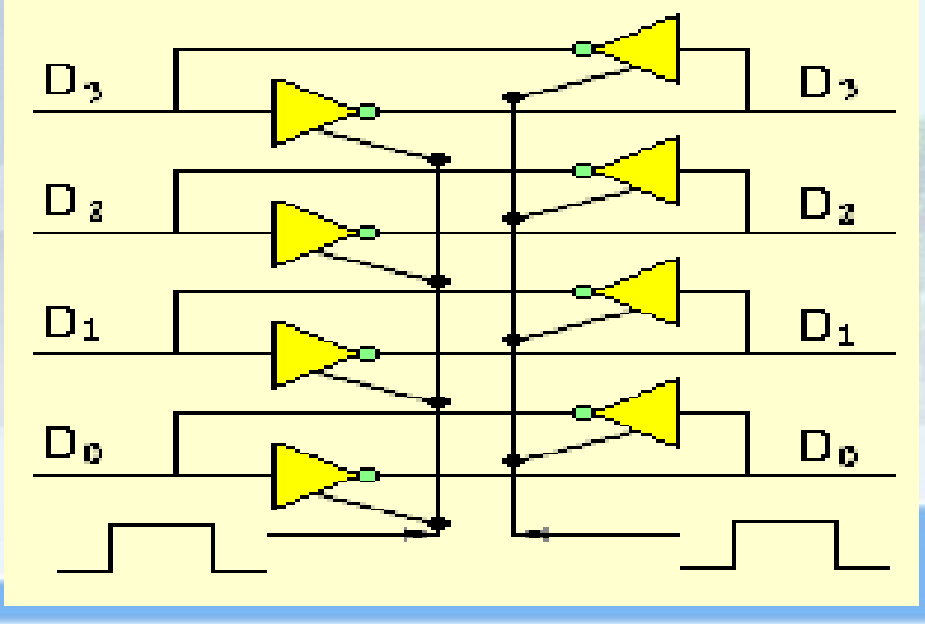
外部总线：也称系统总线，CPU与内存储器、输入输出设备之间的连线，即公共信息交换通路。



2．5．3 内部总线

3、单向传输总线、双向传输总线

实际上，在三总线（地址、控制、数据）结构中，地址线是单向的，数据线是双向的，控制线有的单向、有的双向。

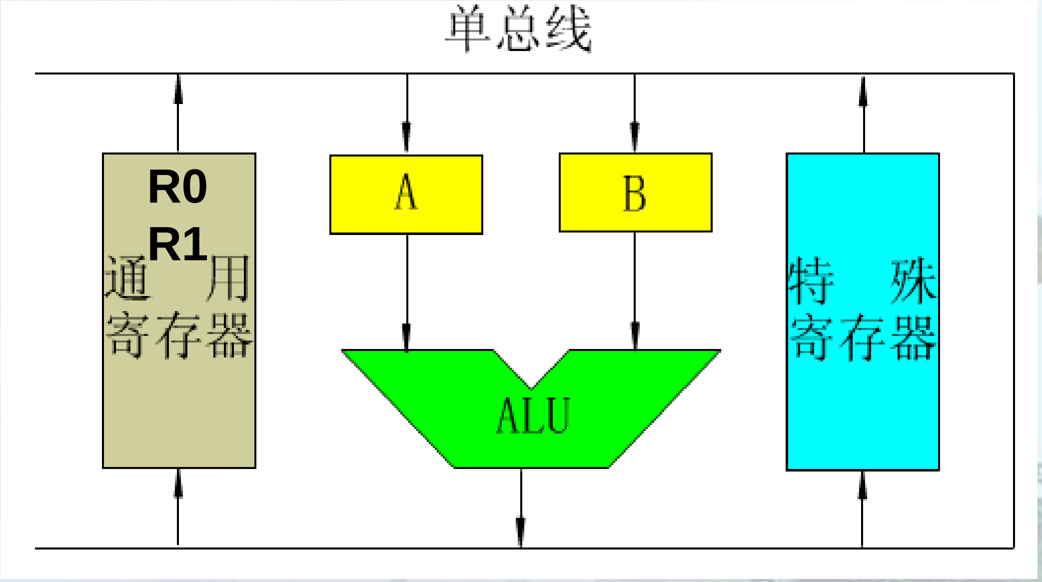


发送选通信号

接收选通信号

2．5．4 定点运算器的基本结构

1、单总线结构的运算器



指令：ADD R0，R1

含义：R0+R1→R0

执行过程（3步）：

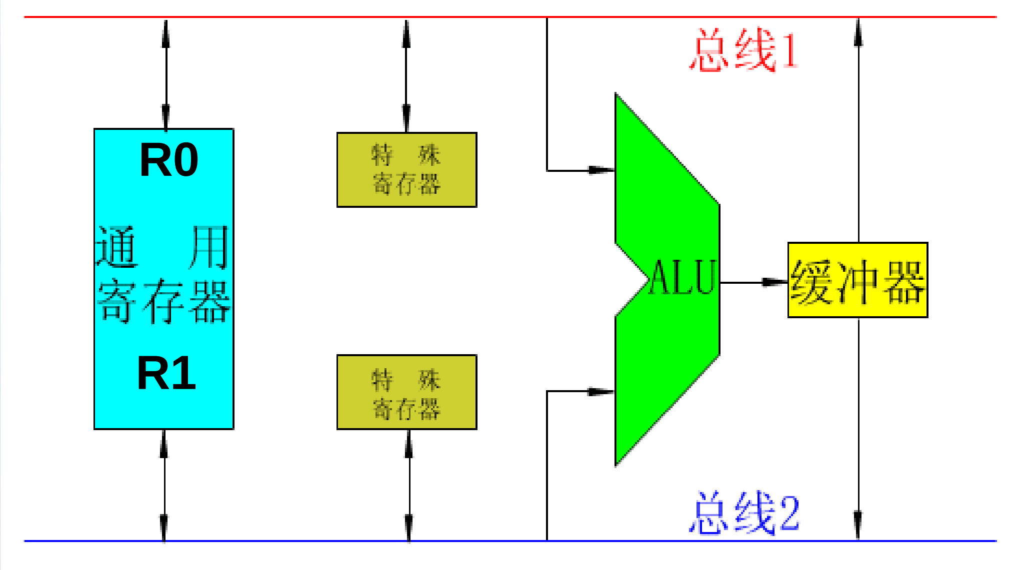
R0 →A

R1 →B

A+B→R0

2．5．4 定点运算器的基本结构

2、双总线结构的运算器



指令：ADD　R0，R1　　含义：R0+R1→R0

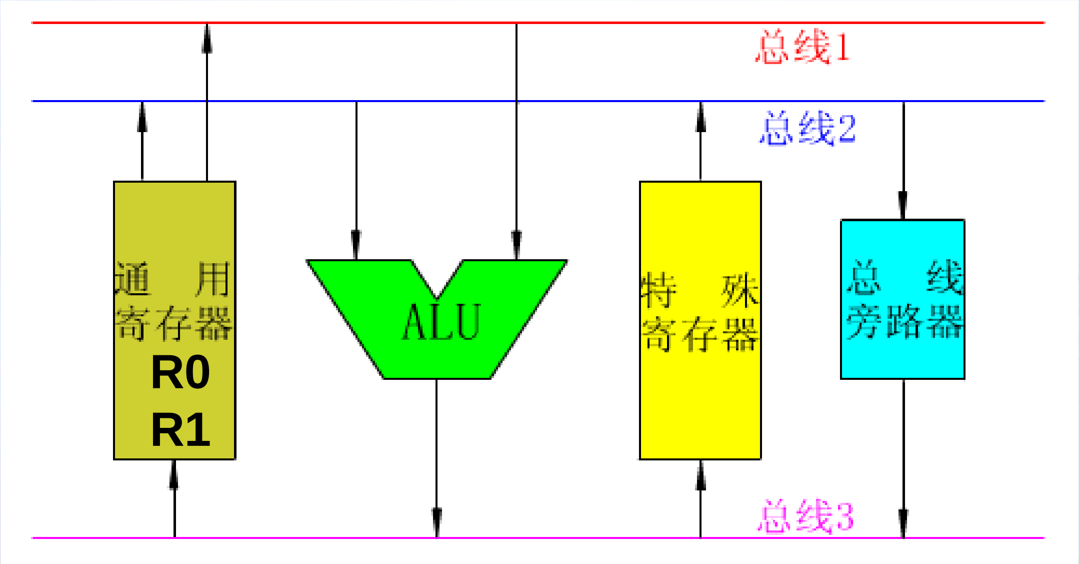
执行过程（2步）：

R0 →总线1 → ALU，　R1→总线2 → ALU；

ALU运算 →总线1 → R0

2．5．4 定点运算器的基本结构

3、三总线结构的运算器



指令：ADD R0，R1 含义：R0+R1→R0

执行过程（2步）：

R0 →总线1 → ALU， R1→总线2 → ALU；

ALU运算 →总线3 → R0

2.6 浮点运算方法和浮点运算器

浮点加减法运算

浮点运算流水线

2．6．1 浮点加法、减法运算

设有两个浮点数 x 和 y，它们分别为：

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

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

则　$z = x \pm y = (M_x \cdot 2^{E_x - E_y} \pm M_y) \cdot 2^{E_y}$　　$E_x \leqslant E_y$

显然，浮点加减法运算的操作步骤大体如下：

第一步：0操作数检查

第二步：比较阶码大小并完成对阶

第三步：尾数加减运算

第四步：结果规格化并进行舍入处理

2.6.1 浮点加法、减法运算

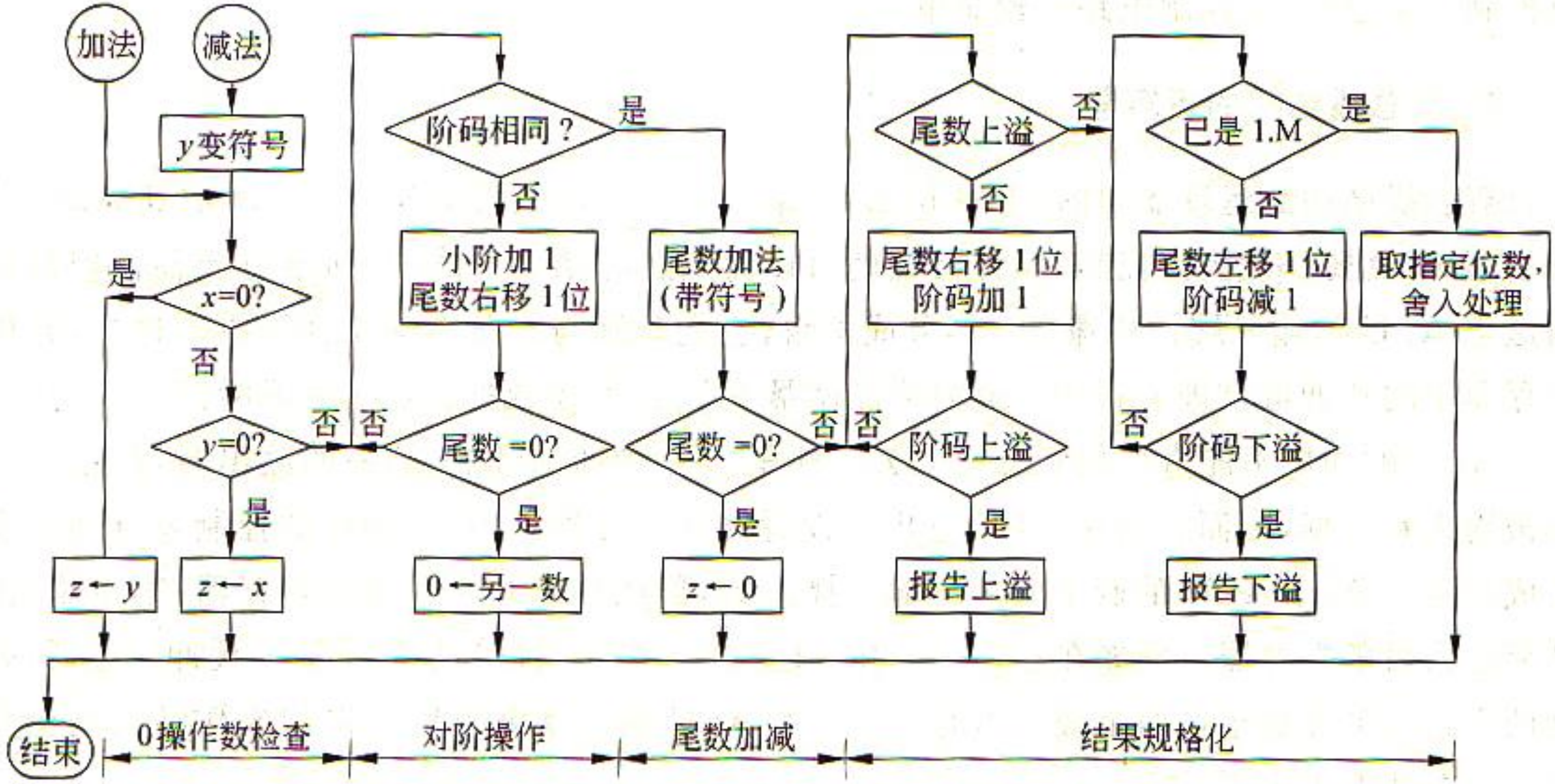


图 2.17 浮点加减运算的操作流程

可以看出，浮点数加减法运算的阶段性非常明显，适合重叠操作（流水线处理）。

2.6.1 浮点加法、减法运算

1、 0操作数检查

在运算开始或运算过程中，如果判知两个操作数 x 或 y 中有一个数为0，即可直接得出运算结果。

2、比较阶码大小并完成对阶

计算阶差；

对阶：小阶向大阶看齐，尾数右移，移掉的是尾数的低位部分。

2. 6. 1 浮点加法、减法运算

3、尾数加减运算

尾数加减法运算采用定点小数补码加减法运算规则、溢出判断规则。运算结果可能出现两种情况，需要进行处理：

（1）溢出：加减法运算结果上溢/下溢，并不一定代表整个浮点数溢出，可以通过阶码进行调整。阶码调整有可能造成阶码上溢/下溢，即报告溢出。

（2）非规格化：需要进行规格化处理。

2. 6. 1 浮点加法、减法运算

4、结果的规格化处理

尾数左移、减少阶码、直到规格化为止。

减少阶码，有可能导致阶码下溢，则报告溢出。

5、舍入处理

0舍1入法：如果被丢掉数位的最高位为0则舍去，为1则将尾数的末位加1。

恒置一法：只要有数位被移掉，就确保尾数末尾始终为1。

- 浮点加、减法运算步骤
  - 零操作数检查
  - 比较阶码大小并对阶
  - 带符号的尾数求和
  - 规格化
  - 舍入处理
  - 溢出处理

第2章 运算方法和运算器

- 例：设$x=2^{010}\times 0.11011011$，$y=2^{100}\times(-0.10101100)$，求$x+y$
  - （1）比较阶码大小并对阶
    - $x=2^{100}\times 0.0011011011$，$x$的尾数用补码表示为00011011011
      - 蓝色表示符号位，红色表示可能因精度问题被丢弃的位，下同
    - y的尾数用补码表示为101010100。注意：计算机中，尾数的补码表示中不存在“0.”或“1.”，符号位在最前面
      - IEEE 754浮点格式：
      - 课本例题采用格式：

IEEE 754浮点格式：	符号位	阶码（含阶符）	尾数M，前面缺省1.
课本例题采用格式：	符号位	阶码（含阶符）	尾数M，前面缺省0.

      - 注意：补码是机器内部计算时的表示，上述格式中M为真值
  - （2）带符号的尾数求和，补码结果为11000101011

– （3）规格化→目的：把真值表示成$2^E\times1.M/0.M$的格式

- 人工判向：$x+y$的真值为$2^{100}\times(-0.0111010101)$，按IEEE 754的格式须改写为$2^{010}\times(-1.11010101)$，也就是左规两位；若按0.M的格式，可写为$2^{011}\times(-0.111010101)$。规格化后的机器表示如下（假设指数采用IEEE 754移码表示）：
  - IEEE 754浮点格式：
  - 课本例题采用格式：

IEEE 754浮点格式：	1	10000001（移码）	11010101
课本例题采用格式：	1	10000010（移码）	111010101

  - 如果不考虑舍入误差，上述两种表示方法完全等价
    - » 第一种表示方式的真值：$(-1)^1\times2^{129-127}\times(1.11010101)$
    - » 第二种表示方式的真值：$(-1)^1\times2^{130-127}\times(0.111010101)$
  - 如果考虑舍入误差，假设尾数部分只能保留8位
    - » 第一种表示方式更精确。在本例中，第一种表示方式不会涉及舍入问题。

- （3）规格化→目的：把真值表示成$2^E \times 1.M/0.M$的格式
  - 机器判向：按课本提法，带符号的尾数求和出现“11xxx”、“00xxx”两种结果时执行左规。
    - 反例：令$x=2^{100}\times 0.00100100$，$y=2^{100}\times(-0.10100100)$。这里不存在对阶的问题，$y$的带符号尾数补码表示为101011100，带符号的尾数和用补码表示为110000000（对应真值为-0.10000000，按0.M格式不需要左规），最终结果为$x+y=2^{100}\times(-0.10000000)$。
  - 操作方式：基于补码运算结果进行，过程为1100010101l→11000101011，最终结果为$2^{010}\times(-1.11010101)$
    - 灰色表示规格化过程中丢弃的位
    - 若按0.M的格式进行规格化，判别方向时只左规一位，规格化过程为11000101011→ 11000101011，最终结果为$2^{011}\times(-0.111010101)$。
    - 第三道报告题：机器进行规格化的正确流程。

– （4）舍入处理→目的：对真值进行处理
- 就近舍入、朝0舍入、朝+∞舍入、朝-∞舍入…
- 本例舍入前：按0.M的格式进行规格化，判别方向时只左规一位，规格化过程为1100010101 1→ 11000101011，最终结果真值为$2^{011}\times(-0.111010101)$
  – 就近舍入：舍入后真值$2^{011}\times(-0.11101010)$ ——4舍6入，5前凑偶数
  – 四舍五入：舍入后真值$2^{011}\times(-0.11101011)$ ——0舍1入
  – 朝0舍入：舍入后真值$2^{011}\times(-0.11101010)$
  – 朝+∞舍入：舍入后真值$2^{011}\times(-0.11101010)$
  – 朝-∞舍入：舍入后真值$2^{011}\times(-0.11101011)$

– （5）溢出处理→按需调整尾数，观察阶码并处理
- 尾数溢出判断：双符号位（$S_1 \oplus S_2$为1时有溢出）、单符号位（$C_S \oplus C_{最高有效位}$为1时有溢出）

2.6.3 浮点运算流水线

1、流水线原理

计算机的流水处理过程同工厂中的流水装配线类似。

把输入的任务分割为一系列的子任务，使各子任务在流水线中时间重叠、并发执行，任务源源不断地进入流水线。

2.6.3 浮点运算流水线

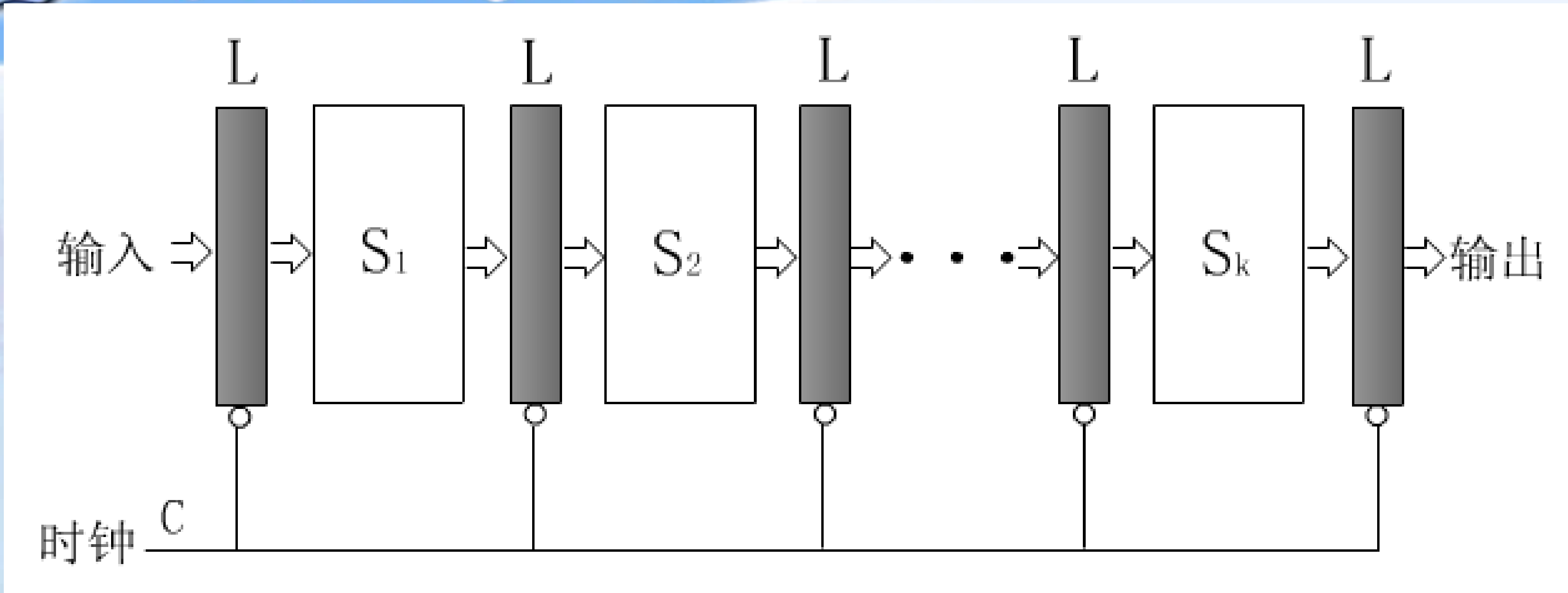
1、流水线原理

任务 T 被分成 k 个子任务，可表达为 $T = \{T_1, T_2, \cdots, T_k\}$。

各个子任务之间有一定的优先关系：若 $i<j$，则必须在 T_i 完成以后，T_j 才能开始工作。

具有这种线性优先关系的流水线称为线性流水线。

2.6.3 浮点运算流水线

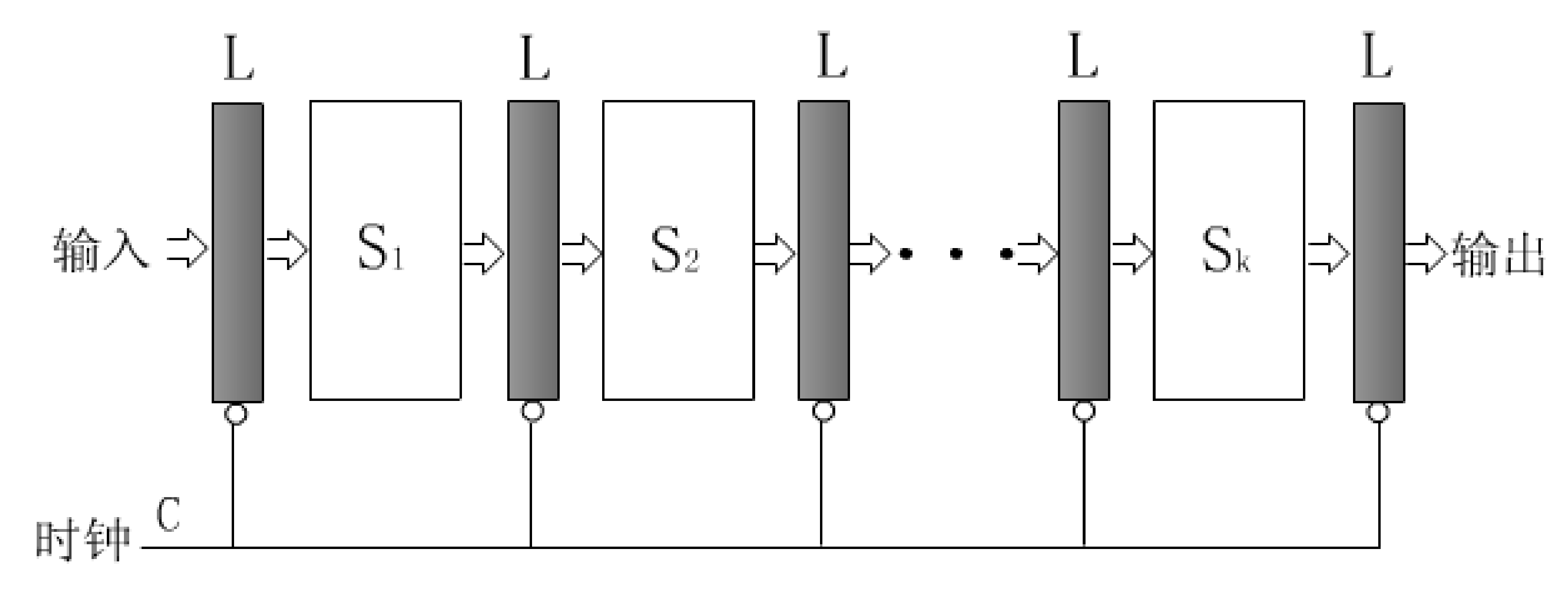


过程段S_i：对应每个划分的子任务。

缓冲寄存器L：暂时保存前一过程段的处理结果。

时钟C：统一时钟信号，每个时钟周期流动一次，即所有任务从上个过程段流向下个过程段。

2.6.3 浮点运算流水线



显然：子任务划分，是影响流水线性能的关键因素。原则上要求各个子任务的处理时间相同。若某个子任务的处理时间较长，势必造成其他阶段的部分空转等待。

2.6.3 浮点运算流水线

1、流水线原理

时钟周期：

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

$T=\max\{\tau_i\}+\tau_l=\tau_m+\tau_l$。

2．6．3 浮点运算流水线

1、流水线原理

k个过程段处理n个任务需要的时间：

t =（K+（n-1））T

K:第一个任务需要k个时钟周期才能输出。k个时钟周期以后，流水线满载 ；

n-1:满载后，每个时钟周期就可以输出一个任务，剩余的n－1个任务只需n－1个时钟周期。

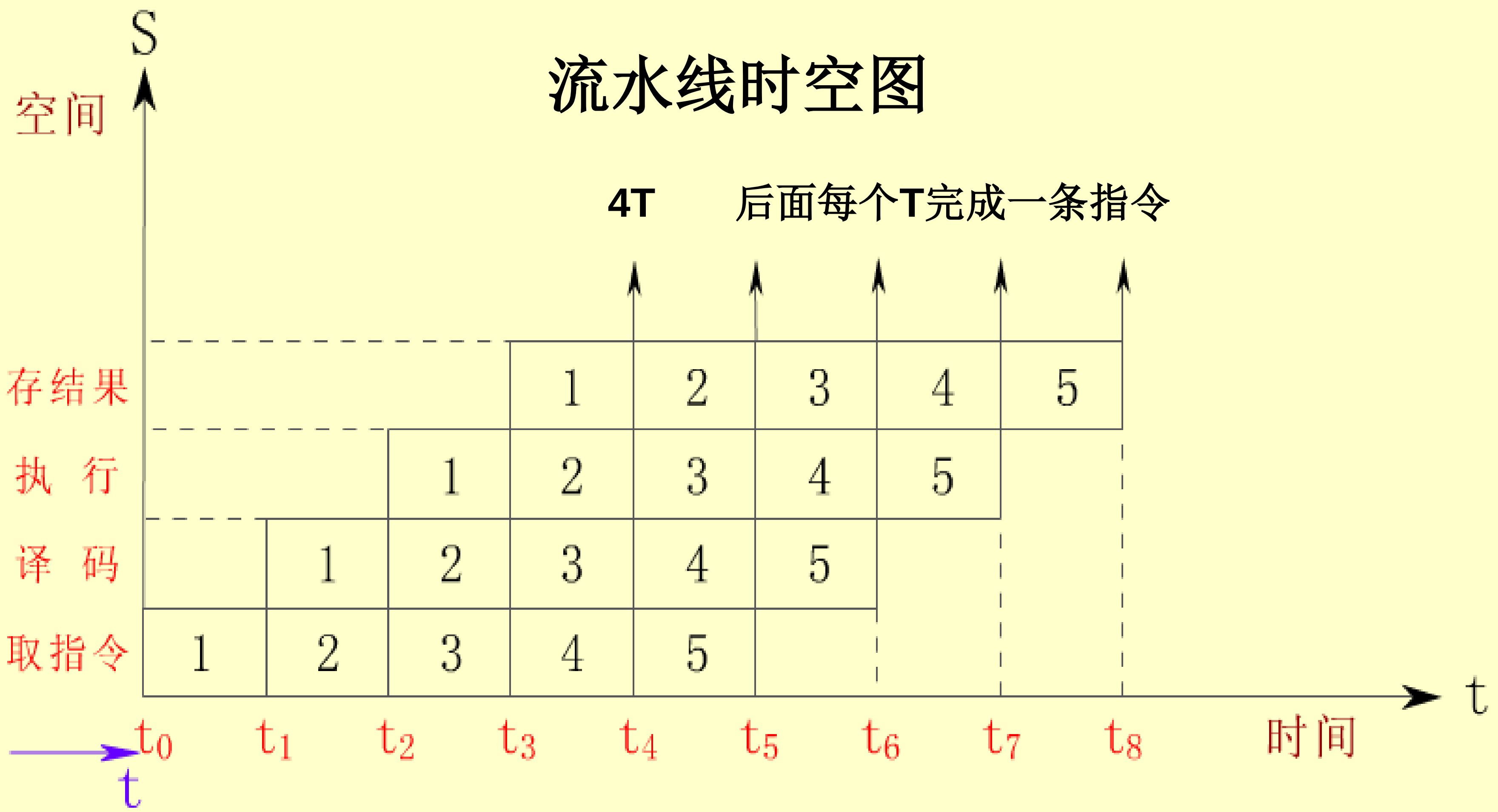
2. 6. 3 浮点运算流水线

	S1	S2	S3	S4
t1:	T11			
t2:	T21	T12		
t3:	T31	T22	T13	
t4:	T41	T32	T23	T14
t5:	T51	T42	T33	T24
t6:	T61	T52	T43	T34
t7:	T62	T53	T44	
t8:		T63	T54	
T9:				T64

k*T

(n-1)*T

2. 6. 3 浮点运算流水线



流水线时空图
S
空间
4T
后面每个T完成一条指令
存结果
执 行
译 码
取指令
1 2 3 4 5
t
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
时间

2．6．3 浮点运算流水线

1、流水线原理

K级线形流水线的加速比C_k：

不采用流水线，顺序完成n个任务所需要的时间$T_L=n \cdot k \cdot T$。

采用线形流水线，并行完成n个任务所需要的时间$T_k=(k+(n-1)) \cdot T$。

2.6.3 浮点运算流水线

1、流水线原理

K级线形流水线的加速比C_k：

T_L和T_k的比值定义为k级线性流水线的加速比：

$$C_k = T_L / T_k = (n \cdot k) / (k + (n-1))$$

当 n>>k 时，$C_k \rightarrow k$，理论上可以提高k倍速度，但实际上由于资源冲突、数据相关，这个理想的加速比是达不到的。

2．6．3 浮点运算流水线

2、流水线浮点加法器

我们知道，浮点数加法操作包括0操作数检查、对阶、尾数运算、结果溢出处理、结果规格化处理、舍入处理等处理过程。

为了提高浮点加法的运算处理速度，完全可以采用流水线技术，流水线浮点加法器就是在此基础上提出来的。

2．6．3 浮点运算流水线

2、流水线浮点加法器

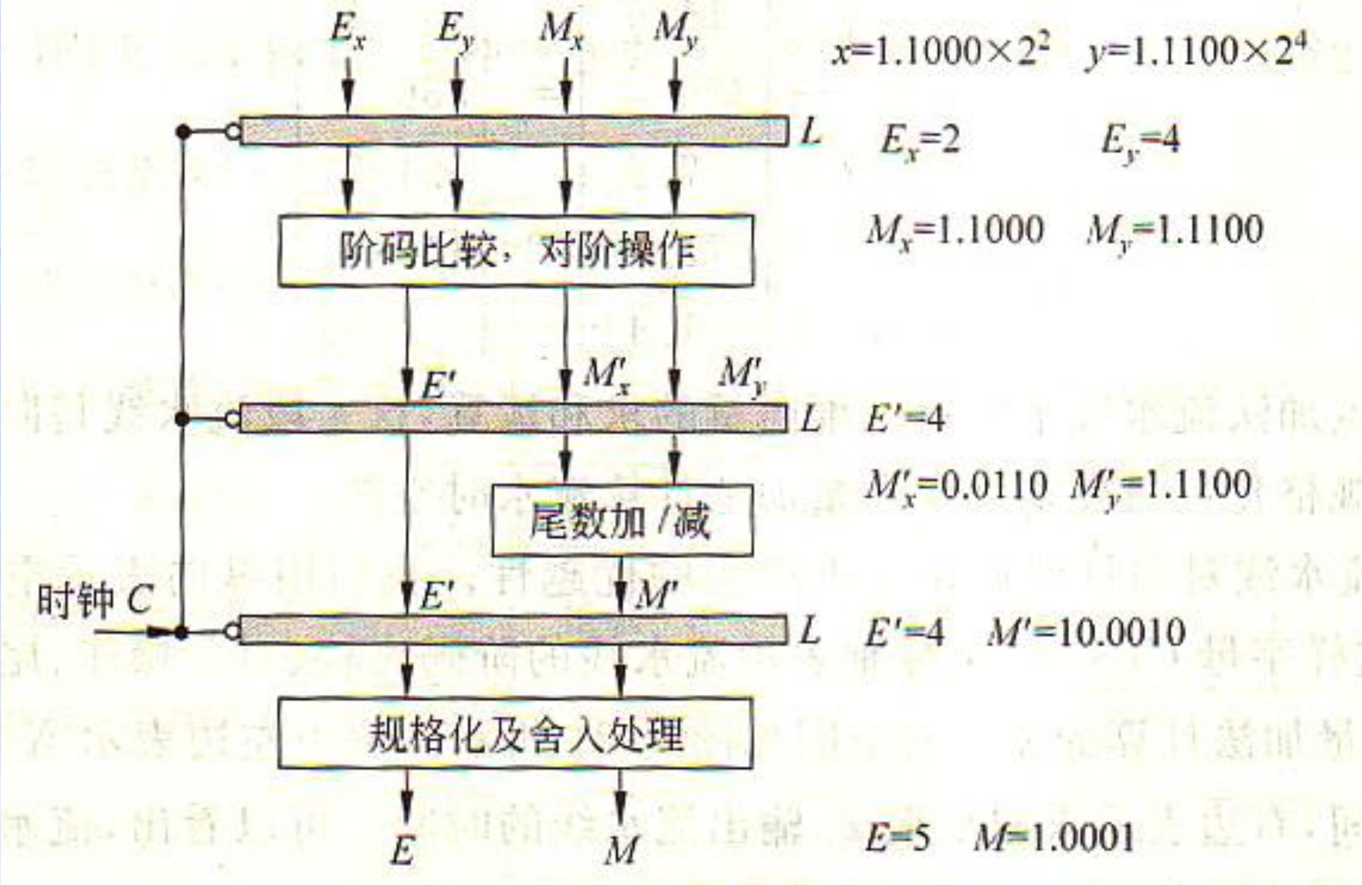


图 2.20 流水线浮点加法器框图

2．6．3 浮点运算流水线

[例32] 在4级流水线加法器中，(1)假设每个过程段所需的时间为：求阶差 τ_1＝70ns, 对阶 τ_2＝60ns, 相加 τ_3＝90ns, 规格化 τ_4＝80ns, 缓冲寄存器L的延时为 t_1＝10ns, 求4级流水线加法器的加速比为多少？(2)如果每个过程段的时间相同，即都为75ns(包括缓冲寄存器时间)，加速比是多少？

2.6.3 浮点运算流水线

[解]

(1)加法器的流水线时钟周期为

$$\tau = 90ns + 10ns = 100ns$$

不采用流水线方式，则浮点加法所需的时间为

$$\tau_1 + \tau_2 + \tau_3 + \tau_4 = 300ns$$

因此，4级流水线加法器的加速比为

$$C_k = 300/100 = 3$$

(2) 当每个过程段的时间都是75ns时，加速比为

$$C_k = (75 \times 4)/75 = 4$$

注：**与加速比定义有点不一致，理解：当任务数n很大时，采用流水线一个时钟周期完成一个任务，而非流水线$\tau_1+\tau_2+\tau_3+\tau_4$ 完成一个任务。**

2. 6. 3 浮点运算流水线

［例33］已知计算一维向量x, y的求和表达式如下，试用4段的浮点加法流水线来实现一维向量的求和运算，这4段流水线是阶码比较、对阶操作、尾数相加、规格化。只要求画出向量加法计算流水时空图。

$$
\begin{array}{ccccc}
x & & y & & z \\
\begin{bmatrix} 56 \\ 20.5 \\ 0 \\ 114.3 \\ 69.6 \\ 3.14 \end{bmatrix} & + & \begin{bmatrix} 65 \\ 14.6 \\ 336 \\ 7.2 \\ 72.8 \\ 1.41 \end{bmatrix} & = & \begin{bmatrix} 121 \\ 35.1 \\ 336 \\ 121.5 \\ 142.4 \\ 4.55 \end{bmatrix}
\end{array}
$$

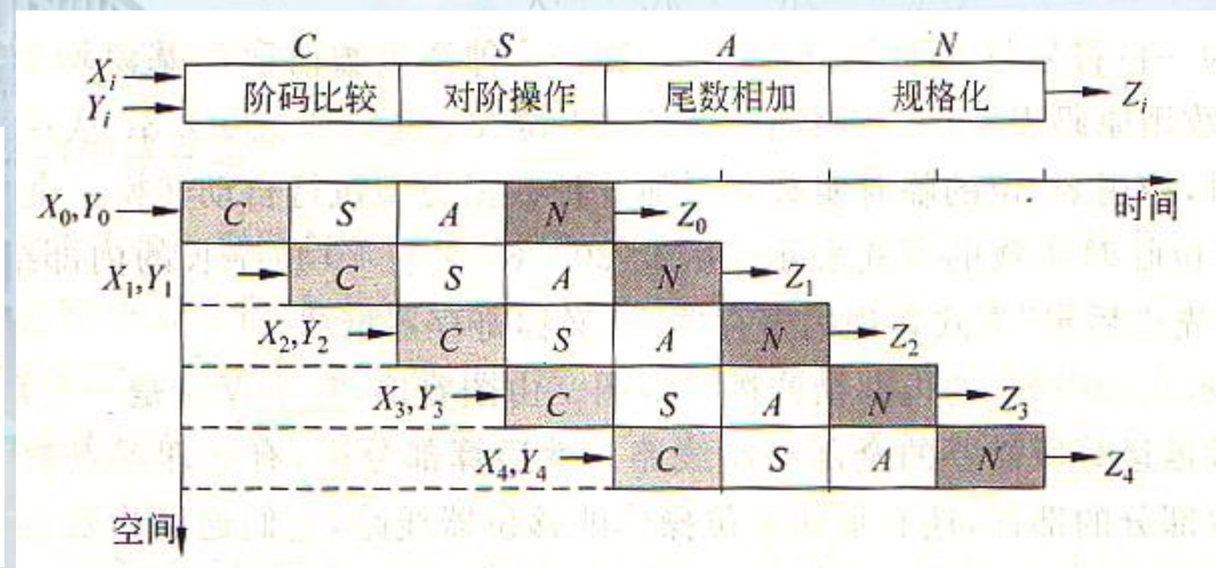


图 2.21 向量加法计算的流水时空图

河海大学

第2章习题

- 1. 写出下列各整数的原码、反码、补码表示（用8位二进制数），其中最高位为符号位。

真值	原码	反码	补码
-35			
128	X	X	X
127			
-127			
-1			

第2章习题

- 5. 已知x和y，用变形补码计算x + y，同时指出结果是否溢出。
  - （1） x = 11011， y = 00011
  - （2） x = 11011， y = -10101
  - （3） x = -10110， y = -00001

第2章习题

- 9. 设阶码3位，尾数6位，按浮点运算方法，计算x+y和x-y。
  - （1） $x = 2^{-011} \times 0.100101$， $y = 2^{-010} \times (-0.011110)$
  - （2） $x = 2^{-101} \times (-0.010110)$， $y = 2^{-010} \times 0.010110$

- 11. 某加法器进位链小组信号为$C_4C_3C_2C_1$，低位来的进位信号为C_0，请分别按下述两种方式写出$C_4C_3C_2C_1$的逻辑表达式：
  - （1）串行进位方式
  - （2）并行进位方式

河海大学

Q&A