# 第2章 运算方法和运算器

2.1 数据与文字的表示方法

2.2 定点加法、减法运算

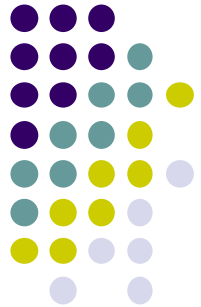
2.3 定点乘法运算

2.4 定点除法运算

2.5 定点运算器的组成

2.6 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

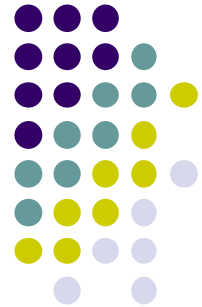
2.1.2 数的机器码表示

2.1.3 字符与字符串的表示方法

2.1.4 汉字的表示方法

2.1.5 校验码

# 2.1 数据与文字的表示方法



- 计算机中使用的数据可分成两大类：
  - 符号数据:非数字符号的表示（ASCII、汉字、图形等）
  - 数值数据:数字数据的表示方式（定点、浮点）
- 计算机数字和字符的表示方法应有利于数据的存储、加工(处理)、传送；
- 编码：用少量、简单的基本符号，选择合适的规则表示尽量多的信息，同时利于信息处理（速度、方便）

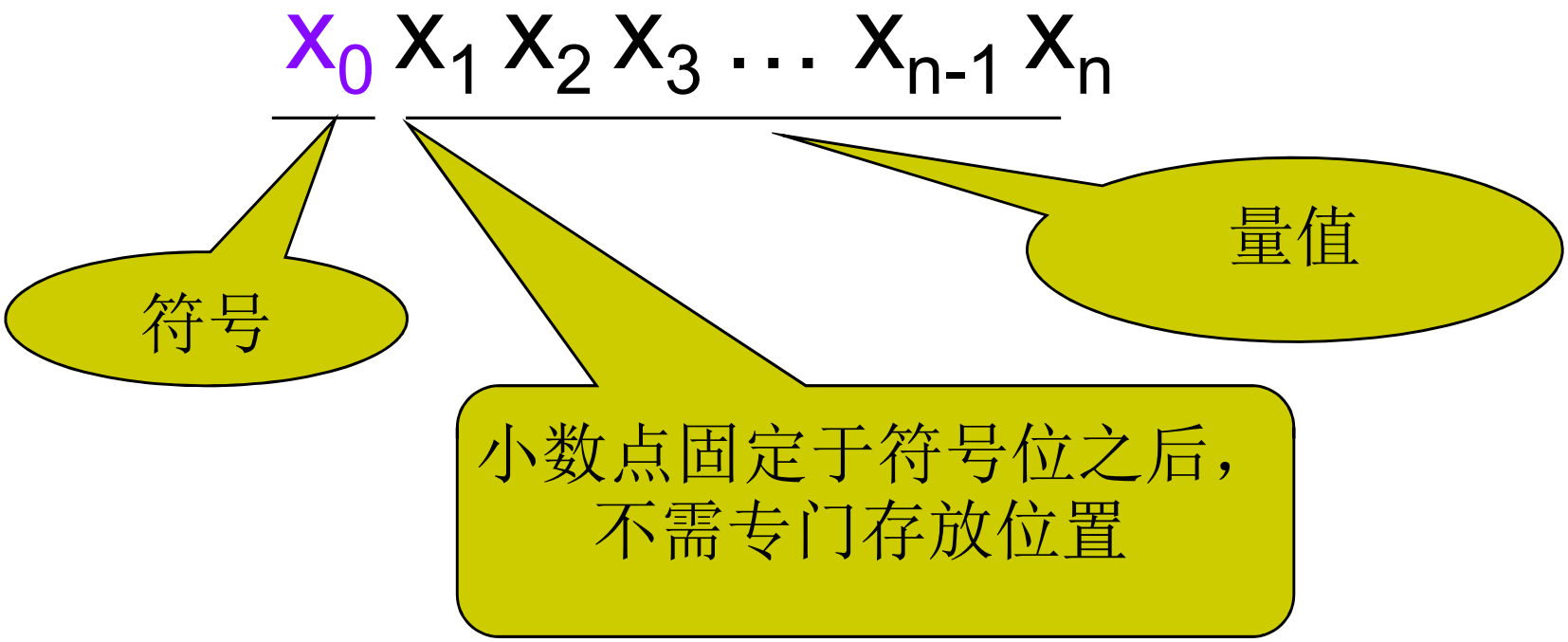
# 2.1.1 数据格式

一、定点表示法

- 所有数据的小数点位置固定不变
- 理论上位置可以任意，但实际上将数据表示有两种方法（小数点位置固定-定点表示法/定点格式）：
  - 纯小数
  - 纯整数

# 2.1.1 数据格式

## 1、定点纯小数




表示数的范围是 $-1 < X < +1$

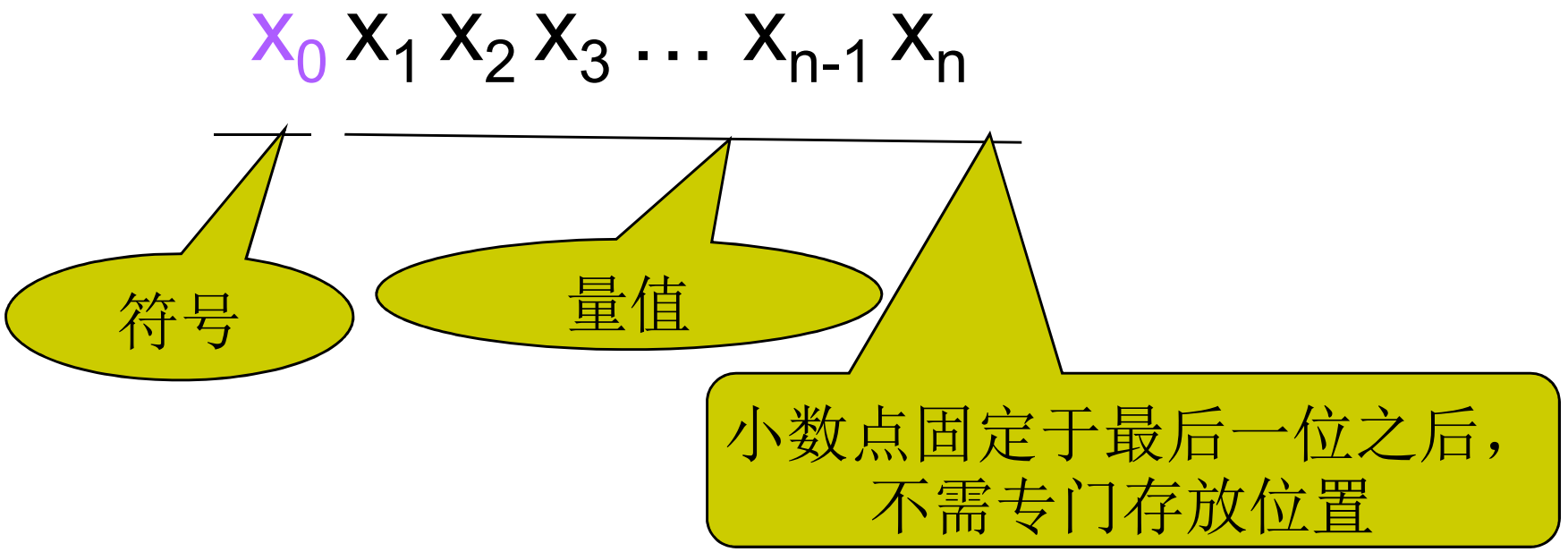
# 2.1.1 数据格式

## 2、纯小数的表示范围

| | | |
|---|---|---|
| x=0.00...0<br>x=1.00...0 | x=0 | 正0和负0都是0 |
| x=0.11...1 | $x=1-2^{-n}$ | 最大正数 |
| x=0.00...01 | $x=2^{-n}$ | 最接近0的正数 |
| x=1.00...01 | $x=-2^{-n}$ | 最接近0的负数 |
| x=1.11...1 | $x=-(1-2^{-n})$ | 最小负数 |

# 2.1.1 数据格式

3、定点纯整数

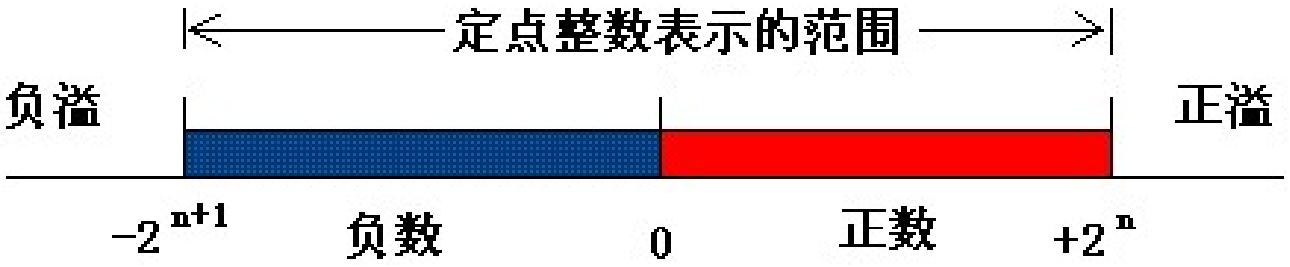



表示数的范围是 $-(2^n-1) \le X \le +(2^n-1)$

# 2.1.1 数据格式

## 4、定点表示法的特点

- 定点数表示数的范围受字长限制，表示数的范围有限;
- 定点表示的精度有限
- 机器中，常用定点纯整数表示;



# 2.1.1 数据格式

2、浮点表示法

电子质量(克)： $9\times10^{-28} = 0.9\times10^{-27}$

太阳质量(克)： $2\times10^{33} = 0.2\times10^{34}$

# 2.1.1 数据格式

2、浮点表示：小数点位置随阶码不同而浮动

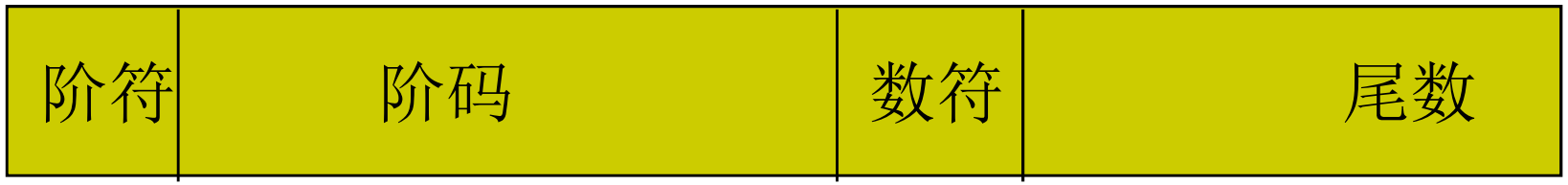
1、格式:$N=R^{E}.M$

基数R,取固定的值，比如10或2，隐含表示

指数E

尾数M

2、机器中表示



# 2.1.1 数据格式

浮点数的规格化表示：

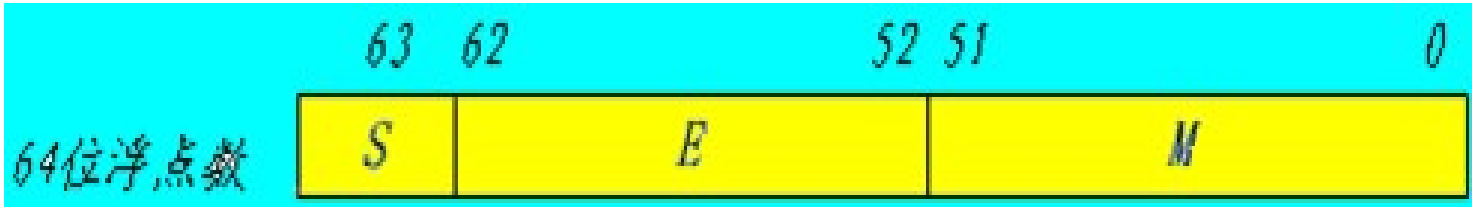
$(1.75)_{10} = 1.11 \times 2^0$ (规格化表示)

$= 0.111 \times 2^1 = 0.0111 \times 2^2$

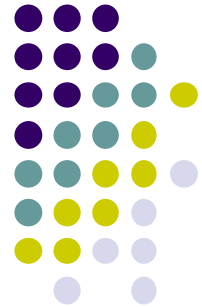
# 2.1.1 数据格式

3、IEEE754标准(规定了浮点数的表示格式,运算规则等)

- 规则规定了单精度(32)和双精度(64)的基本格式.
- 规则中,尾数用原码,指数用移码(便于对阶和比较)

# 2.1.1 数据格式



IEEE754标准

- 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。
  - 规格化： 若不对浮点数的表示作出明确规定，同一个浮点数的表示就不是惟一的。
    - 尾数域最左位(最高有效位)总是1， 故这一位经常不予存储，而认为隐藏在小数点的左边。
    - 采用这种方式时，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。

# 2.1.1 数据格式

- 64位的浮点数中符号位1位，阶码域11位，尾数域52位，指数偏移值是1023。因此规格化的64位浮点数x的真值为：

  $x=(-1)^S\times(1.M)\times2^{E-1023}$

  $e=E-1023$

- 一个规格化的32位浮点数x的真值表示为

  $x=(-1)^S\times(1.M)\times2^{E-127}$

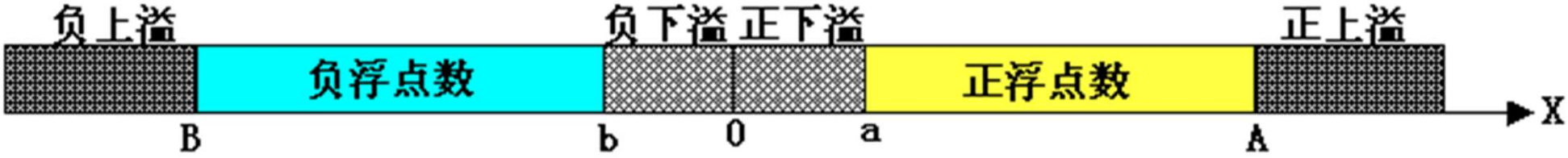
  $e=E-127$

# 2.1.1 数据格式

- 真值x为零表示：当阶码E为全0且尾数M也为全0时的值，结合符号位S为0或1，有正零和负零之分。
- 真值x为无穷大表示：当阶码E为全1且尾数M为全0时，结合符号位S为0或1，也有+∞和-∞之分。
- 这样在32位浮点数表示中，要除去E用全0和全1（$255_{10}$）表示零和无穷大的特殊情况，指数的偏移值不选128（10000000），而选127（01111111）。对于规格化浮点数，E的范围变为1到254，真正的指数值e则为-126到+127。因此32位浮点数表示的绝对值的范围是$10^{-38}$～$10^{38}$（以10的幂表示）。
- 浮点数所表示的范围远比定点数大。一台计算机中究竟采用定点表示还是浮点表示，要根据计算机的使用条件来确定。一般在高档微机以上的计算机中同时采用定点、浮点表示，由使用者进行选择。而单片机中多采用定点表示。

# 2.1.1 数据格式

浮点数表示范围如下图所示




A—最大正数　　a—最小正数
B—最小负数　　b—最大负数

# 2.1.1 数据格式

例1：若浮点数x的754标准存储格式为$(41360000)_{16}$，求其浮点数的十进制数值。

解：将16进制数展开后，可得二制数格式为

0　100 00010　011 0110 0000 0000 0000 0000

符号S　阶码E(8位)　尾数M(23位)

指数e=E-127=10000010-01111111=00000011=$(3)_{10}$

包括隐藏位1的尾数

1.M=1.011 0110 0000 0000 0000 0000=1.011011

于是有

$x=(-1)^S\times1.M\times2^e=+(1.011011)\times2^3=+1011.011=(11.375)_{10}$

# 2.1.1 数据格式

例2：将数$(20.59375)_{10}$转换成754标准的32位浮点数的二进制存储格式。

解：首先分别将整数和分数部分转换成二进制数：

20.59375=10100.10011

然后移动小数点，使其在第1，2位之间

$10100.10011=1.010010011\times 2^4$

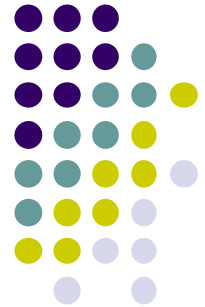
e=4于是得到：

S=0, E=4+127=131, M=010010011

最后得到32位浮点数的二进制存储格式为：

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

# 2.1.2 数的机器码表示



- 真值：一般书写的数
- 机器码：机器中表示的数，要解决在计算机内部数的正、负符号和小数点运算问题。
  - 原码
  - 反码
  - 补码
  - 移码

# 1、原码表示法

- 定点小数$x_0.\ x_1x_2\cdots x_n$

$$[x]_{原}=\begin{cases} x & 1>x\geq 0 \\ 1-x & 0\geq x>-1 \end{cases} \qquad 符号\begin{cases} 0，正 \\ 1，负数 \end{cases}$$

  - 有正0和负0之分
  - 范围$-(1-2^{-n})$ ～ $+(1-2^{-n})$

例：x=+0.11001110 ， y=-0.11001110

$[x]_{原}$=0.11001110　　$[y]_{原}$=1.11001110

# 1、原码表示法

- 定点整数$X_0X_1X_2\cdots X_n$

$$[x]_{原}=\begin{cases} x & 2^n>x\geq 0 \\ 2^n-x & 0\geq x>-2^n \end{cases} \quad 符号 \begin{cases} 0，正数 \\ 1，负数 \end{cases}$$

说明：

  - 有正0和负0之分
  - 范围 $-(2^n-1)\sim+(2^n-1)$

- 例：x=+11001110，y=-11001110

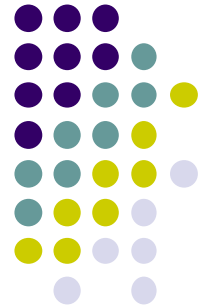
  $[x]_{原}$=011001110　　$[y]_{原}$=111001110

# 1、原码表示法

原码特点：

- 表示简单，易于同真值之间进行转换，实现乘除运算规则简单。
- 进行加减运算十分麻烦。

# 2.1.2 数的机器码表示



2、补码表示法

- 生活例子：现为北京时间下午4点，但钟表显示为7点。有两种办法校对：
  (1) 做减法 7-3 = 4 (逆时针退3格)
  (2) 做加法 7+9 = 16 (顺时针进9格)
  16 (mod 12) = 16-12 = 4 (以12为模，变成4)

# 2、补码表示法

- 定义：正数的补码就是正数的本身，负数的补码是原负数加上模。
- 计算机运算受字长限制,属于有模运算.
  - 定点小数$x_0.x_1x_2\ldots x_n$，以2为模
  - 定点整数$x_0.x_1x_2\ldots x_n$，以$2^{n+1}$为模
- 定点小数$x_0.x_1x_2\ldots x_n$

$$[x]_{补}=\begin{cases} x & 1>x\geq 0 \\ 2+x & 0\geq x>-1 \end{cases} \quad 符号 \begin{cases} 0，正小数 \\ 1，负小数 \end{cases}$$

# 2、补码表示法

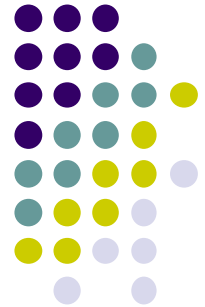
- 定点整数$x_0.x_1x_2\ldots x_n$

$$[x]_{补}=\begin{cases} x & 2^n>x\geq 0 \\ 2^{n+1}+x & 0\geq x>-2^n \end{cases} \qquad 符号\begin{cases} 0，正整数 \\ 1，负整数 \end{cases}$$

补码最大的优点就是将减法运算转换成加法运算。

通常不按表达式求补码，而通过反码来得到。

# 3、反码表示法



- 定义：正数的表示与原、补码相同，负数的补码符号位为1，数值位是将原码的数值按位取反，就得到该数的反码表示。
- 电路容易实现，触发器的输出有正负之分。

# 3、反码表示法

- 对尾数求反，它跟补码的区别在于末位少加一个1，所以可以推出反码的定义
  - 定点小数x0.x1x2…xn

$$[x]_{反}=\begin{cases} x & 1>x\geq 0 \\ 2+x-2^{-n} & 0\geq x>-1 \end{cases}$$

X1=+0.1011011 , [X1] 反 =0.1011011

X2= -0.1011011 , [X2] 反 =1.0100100

$$\begin{array}{r} 1.\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 0.\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ \hline 1.\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \end{array}$$

# 3、反码表示法

- $[x]_{补}=[x]_{反}+2^{-n}$（证明见书）
- 反码表示有正0和负0之分

# 4、移码表示法

- 移码表示法（用在阶码中）
  - 定点整数定义　$[x]_{移}=2^n+x$　$2^n > x \geq -2^n$
  - 00000000~11111111($-2^n$~$2^n-1$)
  - 例1 x=+1011111

原码为01011111

补码为01011111

反码为01011111

移码为11011111

# 4、移码表示法

例2：x=-1011111

原码为11011111

补码为10100001

反码为10100000

移码为00100001

特点：移码和补码尾数相同，符号位相反

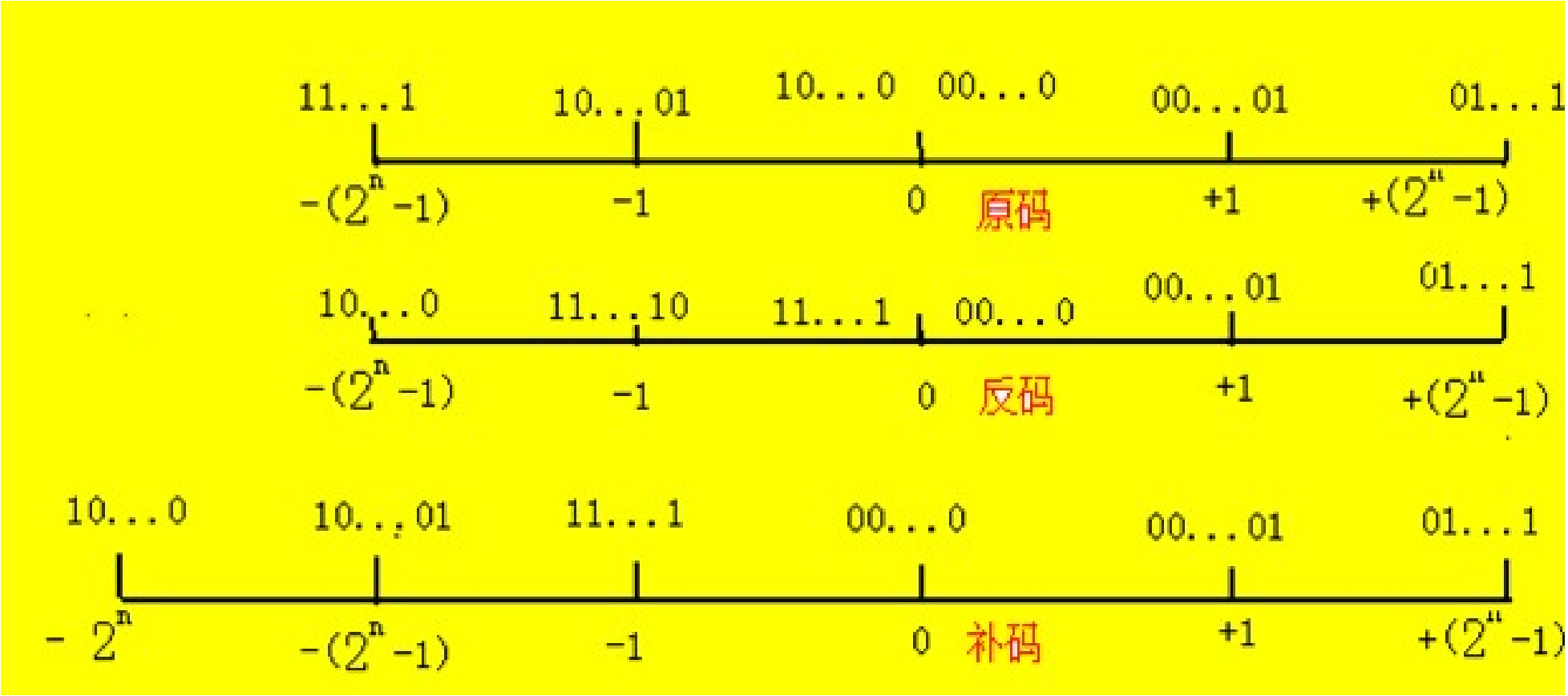
范围:$-2^n$~$2^n-1$

浮点IEEE754表示e=-127~+128

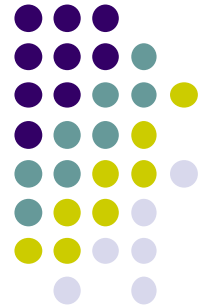
00000000阶码表示数字”0”,尾数的隐含位为0

11111111阶码表示数字”无穷大” ,尾数的隐含位为0

**[例6]**以定点整数为例,用数轴形式说明原码、反码、补码表示范围和可能的数码组合情况。



**[例7]**将十进制真值(－**127**,－**1**,**0**,＋**1**,＋**127**)列表表示成二进制数及原码、反码、补码、移码值。



| 真值x（十进制） | 真值x（二进制） | [x]原 | [x]反 | [x]补 | [x]移 |
|---|---|---|---|---|---|
| −127 | −01111111 | 11111111 | 10000000 | 10000001 | 00000001 |
| −1 | −00000001 | 10000001 | 11111110 | 11111111 | 01111111 |
| | | 00000000 | 00000000 | | |
| 0 | 00000000 | | | 00000000 | 10000000 |
| | | 10000000 | 11111111 | | |
| +1 | +00000001 | 00000001 | 00000001 | 00000001 | 10000001 |
| +127 | +01111111 | 01111111 | 01111111 | 01111111 | 11111111 |

# [例8]设机器字长16位,定点表示,尾数15位

(1)定点原码整数表示时,最大正数是多少?最小负数是多少?

0 111 111 111 111 111 最大正整数

$x=(2^{15}-1)_{10}=(+32767)_{10}$

1 111 111 111 111 111 最小负整数

$x=(1-2^{15})_{10}=-(2^{15}-1)_{10}=(-32767)_{10}$

(2)定点原码小数表示，最大正数是多少?最小负数是多少?

0 111 111 111 111 111 最大正小数

$x=(1-2^{15})_{10}$

1 111 111 111 111 111 最小负小数

$x=-(1-2^{15})10$

例9　假设由*S*,*E*,*M*三个域组成的一个32位二进制字所表示的非零规格化浮点数 $x$,真值表示为（注意此例非IEEE754标准）：

$x=(-1)^{s}\times(1.M)\times 2^{E-128}$

问：它所表示的规格化的最大正数、最小正数、最大负数、最小负数是多少？

(1)最大正数

0 11 111 111 111 111 111 111 111 111 111 111 11

$x=[1+(1-2^{-23})]\times 2^{127}$

(2)最小正数

0 00 000 000 000 000 000 000 000 000 000 00

$x=1.0\times 2^{-128}$

(3)最小负数

1 11 111 111 111 111 111 111 111 111 111 11

$x=-[1+(1-2^{-23})]\times 2^{127}$

(4)最大负数

1 00 000 000 000 000 000 000 000 000 000 00

$x=-1.0\times 2^{-128}$

$N=R^{E}.M$

# 2.1.3 字符与字符串的表示方法

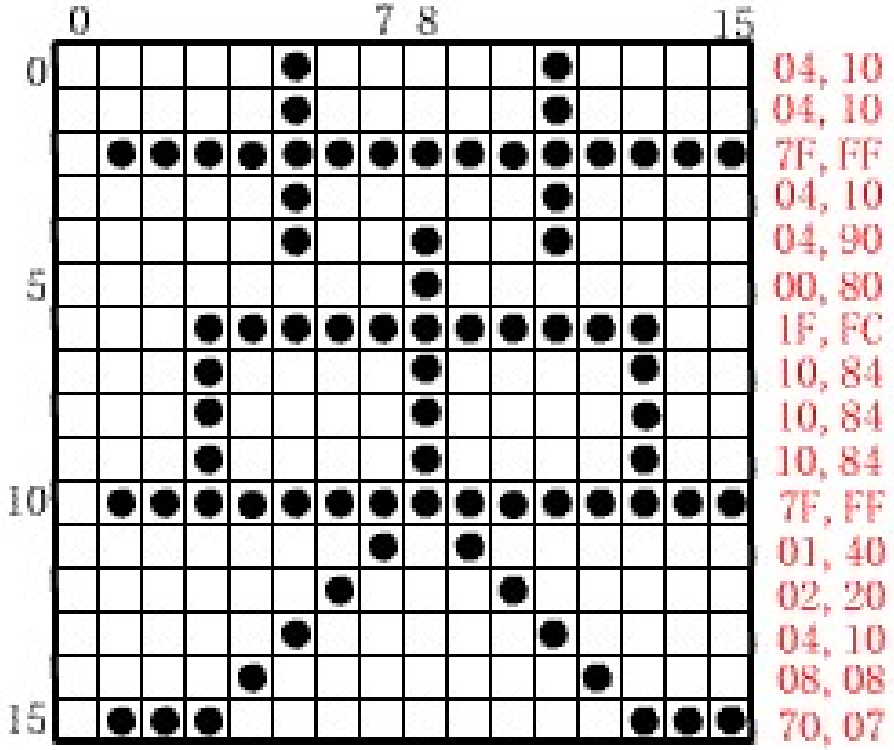
- 符号数据：字符信息用数据表示，如ASCII等；
- 字符表示方法ASCII:用一个字节来表示,低7位用来编码(128),最高位为校验位,参见教材P24表2.1
- 字符串的存放方法

# 2.1.4 汉字的表示方法

- 汉字的表示方法
（一级汉字3755个，二级汉字3008个）
  - 输入码
    - 国标码
      - 一级（16~55）*94
      - 二级（56~87）*94
      - 图形符号（682个）（01~09）*94
    - 拼音、五笔
  - 汉字内码：汉字信息的存储，交换和检索的机内代码，两个字节组成，每个字节高位都为1（区别于英文字符）

# 2.1.4 汉字的表示方法

- 汉字字模码：汉字字形
  - 点阵

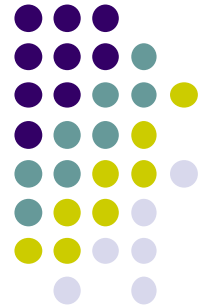



  - 汉字库

# 2.1.5 校验码

- 校验码（只介绍奇偶校验码）
  - 引入：信息传输和处理过程中受到干扰和故障，容易出错。
  - 解决方法：是在有效信息中加入一些冗余信息（校验位）
  - 奇偶校验位定义
  - 设 $x=(x_0 x_1 \ldots x_{n-1})$是一个n位字,则奇校验位 C 定义为:$\overline{C}=x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$,式中⊕代表按位加,表明只有当 x 中包含有奇数个1时,才使C＝1,即$\overline{C}=0$。同理可以定义偶校验。
  - 只能检查出奇数位错；不能纠正错误。
  - p27例2.11自己看一下。
  - 其它还有Hamming,CRC

# 2.2 定点加法、减法运算



2.2.1 补码加法

2.2.2 补码减法

2.2.3 溢出概念与检测方法

2.2.4 基本的二进制加法/减法器

# 2.2.1 补码加法

- 补码加法

  公式：$[x+y]_{补}=[x]_{补}+[y]_{补} \pmod{2^{n+1}}$

# $[x]_{补}+[y]_{补}=[x+y]_{补}$证明

- 假设 $|x|<1$, $|y|<1$, $|x+y|<1$
- 现分四种情况来证明

(1) $x>0$, $y>0$,则 $x+y>0$

$[x]_{补}=x$, $[y]_{补}=y$, $[x+y]_{补}=x+y$

所以等式成立.

(2) $x>0$, $y<0$,则 $x+y>0$或 $x+y<0$

$[x]_{补}=x$, $[y]_{补}=2+y$,

$[x]_{补}+[y]_{补}=x+2+y$

当 $x+y>0$时,$2+(x+y)>2$,进位2必丢失,又因$(x+y)>0$,

故 $[x]_{补}+[y]_{补}=x+y=[x+y]_{补}$

当 $x+y<0$时,$2+(x+y)<2$,又因$(x+y)<0$,

故 $[x]_{补}+[y]_{补}=2+(x+y)=[x+y]_{补}$

所以上式成立

# $[x]_{补}+[y]_{补}=[x+y]_{补}$证明

(3) $x<0, y>0$,则 $x+y>0$或 $x+y<0$

这种情况和第2种情况一样,把 $x$ 和 $y$ 的位置对调即得证。

(4) $x<0, y<0$,则 $x+y<0$

相加两数都是负数,则其和也一定是负数。

$\because [x]_{补}=2+x, \quad [y]_{补}=2+y$

$\therefore [x]_{补}+[y]_{补}=2+x+2+y=2+(2+x+y)$

上式右边分为”2”和$(2+x+y)$两部分.既然$(x+y)$是负数,而其绝对值又小于1,那么$(2+x+y)$就一定是小于2而大于1的数,进位”2”必丢失.又因$(x+y)<0$, 所以$[x]_{补}+[y]_{补}=2+(x+y)=[x+y]_{补}$

# 2.2.1 补码加法

- [例11] x=+1011 , y=+0101 , 求 x+y=?

  解：$[x]_{补}$ = 01001 , $[y]_{补}$ = 00101

| | | |
|---|---|---|
| | $[x]_{补}$ | 0 1 0 0 1 |
| + | $[y]_{补}$ | 0 0 1 0 1 |
| | $[x+y]_{补}$ | 0 1 1 1 0 |

∴ x+y = +1110

# 2.2.1 补码加法

[例12] x=+1011，y=-0101，求 x+y=？

解：$[x]_{补}$ = 01001，$[y]_{补}$ = 11011

| | | |
|---|---|---|
| | $[x]_{补}$ | 0 1 0 0 1 |
| ＋ | $[y]_{补}$ | 1 1 0 1 1 |
| | $[x+y]_{补}$ | 1 0 0 1 1 0 |

∴ x+y = +0110

# 2.2.2 补码减法

公式：

$$[x]_{补} - [y]_{补} = [x]_{补} + [-y]_{补}$$

$$[-y]_{补} = -[y]_{补} + 2^{-n}$$

# 2.2.2 补码减法

[例13] 已知$x_1$=-1110，$x_2$=+1101，求：

$[x_1]_{补}$，$[-x_1]_{补}$，$[x_2]_{补}$，$[-x_2]_{补}$。

解：

$[x_1]_{补}$ = 10010

$[-x_1]_{补}$ = -$[x_1]_{补}$ + $2^{-4}$ = 01101 + 00001 = 01110

$[x_2]_{补}$ = 01101

$[-x_2]_{补}$ = -$[x_2]_{补}$ + $2^{-4}$ = 10010 + 00001 = 10011

# 2.2.2 补码减法

[例14] x=+1101 , y=+0110 , 求 x-y=?

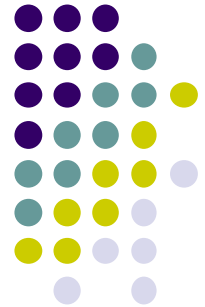
解：$[x]_{补}$ = 01101

$[y]_{补}$ = 00110 , $[-y]_{补}$ = 11010

| | | |
|---|---|---|
| | $[x]_{补}$ | 0 1 1 0 1 |
| ＋ | $[-y]_{补}$ | 1 1 0 1 0 |
| | $[x-y]_{补}$ | 1 0 0 1 1 1 |

∴ x-y = +0111

# 2.2.3 溢出概念与检测方法



- 溢出的概念
  - 可能产生溢出的情况
    - 两正数加，变负数，正溢（大于机器所能表示的最大数）
    - 两负数加，变正数，负溢（小于机器所能表示的最小数）

下面举两个例子。

# 2.2.3 溢出概念与检测方法

[例15] x=+1101，y=+1001，求 x+y 。

解：$[x]_{补}$=01011，$[y]_{补}$=01001

| | | |
|---|---|---|
| | $[x]_{补}$ | 0 1 0 1 1 |
| + | $[x]_{补}$ | 0 1 0 0 1 |
| | $[x+y]_{补}$ | 1 0 1 0 0 |

两个正数相加的结果成为负数，表示正溢。

# 2.2.3 溢出概念与检测方法

[例16] x=-1101 , y=-1011 , 求 x+y 。

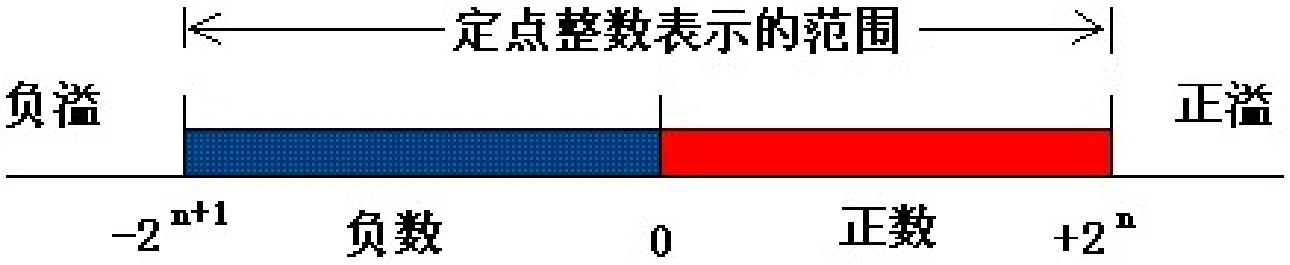
解：$[x]_{补}$=10011 , $[y]_{补}$=10101

| | | |
|---|---|---|
| | $[x]_{补}$ | 1 0 0 1 1 |
| ＋ | $[x]_{补}$ | 1 0 1 0 1 |
| | $[x+y]_{补}$ | 0 1 0 0 0 |

两个负数相加的结果成为正数，表示负溢。

# 2.2.3 溢出概念与检测方法

溢出的概念



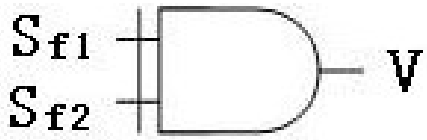
# 2.2.3 溢出概念与检测方法

检测方法

1、双符号位法（变形补码）

$[x]_{补} = 2^{n+2} + x \quad (\bmod 2^{n+2})$

| $S_{f1}$ | $S_{f2}$ | |
|---|---|---|
| 0 | 0 | 正确（正数） |
| 0 | 1 | 正溢 |
| 1 | 0 | 负溢 |
| 1 | 1 | 正确（负数） |




$S_{f1}$ 表示正确的符号，逻辑表达式为$V=S_{f1} \oplus S_{f2}$，可以用异或门来实现

# 2.2.3 溢出概念与检测方法

[例17] x=+01100 , y=+01000 , 求 x+y 。

解： $[x]_{补}$ = 001100 , $[y]_{补}$ = 001000

| | | |
|---|---|---|
| | $[x]_{补}$ | 0 0 1 1 0 0 |
| ＋ | $[y]_{补}$ | 0 0 1 0 0 0 |
| | $[x+y]_{补}$ | 0 1 0 1 0 0 (表示正溢) |

# 2.2.3 溢出概念与检测方法

[例18] x=-1100 , y=-1000 , 求 x+y 。

解： $[x]_{补}$ = 110100 , $[y]_{补}$ = 111000

| | | |
|---|---|---|
| | $[x]_{补}$ | 1 1 0 1 0 0 |
| ＋ | $[y]_{补}$ | 1 1 1 0 0 0 |
| | $[x+y]_{补}$ | 1 0 1 1 0 0 (表示负溢) |

# 2.2.3 溢出概念与检测方法

单符号位法

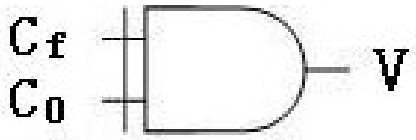
- $C_f$ $C_0$

  0 0 正确（正数）

  0 1 正溢

  1 0 负溢

  1 1 正确（负数）




- $V=C_f \oplus C_0$，其中$C_f$为符号位产生的进位,$C_0$为最高有效位产生

# 2.2.4 基本的二进制加法/减法器

## 一位全加器真值表

| 输入 | | | 输出 | |
|---|---|---|---|---|
| $A_i$ | $B_i$ | $C_i$ | $S_i$ | $C_{i+1}$ |
| 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 |

# 2.2.4 基本的二进制加法/减法器

- FA逻辑方程

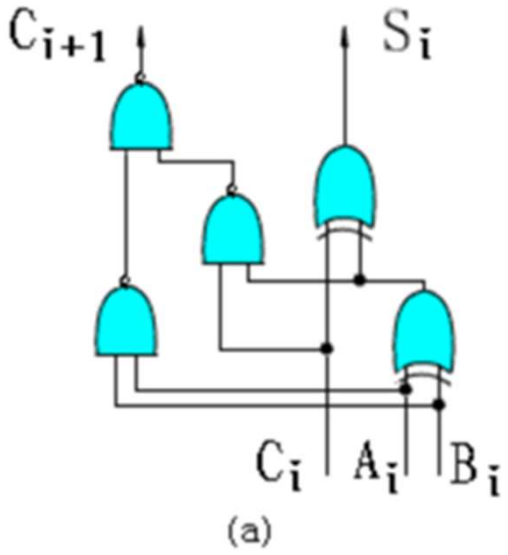
$$S_i = A_i \oplus B_i \oplus C_i$$

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

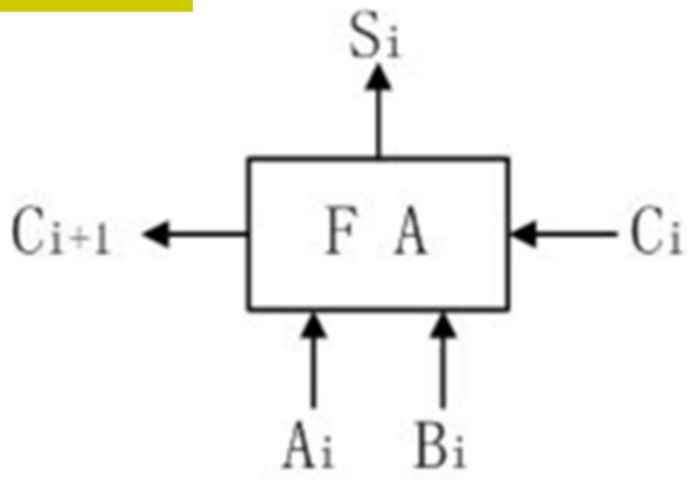
$$= A_iB_i + (A_i \oplus B_i)C_i$$

# 2.2.4 基本的二进制加法/减法器

FA逻辑电路和框图




FA（全加器）逻辑电路图




FA框图

# 2.2.4 基本的二进制加法/减法器

## n位行波进位加法器

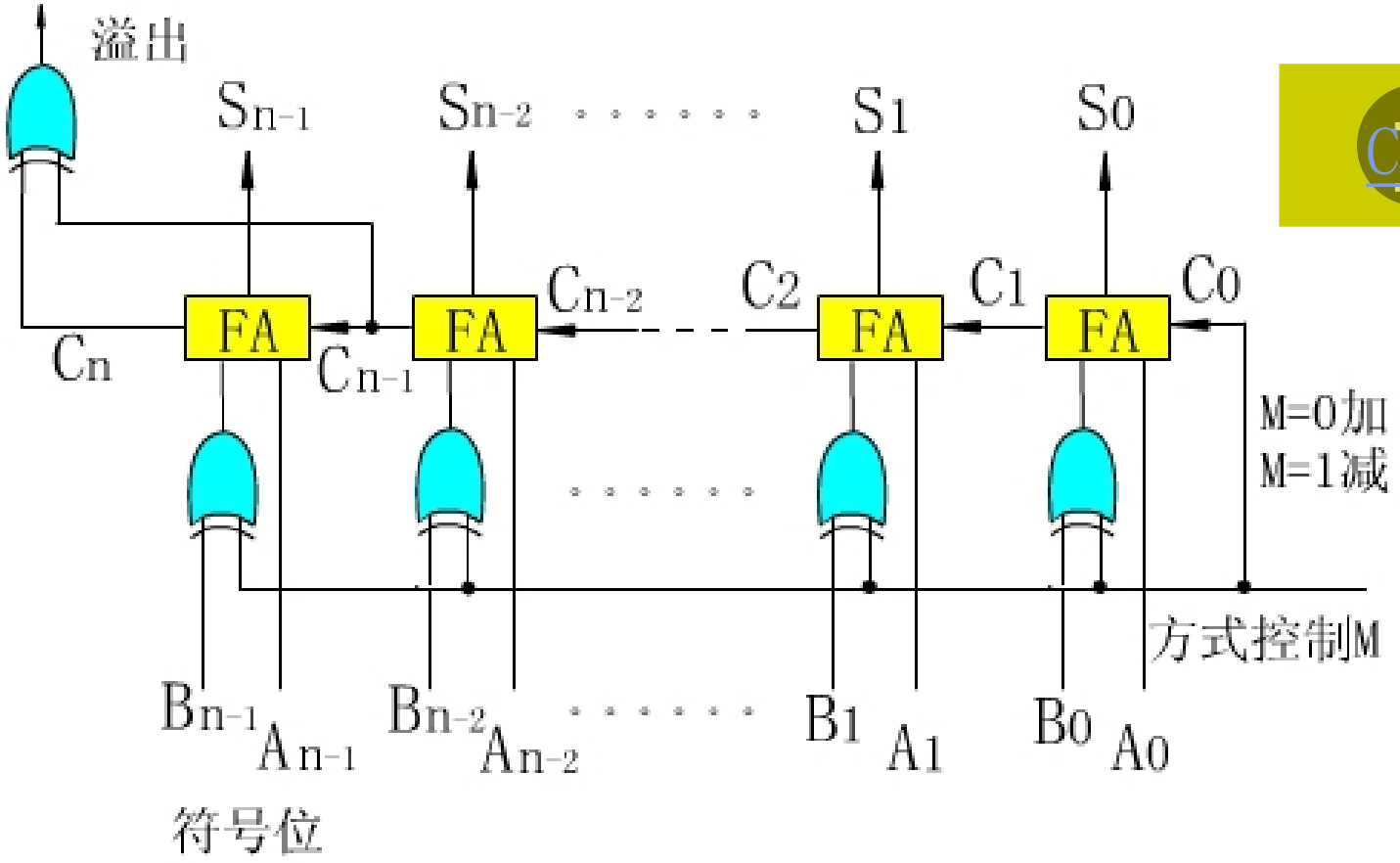
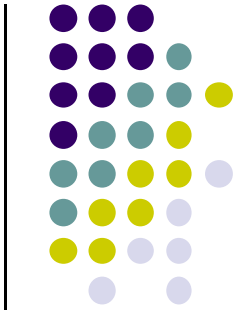


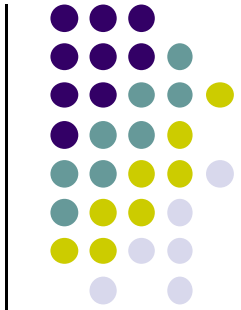

图2-3行波进位的补码加法/加法器



# 2.3 定点乘法运算

2.3.1 原码并行乘法

2.3.2 直接补码并行乘法



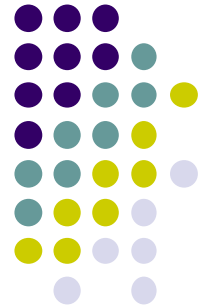
# 2.3.1 原码并行乘法

1、人工算法与机器算法的同异性

2、不带符号的阵列乘法器

3、带符号的阵列乘法器

# 1、人工算法与机器算法的同异性

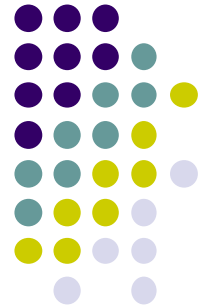


- $[x]_{原}=x_f.x_{n-1}\cdots x_1x0$　　$[y]_{原}=y_f.y_{n-1}\cdots y_1y_0$
- $[x.y]原=(x_f \oplus y_f)+(x_{n-1}\cdots x_1x_0).(y_{n-1}\cdots y_1y_0)$
- 用习惯方法求乘积如下：

  设 x ＝1101, y ＝1011

$$
\begin{array}{rcccccccc}
 & & & & 1 & 1 & 0 & 1 & (x) \\
\times & & & & 1 & 0 & 1 & 1 & (y) \\
\hline
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 0 & 0 & 0 & 0 & & & \\
+ & 1 & 1 & 0 & 1 & & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & (z)
\end{array}
$$

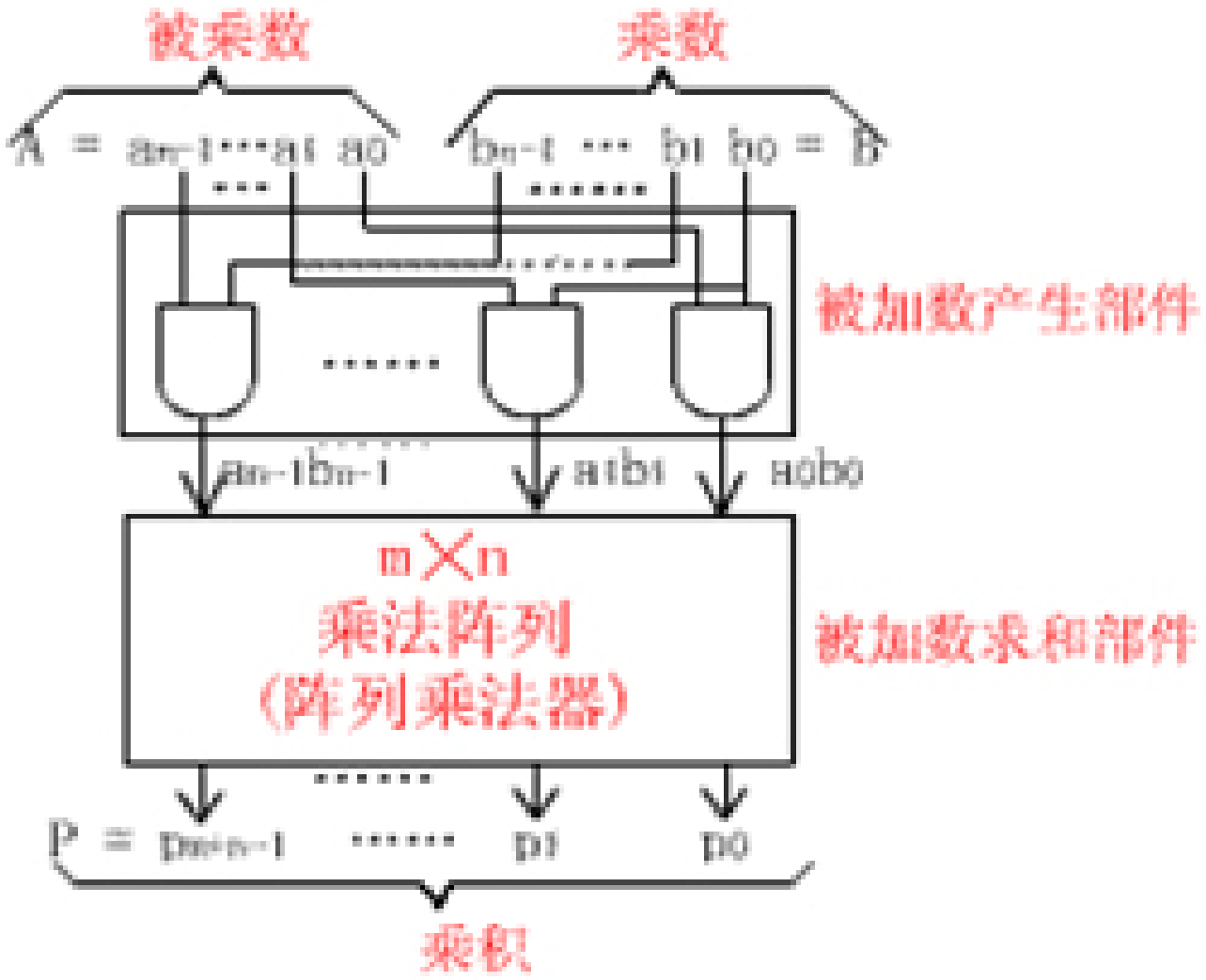
# 1、人工算法与机器算法的同异性



- n位乘n位积可能为2n位.
- 乘积的最后是所有部分积之和

采用流水式阵列乘法器，取代串行方案。

# 2、不带符号位的阵列乘法器




不带符号阵列乘法器逻辑图

# 2、不带符号位的阵列乘法器

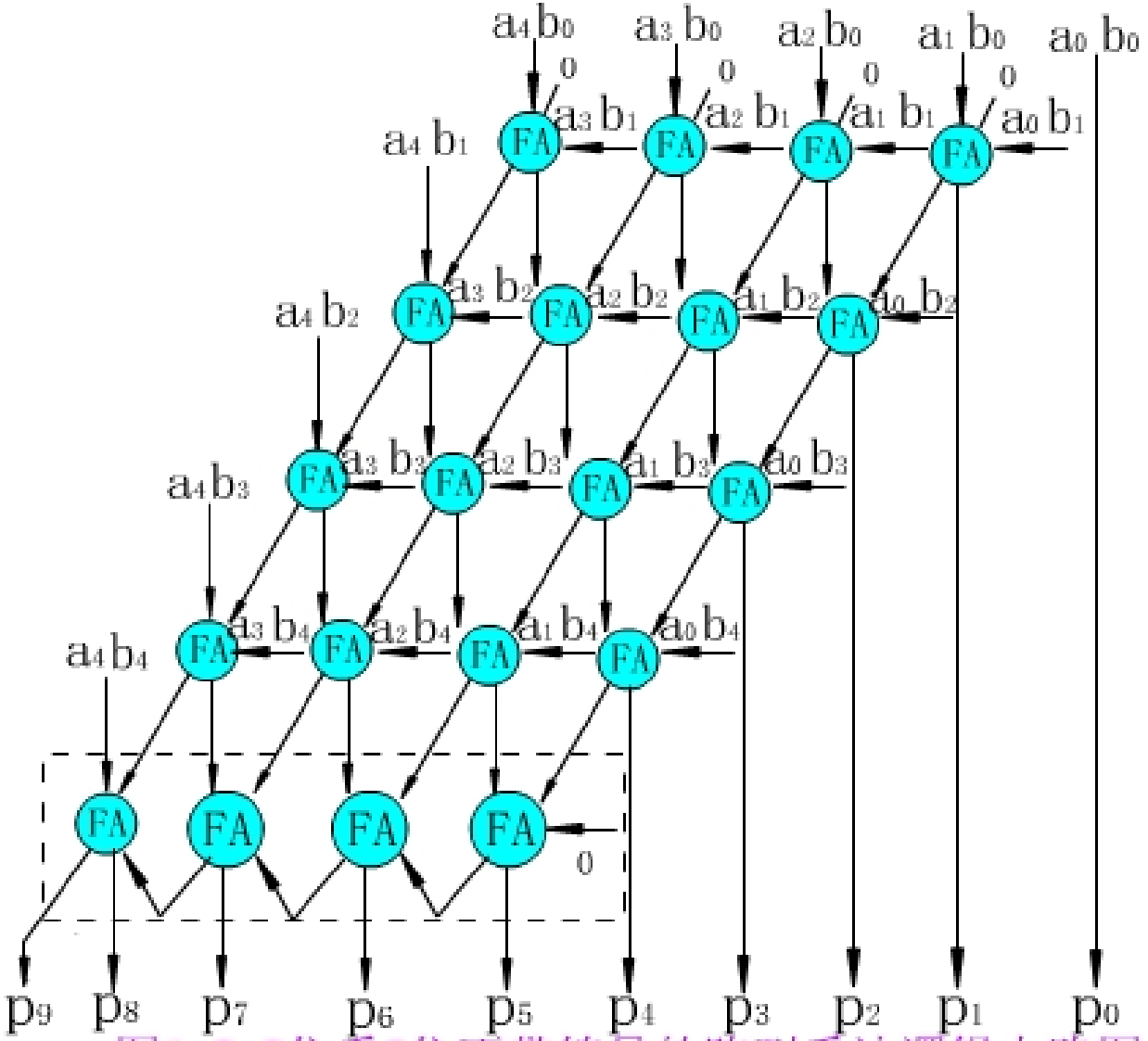
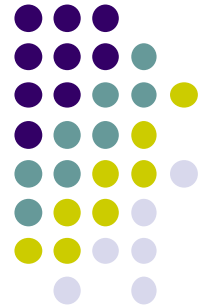



图2.5 5位乘5位不带符号的阵列乘法逻辑电路图

# 2、不带符号位的阵列乘法器



[例19] 参见图2.5，已知不带符号的二进制整数A=11011，B=10101，求每一部分乘积项$a_ib_j$的值与$p_9p_8\ldots p_0$的值。

解：

$$
\begin{array}{ll}
 & 1\ 1\ 0\ 1\ 1 = A\ (27_{10}) \\
\times & 1\ 0\ 1\ 0\ 1 = B\ (21_{10}) \\
\hline
 & 1\ 1\ 0\ 1\ 1 \quad a_4b_0=1,\ a_3b_0=1,\ a_2b_0=0,\ a_1b_0=1,\ a_0b_0=1 \\
 & 0\ 0\ 0\ 0\ 0 \quad a_4b_1=0,\ a_3b_1=0,\ a_2b_1=0,\ a_1b_1=0,\ a_0b_1=0 \\
 & 1\ 1\ 0\ 1\ 1 \quad a_4b_2=1,\ a_3b_2=1,\ a_2b_2=0,\ a_1b_2=1,\ a_0b_2=1 \\
 & 0\ 0\ 0\ 0\ 0 \quad a_4b_3=0,\ a_3b_3=0,\ a_2b_3=0,\ a_1b_3=0,\ a_0b_3=0 \\
+ & 1\ 1\ 0\ 1\ 1 \quad a_4b_4=1,\ a_3b_4=1,\ a_2b_4=0,\ a_1b_4=1,\ a_0b_4=1 \\
\hline
 & 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1 = P
\end{array}
$$

$P = p_9p_8p_7p_6p_5p_4p_3p_2p_1p_0 = 1000110111\ (567_{10})$

# 3、带符号位的阵列乘法器

- 求补电路

原理：算前求补—乘法器—算后求补，见下图

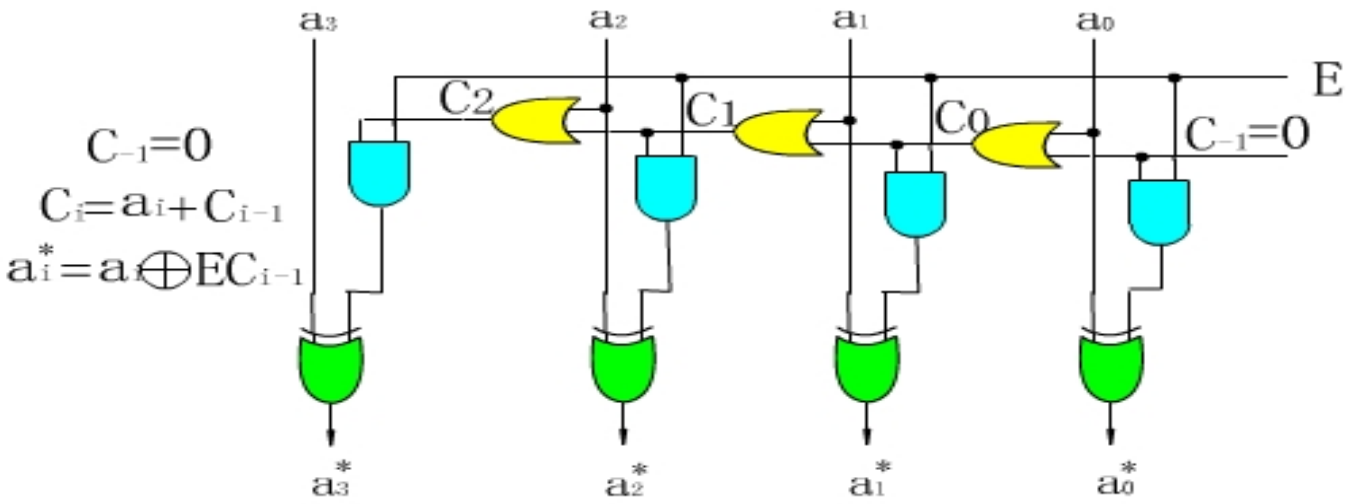



图2.6 对2求补器电路图

# 3、带符号的阵列乘法器

- 求补电路小结
- E=0时，输入和输出相等
- E=1时，则从数最右端往左边扫描，直到第一个1的时候，该位和右边各位保持不变，左边各数值位按位取反
- 可以用符号作为E 的输入
- 原：1.11110　　补：1.00010

不变，左边数值位取反

# 3、带符号的阵列乘法器

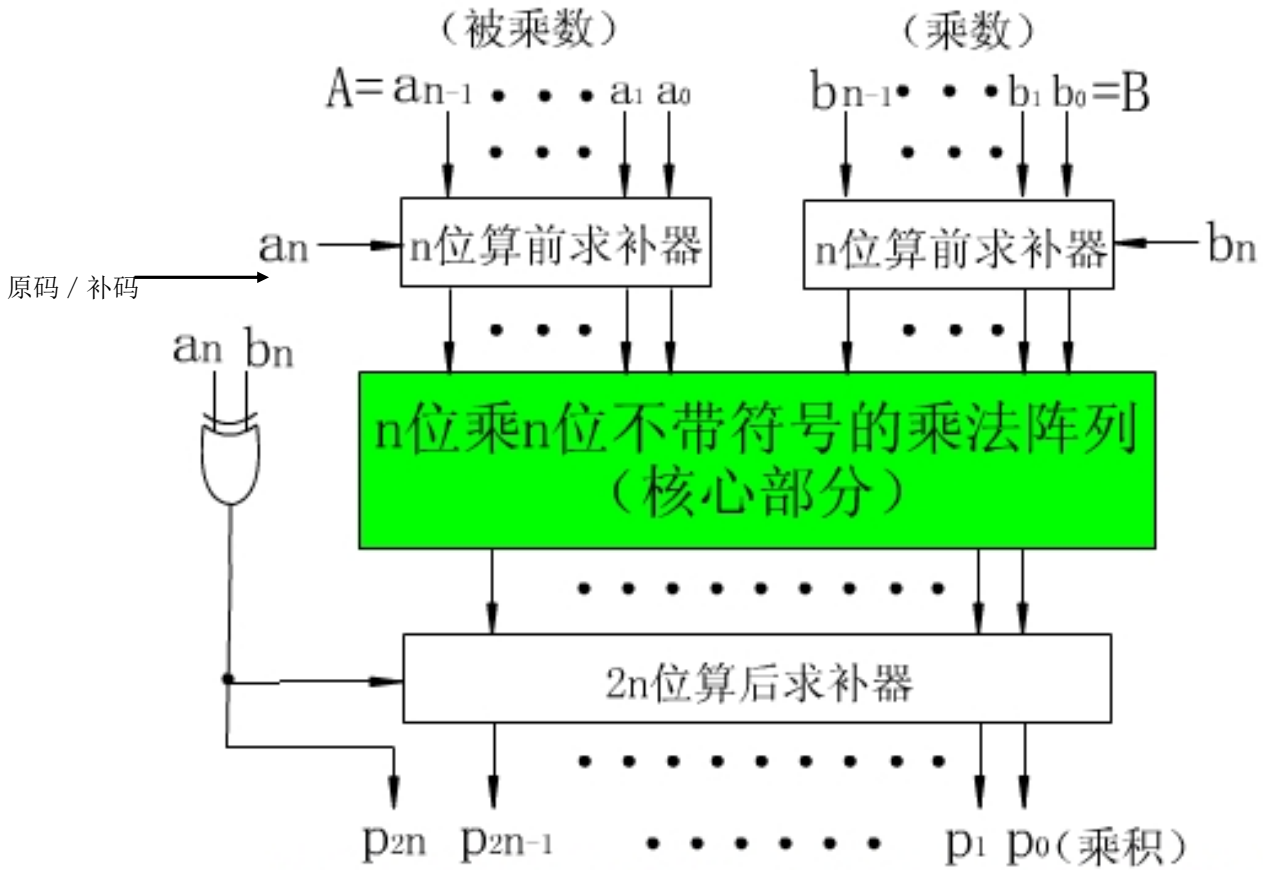



图2.7 （n+1）位乘（n+1）位带补级的阵列乘法器

[例20] 设x=+15，y=-13，用带求补器的原码阵列乘法器求出乘积 x·y=?

解：$[x]_{原}$=01111，$[y]_{原}$=11101，|x|=1111，|y|=1101

符号位运算：0⊕1=1

```
          1 1 1 1
×         1 1 0 1
-------------------
          1 1 1 1
        0 0 0 0
      1 1 1 1
+   1 1 1 1
-------------------
  1 1 0 0 0 0 1 1
```

乘积符号为1，算后求补器输出11000011，$[x\times y]_{原}$=111000011

换算成二进制数真值是 $x\cdot y = (-11000011)_2 = (-195)_{10}$

[例21] 设x=-15，y=-13，用带求补器的补码阵列乘法器求出乘积 x·y=? 并用十进制数乘法进行验证。

解：$[x]_{补}$=10001，$[y]_{补}$=10011，乘积符号位运算：1⊕1=0

尾数部分算前求补器输出 |x|=1111，|y|=1101

```
          1 1 1 1
×         1 1 0 1
-------------------------
          1 1 1 1
        0 0 0 0
      1 1 1 1
+   1 1 1 1
-------------------------
  1 1 0 0 0 0 1 1
```

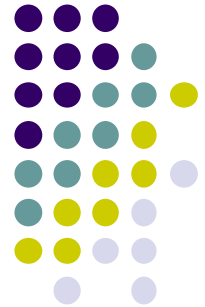
乘积符号为0，算后求补器输出11000011，$[x\times y]_{补}$=011000011

补码二进制数真值 $x\cdot y=0\times2^8+1\times2^7+1\times2^6+1\times2^1+1\times2^0$

$=(+195)_{10}$

十进制数乘法验证 x·y = (-15)×(-13) = +195

# 2.4 定点除法运算



2.4.1 原码除法算法原理
2.4.2 并行除法器

# 2.4.1 原码除法算法原理

设有n位定点小数（定点整数也适用）

被除数x，$[x]_{原}=x_f.x_{n-1}\cdots x_1x_0$

除数y，$[y]_{原}=y_f.y_{n-1}\cdots y_1y_0$

则有商q=x/y，

$[q]_{原}=(x_f\oplus y_f)+(0.x_{n-1}\cdots x_1x_0/0.y_{n-1}\cdots y_1y_0)$

商的符号运算$q_f=x_f\oplus y_f$与原码乘法一样，用模2求和得到。

下面是人工算法例子。

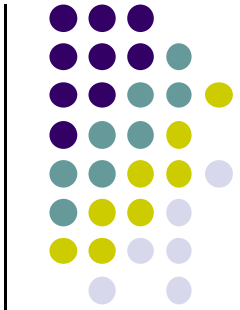
# 1、定点原码除法

定点原码一位除法实现方案（手工）0．10010／0．1011

| | | | |
|---|---|---|---|
| | 0.1101 | 商$q$ | |
| 0.1011 | 0.10010 | $x(r_0)$ | 被除数 |
| | −0.01011 | $2^{-1}y$ | 除数右移1位,减除数 |
| | 0.0011110 | $r_1$ | 得余数r1 |
| | −0.001011 | $2^{-2}y$ | 除数右移1位,减除数 |
| | 0.0000110 | $r_2$ | 得余数r2 |
| | −0.0001011 | $2^{-3}y$ | 除数右移1位,不减除数 |
| | 0.0000110 0 | $r_3$ | 得余数r3 |
| | −0.0000 1011 | $2^{-4}y$ | 除数右移1位,减除数 |
| | −0.0000 0001 | $r_4$ | 得余数r4 |

- 商0还是商1人可以比较后确定，计算机如何确定？

# 2、不恢复余数的除法



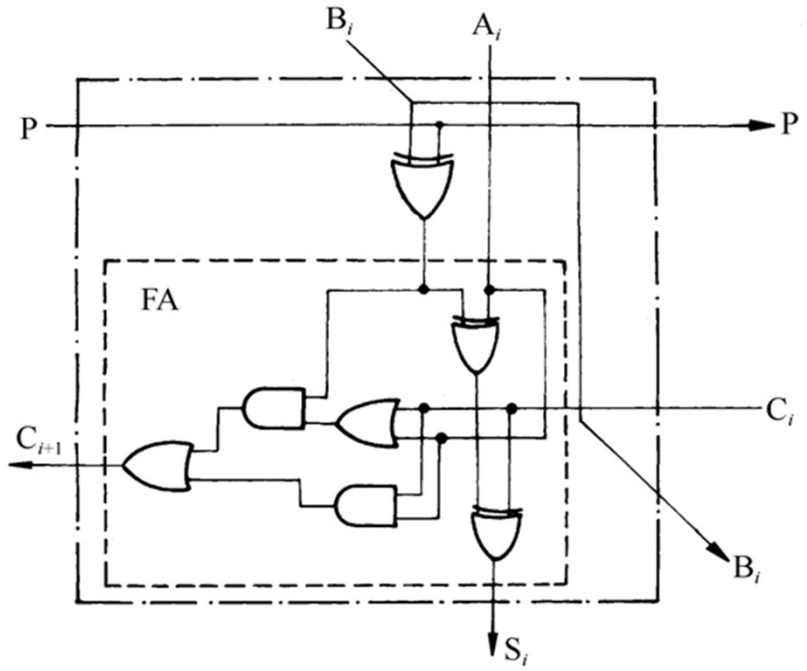
- 人工除法时，人可以比较被除数（余数）和除数的大小来确定商1（够减）或商0（不够减）
- 机器除法时，余数为正表示够减，余数为负表示不够减。不够减时必须恢复原来余数，才能继续向下运算。这种方法叫恢复余数法，控制比较复杂。
- 不恢复余数法（加减交替法）
  余数为正，商1，下次除数右移做减法；
  余数为负，商0，下次除数右移做加法。
  控制简单，有规律。

# 2.4.2 并行除法器

1、可控加法/减法(CAS)单元

原理：采用不恢复余数(加减交替)法

- P=0，作加法运算
- P=1，作减法运算



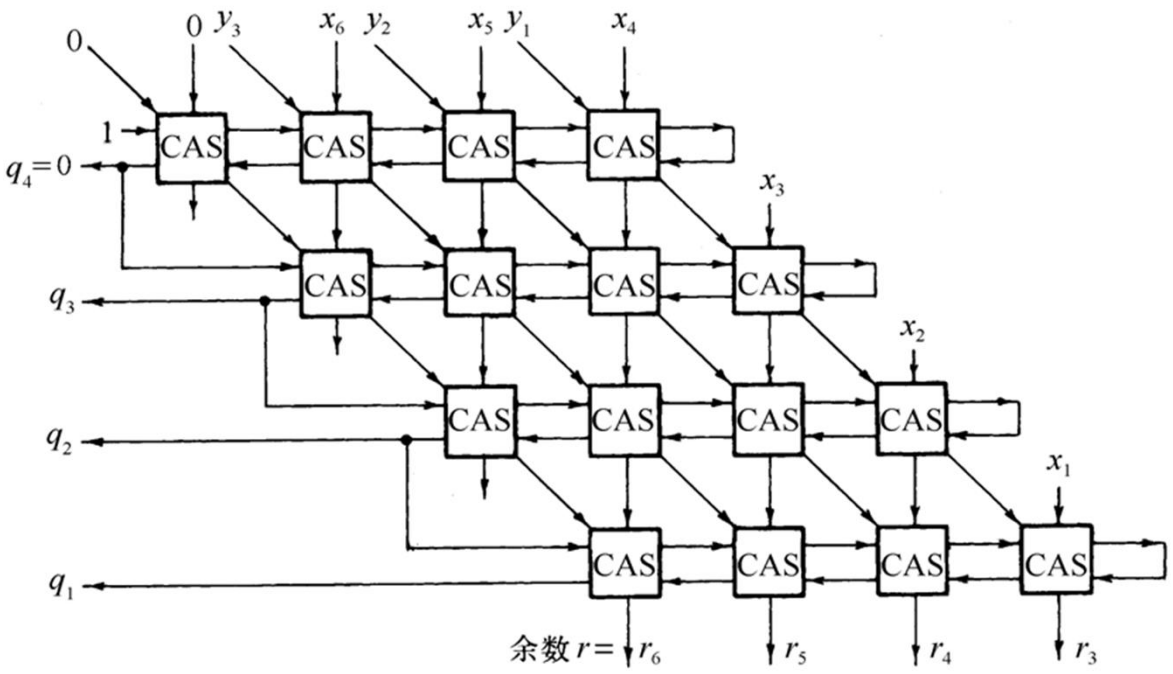

(a) 可控加法/减法(CAS)单元的逻辑图

# 2.4.2 并行除法器

## 2、不恢复余数的阵列除法器

被除数 $x=0.x_6x_5x_4x_3x_2x_1$
(双倍长)
除数　$y=0.y_3y_2y_1$
商数　$q=0.q_3q_2q_1$
余数　$r=0.00r_6r_5r_4r_3$

除数右移




(b) 4 位除 4 位阵列除法器

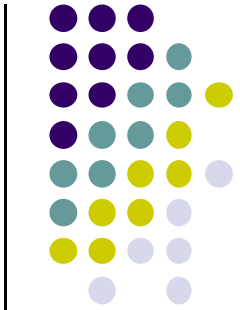
# 2.4.2 并行除法器

**[例23]** $x=0.101001$，$y=0.111$，求 $x \div y$。

**[解:] $[x]_{补}$=0.101001 , $[y]_{补}$=0.111 , $[-y]_{补}$=1.001**

| | | | |
|---|---|---|---|
| | 0.1 0 1 0 0 1 | | ;被除数 |
| ＋$[-y]_{补}$ | 1.0 0 1 | | ;第一步减除数y |
| ＋$[y]_{补}\rightarrow$ | 1.1 1 0 0 0 1<br>0.0 1 1 1 | <0 $q_4$=0 | ;余数为负,商0<br>;除数右移1位加 |
| ＋$[-y]_{补}\rightarrow$ | 0.0 0 1 1 0 1<br>1.1 1 0 0 1 | >0 $q_3$=1 | ;余数为正,商1<br>;除数右移2位减 |
| ＋$[y]_{补}\rightarrow$ | 1.1 1 1 1 1 1<br>0.0 0 0 1 1 1 | <0 $q_2$=0 | ;余数为负,商0<br>;除数右移3位加 |
| | 0.0 0 0 1 1 0 | >0 $q_1$=1 | ;余数为正,商1 |

商$q=q_4.q_3q_2q_1=0.101$，余数$r=(0.00r_6r_5r_4r_3)=0.000110$



# 2.5 定点运算器的组成

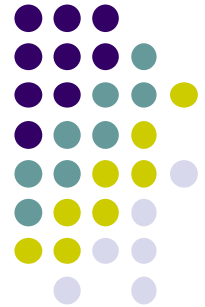
2.5.1 逻辑运算

2.5.2 多功能算术/逻辑运算单元ALU

2.5.3 内部总线

2.5.4 定点运算器的基本结构

# 2.5.1 逻辑运算



1、逻辑非运算

2、逻辑加运算

3、逻辑乘运算

4、逻辑异运算

# 1、逻辑非运算

$x = x_0x_1x_2\ldots x_n$，对x求逻辑非，则有

$$\overline{x} = z = z_0z_1z_2\ldots z_n$$

$$z_i = \overline{x_i},\ i=0,1,2,\ldots,n$$

# 1、逻辑非运算

[例24] $x_1$=01001001，$x_2$=11110000，求 $\overline{x}_1$，$\overline{x}_2$

解：

$\overline{x}_1$ = 10110100

$\overline{x}_2$ = 00001111

# 2、逻辑加运算

$x = x_0x_1x_2\cdots x_n$ , $y = y_0y_1y_2\cdots y_n$

则有

$x+y = z = z_0z_1z_2\cdots z_n$

$z_i = x_i+y_i$ , $i=0,1,2,\ldots,n$

# 2、逻辑加运算

[例25] x=10100001，y=100111011，求 x+y

解：

```
    1 0 1 0 0 0 0 1      x
+   1 0 0 1 1 0 1 1      y
-------------------------------
    1 0 1 1 1 0 1 1      z
```

即 x+y = 10111011

# 3、逻辑乘运算

$x = x_0x_1x_2\cdots x_n$ , $y = y_0y_1y_2\cdots y_n$

则有

$x \cdot y = z = z_0z_1z_2\cdots z_n$

$z_i = x_i \cdot y_i$ , $i=0,1,2,\ldots,n$

# 3、逻辑乘运算

[例26] x=10111001，y=11110011，求 x+y

解：

```
      1 0 1 1 1 0 0 1      x
·     1 1 1 1 0 0 1 1      y
-----------------------------
      1 0 1 1 0 0 0 1      z
```

即 x·y = 10110001

# 4、逻辑异运算

$x = x_0x_1x_2\cdots x_n$ , $y = y_0y_1y_2\cdots y_n$

则有

$x \oplus y = z = z_0z_1z_2\cdots z_n$

$z_i = x_i \oplus y_i$ , $i=0,1,2,\ldots,n$

# 4、逻辑异运算

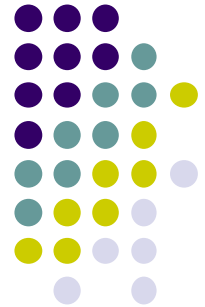
[例27] x=10101011，y=11001100，求 x+y

解：

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

即 $x \oplus y = 01100111$

# 2.5.2 多功能算术/逻辑运算单元ALU



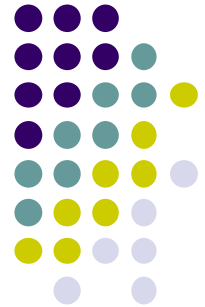
1、基本思想

2、逻辑表达式

3、算术逻辑运算的实现

4、两级先行进位的ALU

# 2.5.2 多功能算术/逻辑运算单元ALU



- 1、基本思想
  - 创新点：
    (1) 实现并行进位(先行进位)
    (2) 实现16种算术运算，16种逻辑运算

  - 基本思想：一位全加器FA的逻辑表达式：

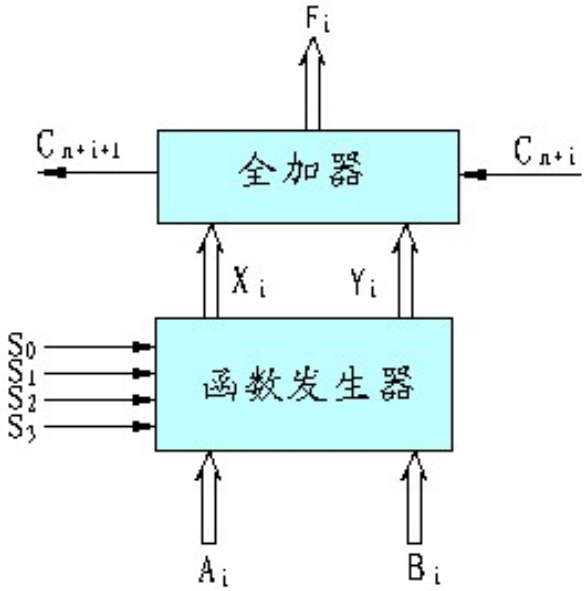
$$F_i = A_i \oplus B_i \oplus C_{n+i}$$

$$C_{n+i+1} = A_i B_i + A_i C_{n+i} + B_i C_{n+i}$$

  - 为了实现多种算术逻辑运算，可将Ai和Bi输入一个函数发生器（进位传递函数和进位产生函数）得到输出Xi和Yi，作为一位全加器的输入（见下页图）。

# 2.5.2 多功能算术/逻辑运算单元ALU

## ALU的逻辑图与逻辑表达式




一位ALU逻辑图

$$F_i = X_i \oplus Y_i \oplus C_{n+1}$$

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

4位一片，i=0,1,2,3

# 2.5.2 多功能算术/逻辑运算单元ALU

## 2、逻辑表达式

XiYi 与控制参数和输入量的关系构造如下真值表

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

$$Y_i = \overline{S_0}\,\overline{S_1}\,\overline{A_i} + \overline{S_0} S_1 \overline{A_i} B_i + S_0 \overline{S_1}\,\overline{A_i}\,\overline{B_i}$$

$$X_i = \overline{S_2}\,\overline{S_3} + \overline{S_2} S_3 (\overline{A_i} + \overline{B_i}) + \overline{S_2} S_3 (\overline{A_i} + B_i) + S_2 S_3 \overline{A_i}$$

# 2.5.2 多功能算术/逻辑运算单元ALU

- ALU的某一位逻辑表达式见下:

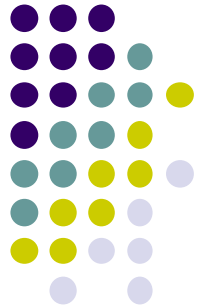
$$X_i = \overline{S_3 A_i B_i + S_2 A_i \overline{B}_i}$$

$$Y_i = \overline{A_i + S_0 B_i + S_1 \overline{B}_i}$$

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

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

# 2.5.2 多功能算术/逻辑运算单元ALU



- 如何实现先行进位？

答：由于每一位中X、Y的产生是同时的，则可以由下面方法算出并行进位的$C_{n+4}$

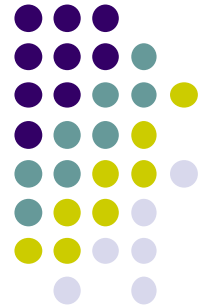
$$C_{n+1}=Y_0+X_0C_n$$

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

$$C_{n+3}=Y_2+X_2C_{n+2}=Y_2+Y_1X_1+Y_0X_1X_2+X_0X_1X_2C_n$$

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

# 2.5.2 多功能算术/逻辑运算单元ALU



- 令$G=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3$

  $P=X_0X_1X_2X_3$

- G为进位发生输出　　P为进位传送输出
- 增加P和G的目的在于实现多片（组）ALU之间的先行进位，需要配合电路，称为先行进位发生器（CLA）
- 器件： 74181

# 2.5.2 多功能算术/逻辑运算单元ALU

## 3、算术逻辑运算的实现

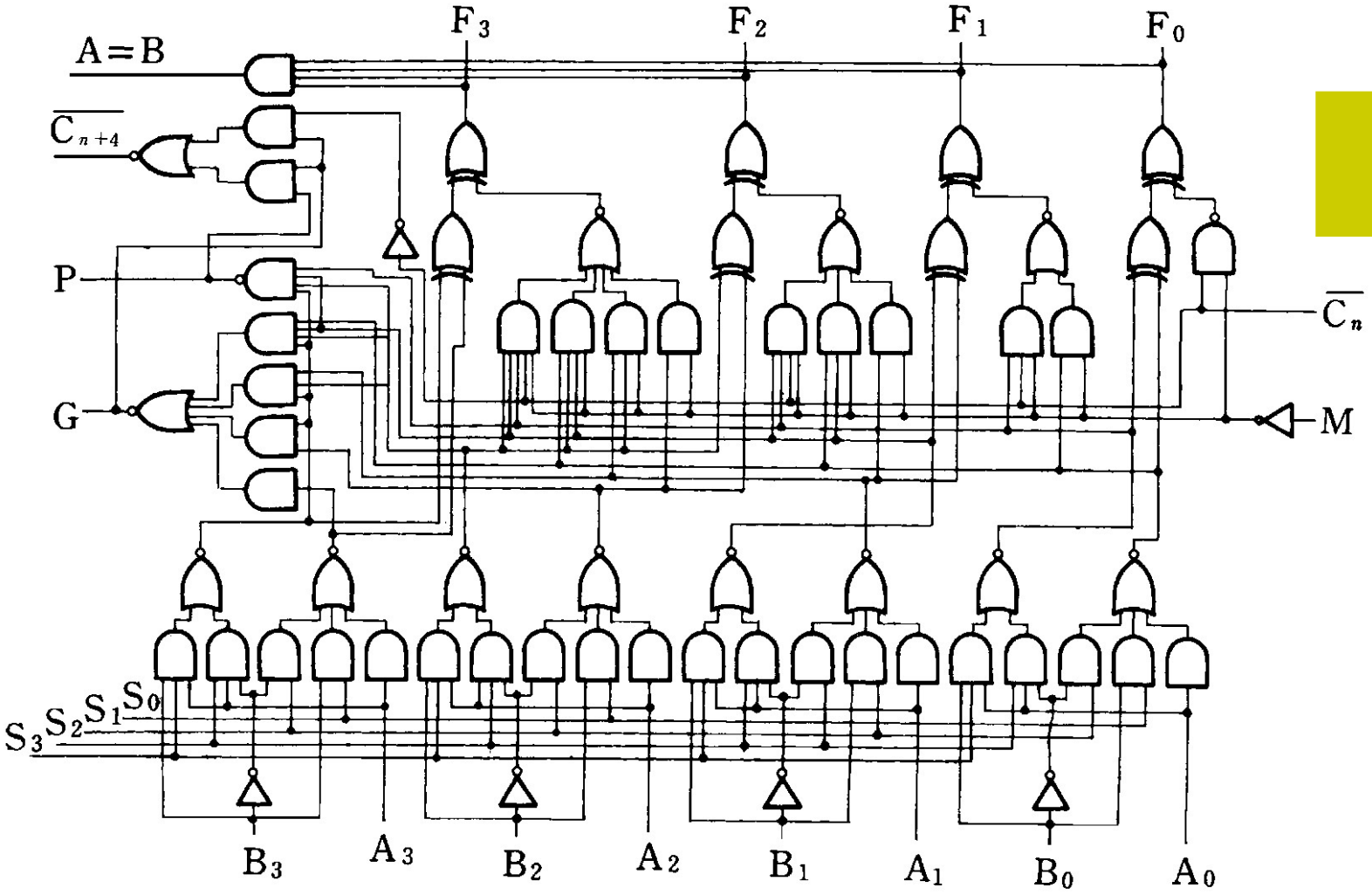
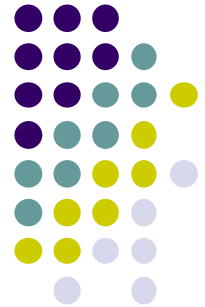


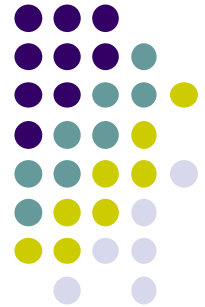

图2.11 正逻辑操作数表示的74181ALU逻辑电路图

# 2.5.2 多功能算术/逻辑运算单元ALU



- 算术逻辑运算的实现（74181）
  - M=L时，对进位信号没有影响，做算术运算
  - M=H时，进位门被封锁，做逻辑运算
- 说明：
  - 74181执行正逻辑输入/输出方式的一组算术运算和逻辑运算和负逻辑输入/输出方式的一组算术运算和逻辑运算是等效的。
  - A=B端可以判断两个数是否相等。

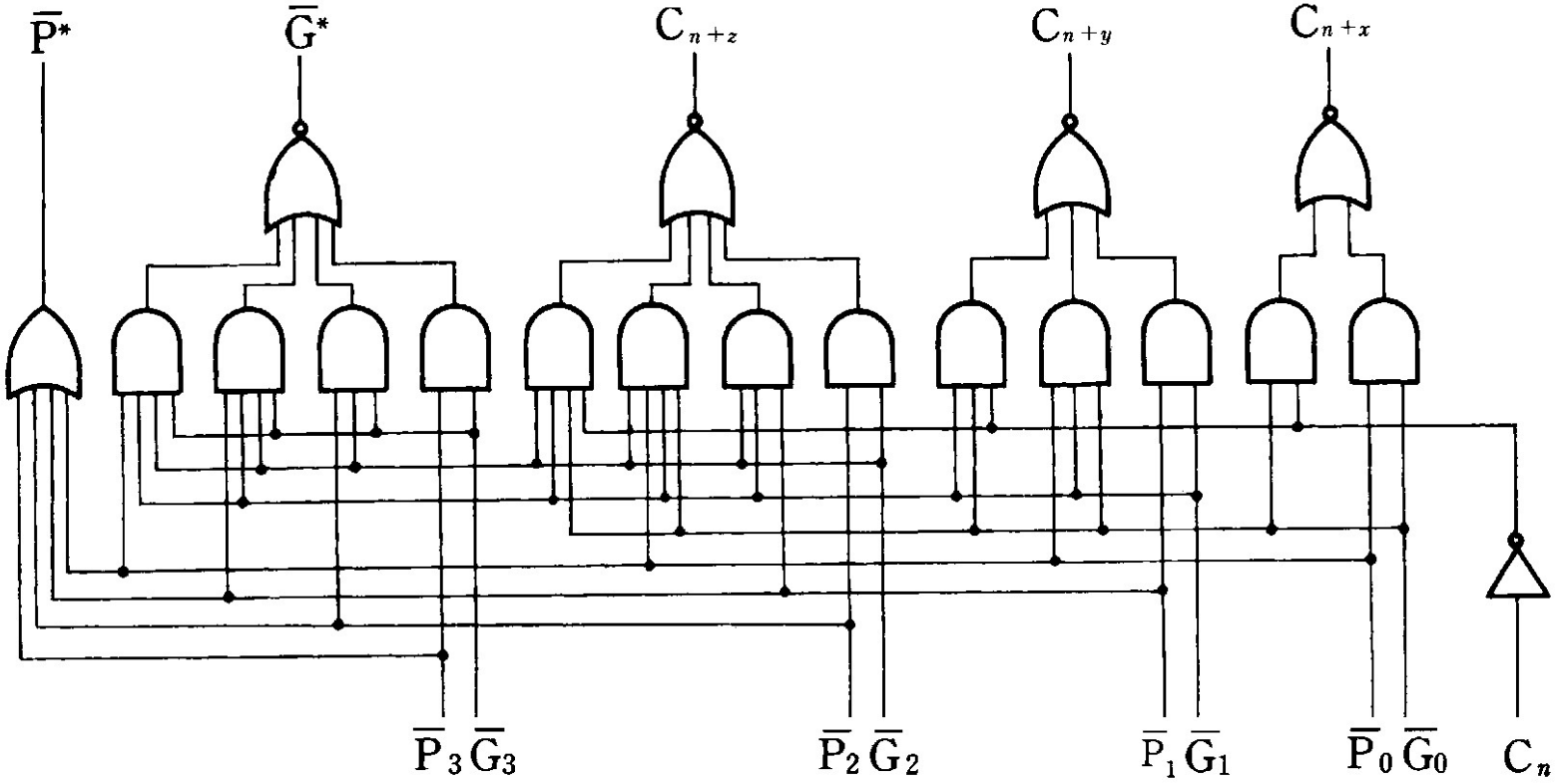
# 2.5.2 多功能算术/逻辑运算单元ALU



- 4、两级先行进位的ALU
  - 4片（组）的先行进位逻辑
  - $C_{n+x}=G_0+P_0C_n$
  - $C_{n+y}=G_1+P_1C_{n+x}=G_1+G_0P_1+P_0P_1C_n$
  - $C_{n+x}=G_2+P_2C_{n+y}$
  - $=G_2+G_1P_2+G_0P_1P_2+P_0P_1P_2C_n$
  - $C_{n+4}=G_3+P_3C_{n+z}$
  - $=G_3+G_2P_3+G_1P_2P_3+G_0P_1P_2P_3+P_0P_1P_2P_3C_n$
  - $=G^*+P^*C_n$
    - G*为成组先行进位发生输出
    - P*为成组先行进位传送输出

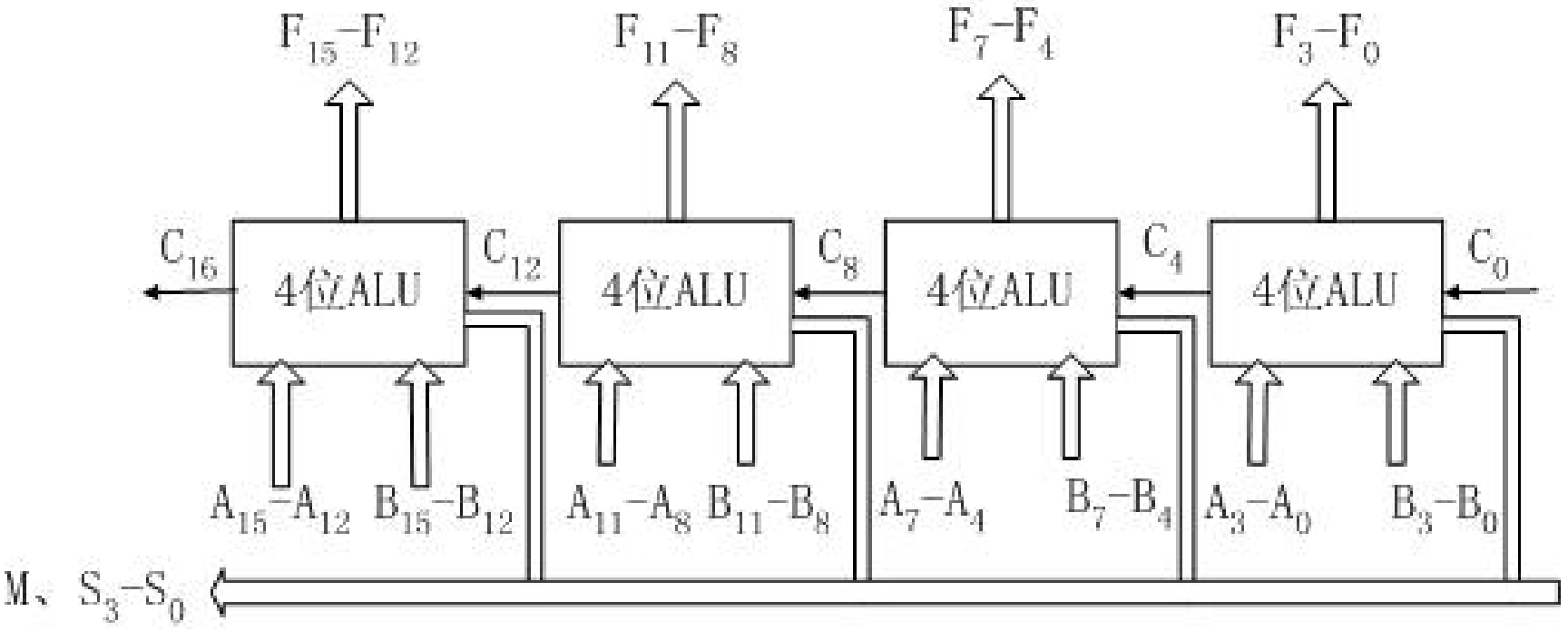
# 2.5.2 多功能算术/逻辑运算单元ALU

## 成组先行进位部件CLA的逻辑图



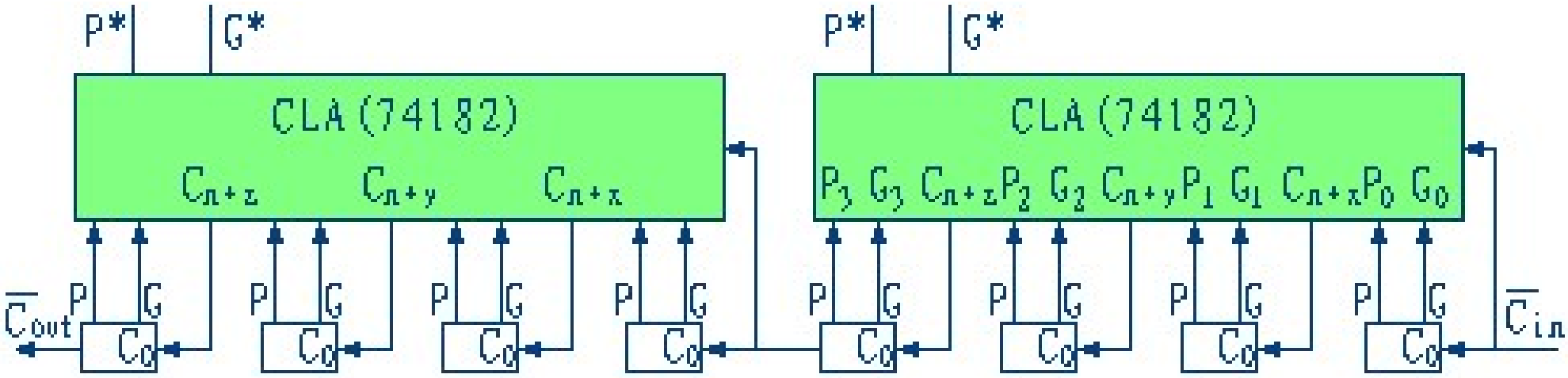
# 2.5.2 多功能算术/逻辑运算单元ALU

- 设计16位ALU




- $C_{n+x}=G_2+P_2C_{n+y}$ $C_{n+4}=G_3+P_3C_{n+z}$
- 片内先行进位,片间先行进位.

# 2.5.2 多功能算术/逻辑运算单元ALU




2个74L182

8个4位ALU74L181

图2.13 用两个16位全先行进位逻辑级联组成的32位ALU

# 2.5.3 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

# 2.5.3 内部总线

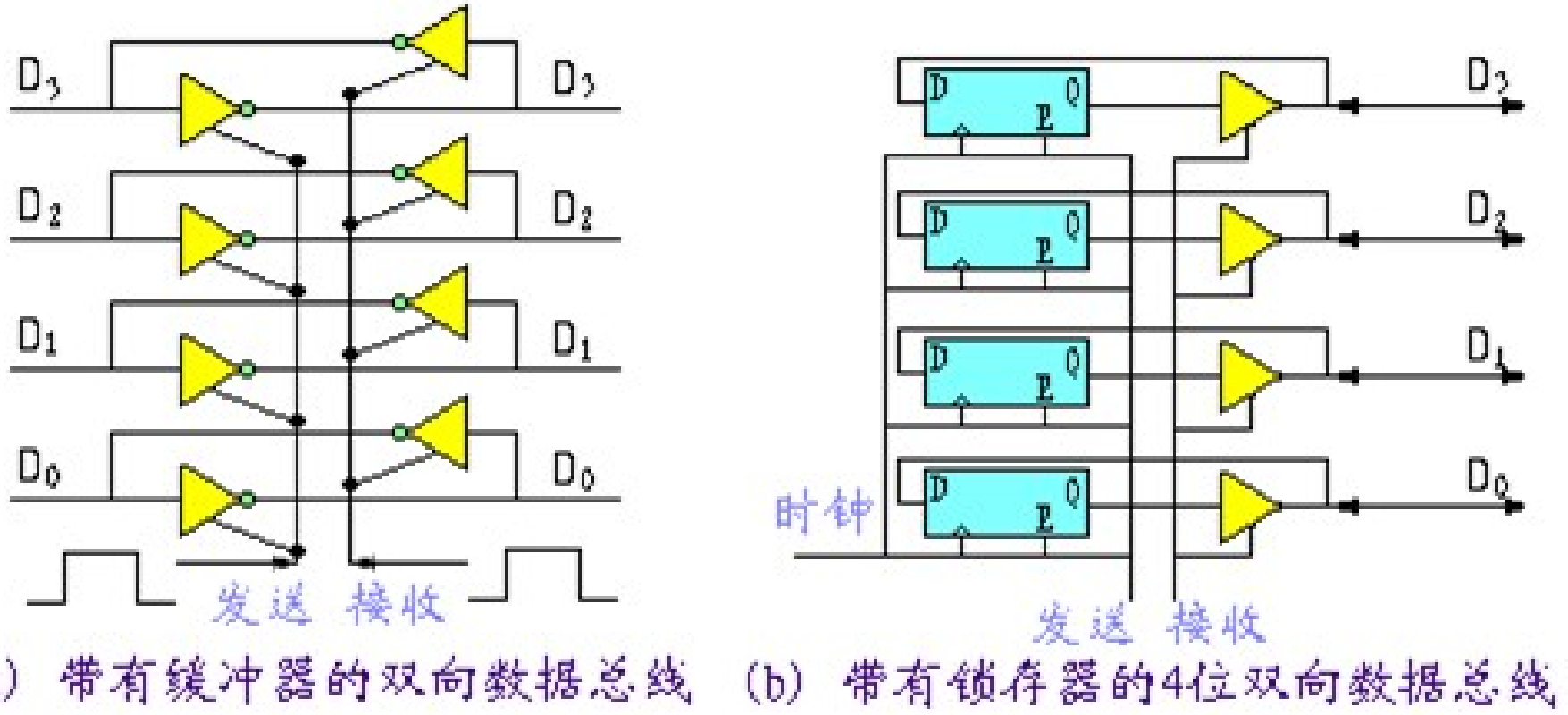
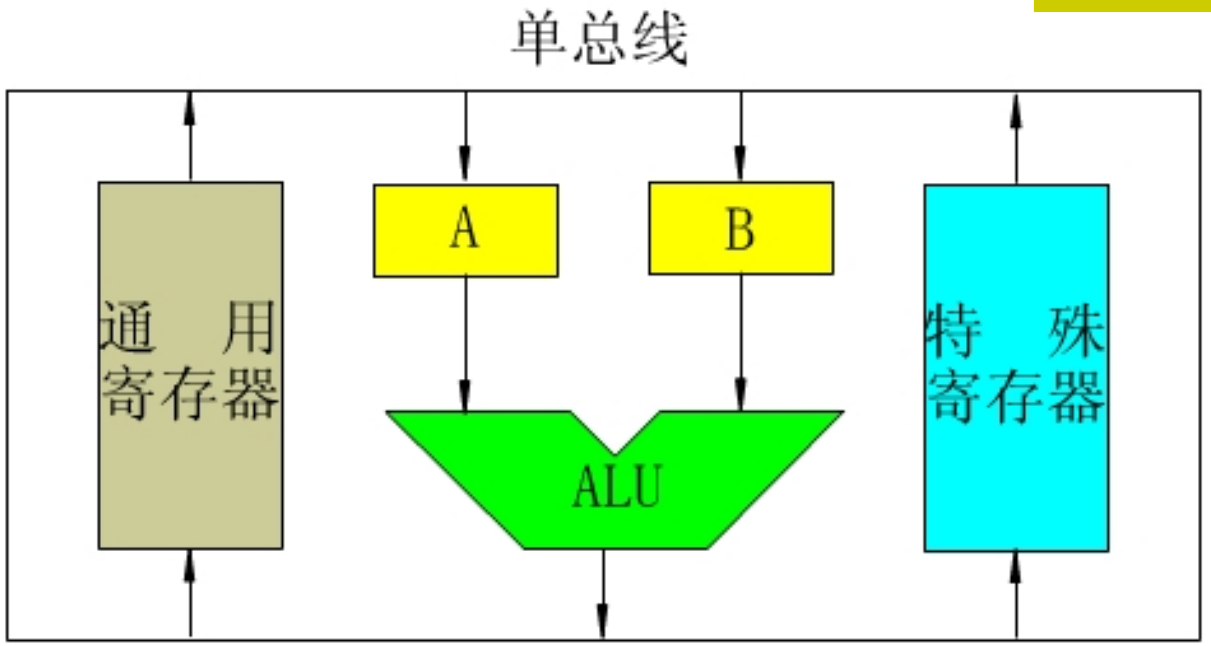



图2.14 由三态门组成的双向数据总线

# 2.5.4 定点运算器的基本结构

## 1、单总线结构的运算器

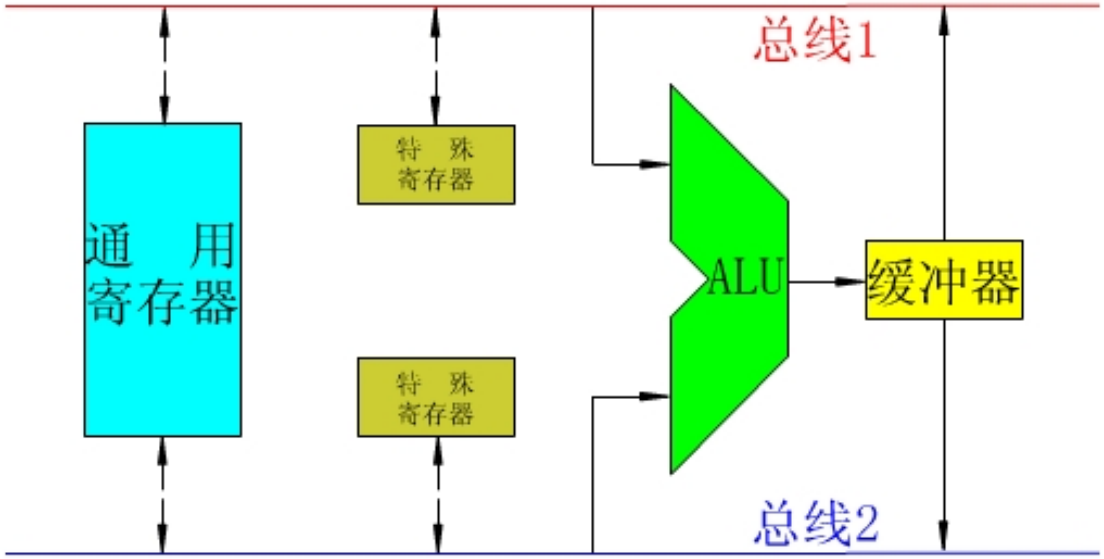



（a）单总线结构的运算器

图2.15 运算器的三种基本结构形式

# 2.5.4 定点运算器的基本结构

## 2、双总线结构的运算器

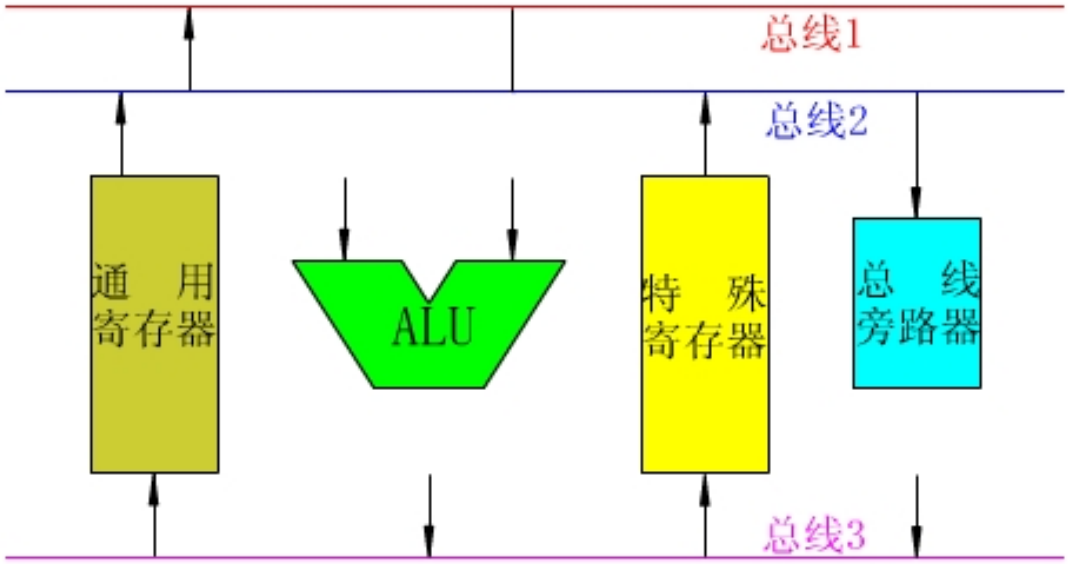



（b）双总线结构的运算器

图2.15 运算器的三种基本结构形式

# 2.5.4 定点运算器的基本结构

## 3、三总线结构的运算器




（c）三总线结构的运算器

图2.15 运算器的三种基本结构形式

# 2.6 浮点运算方法和浮点运算器

## 2.6.1 浮点加法、减法运算

## 2.6.2 浮点乘法、除法运算

## 2.6.3 浮点运算流水线

## 2.6.4 浮点运算器实例

# 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

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

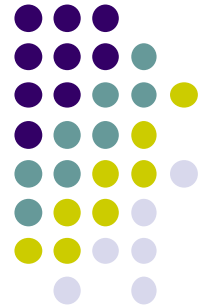
$$y = 2^{E_y} \cdot M_y$$

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

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

设$E_x <= E_y$

# 2.6.1 浮点加法、减法运算

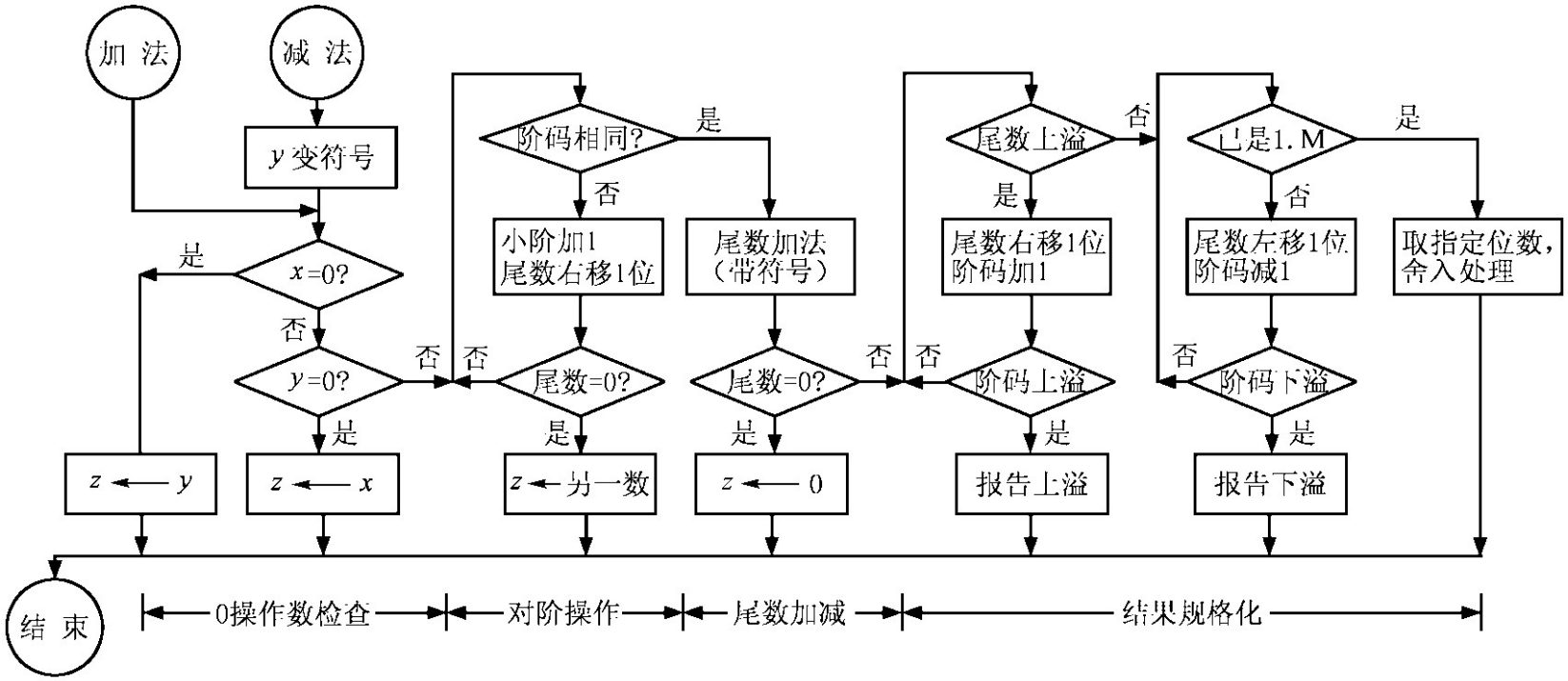


2、浮点运算步骤如下：

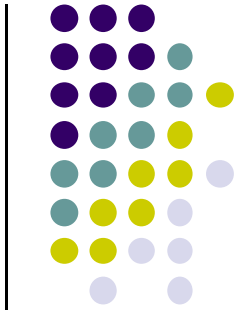
1. 0 操作数检查；
2. 比较阶码并完成对阶（小阶向大阶对齐）；
3. 尾数求和运算；
4. 结果规格化；
5. 舍入处理。

# 2.6.1 浮点加法、减法运算

- 浮点加减法运算操作流程



# 2.6.1 浮点加法、减法运算



[例28] 设$x=2^{010}\times 0.11011011$, $y=-2^{100}\times 0.10101100$，求**x+y** 。

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。

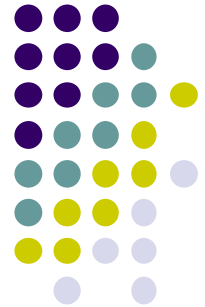
$[x]_{浮}$=00010, 0.11011011; $[y]_{浮}$=00100, 1.01010100

阶差=$[Ex]_{补}$-$[Ey]_{补}$=00010-00100=11110

即阶差为-2，Mx右移两位，Ex加2。

$[x]_{浮}$=00100, 0.00110110(11)

# 2.6.1 浮点加法、减法运算



3、尾数相加

- 0. 0 0 1 1 0 1 1 0 (11)
  ＋ 1. 0 1 0 1 0 1 0 0

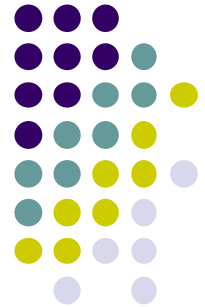
  ---

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

4、结果规格化

- 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

# 2.6.1 浮点加法、减法运算



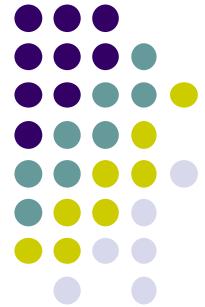
- 舍入处理（对阶和向右规格化时）
  - 就近舍入(0舍1入):类似”四舍五入”,丢弃的最高位为1,进1
  - 朝0舍入:截尾
  - 朝＋∞舍入:正数多余位不全为”0”,进1;负数,截尾
  - 朝－∞ 舍入:负数多余位不全为”0”,进1;正数,截尾

  采用0舍1入法处理，得到1.00010110。
- 溢出判断和处理
  - 阶码上溢，一般将其认为是＋∞和－∞ 。
  - 阶码下溢，则数值为0。

  阶码符号位为00，不溢出。得最终结果为

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

# 2.6.1 浮点加法、减法运算



[例29] 设$x = 10^{Ex} \times Mx = 10^{2} \times 0.3$ ,
$y = 10^{Ey} \times My = 10^{3} \times 0.2$ , 求 x+y=? x-y=?

解：Ex=2, Ey=3, Ex<Ey, 对阶时小阶向大阶看齐。

$$x+y = (Mx \cdot 10^{Ex-Ey} + My) \times 10^{Ey}$$
$$= (0.3 \times 10^{2-3} + 0.2) \times 10^{3}$$
$$= 0.23 \times 10^{3} = 230$$

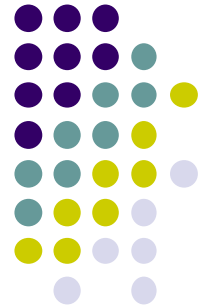
$$x-y = (Mx \cdot 10^{Ex-Ey} - My) \times 10^{Ey}$$
$$= (0.3 \times 10^{2-3} - 0.2) \times 10^{3}$$
$$= -0.17 \times 10^{3} = -170$$

# 课堂作业

- 设[ x 1]补＝11.01100000,
  [ $x$2]补＝11.01100001,
  [ $x$3]补＝11.01101000,
  [ $x$4]补＝11.01111001,

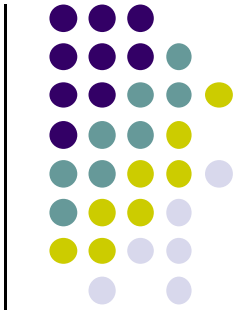
  求执行只保留小数点后4位有效数字的舍入操作值。

# 2.6.1 浮点加法、减法运算



- 课堂练习：$x=0.1101*2^{01}$　$y=-0.1010*2^{11}$
- 尾数和阶符都采用补码表示，都采用双符号位表示法。
- 求x+y

# 2.6.1 浮点加法、减法运算



$[x]_{浮}$=0001，00.1101

$[y]_{浮}$=0011，11.0110

阶差=1110　　即为-2

Mx应当右移2位，

$[x]_{浮}$=0011，00.0011（01）

尾数和为11.1001（01）

左规11.0010（10），阶码减1为0010

舍入（就近舍入）11.0011　　丢弃10

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

# 2.6.2 浮点乘法和除法运算

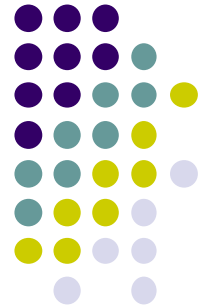
- 设有两个浮点数 x 和 y：

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

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

- $x \times y = 2^{(E_x + E_y)} \cdot (M_x \times M_y)$
- $x \div y = 2^{(E_x - E_y)} \cdot (M_x \div M_y)$
- 乘除运算分为四步
  - 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

# 2.6.2 浮点乘法和除法运算



- 补码采用双符号位，为了对溢出进行判断
- 00　为正　　11　为负
- 01　上溢　　10　下溢

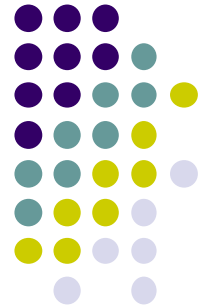
$x=+011, y=+110$,求$[x+y]_{补}$ 和 $[x-y]_{补}$,并判断是否溢出。

$[x]_{补}=00011, [y]_{补}=00110, [-y]_{补}=11010$

$[x+y]_{补}=[x]_{补}+[y]_{补}=01001$, 结果上溢。

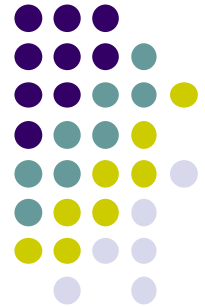
$[x-y]_{补}=[x]_{补}+[-y]_{补}=11101$, 结果正确,为−3。

# 2.6.2 浮点乘法和除法运算



- 尾数处理
  - 截断
  - 舍入
    - 尾数用原码表示时
      - 只要尾数最低为1或者移出位中有1数值位，使最低位置1
      - 0舍1入
    - 尾数用补码表示时
      - 丢失的位全为0，不必舍入。
      - 丢失的最高位为0，以后各位不全为0时；或者最高为1，以后各位全为0时，不必舍入。
      - 丢失的最高位为1，以后各位不全为0时，则在尾数的最低位入1的修正操作。

# 2.6.2 浮点乘法和除法运算



[例30]　　设有浮点数 $x=2^{-5}\times0.0110011$, $y=2^{3}\times(-0.1110010)$,阶码用4位移码表示,尾数(含符号位)用8位补码表示。求$[x\times y]_{浮}$。要求用补码完成尾数乘法运算,运算结果尾数保留高8位(含符号位),并用尾数低位字长值处理舍入操作。

[解:]

阶码采用双符号位,尾数原码采用单符号位,则有

$[Mx]_{原}$=0.0110011 , $[My]_{原}$=1.1110010

$[Ex]_{补}$=11011 , $[Ey]_{补}$=00011

$[x]_{浮}$=11011,0.0110011 , $[y]_{浮}$=00011,1.1110010

(1) 求阶码和：$[Ex]_{补}+[Ey]_{补}$=11011+00011=11110 (补码形式-2)

(2) 尾数乘法运算可采用原码阵列乘法器实现，即有

$[Mx]_{原}\times[My]_{原}=[0.0110011]_{原}\times[1.1110010]_{原}$

$=[1.0101101,0110110]_{原}$

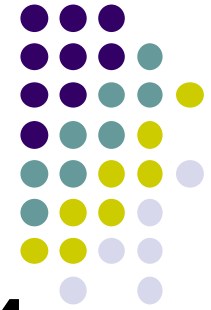
(3) 规格化处理：乘积不是规格化的数，需要左规。尾数左移1位变为1.1011010,1101100 , 阶码变为11101 (-3)。

(4) 舍入处理：尾数为负数，取高位字长，按舍入规则舍去低位字长，故尾数为1.1011011 。

最终相乘结果为 $[x\times y]_{浮}$=11101,1.1011011

其真值为 $x\times y=2^{-3}\times(-0.1011011)$

# 2.6.2 浮点乘法和除法运算

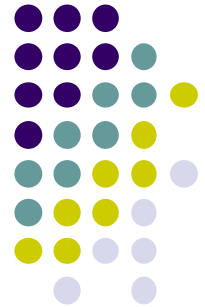


- **[例31] 设基数R=10, $x=10^{Ex}\times Mx=10^{2}\times 0.4$ , $y=10^{Ey}\times My=10^{3}\times 0.2$ , 用浮点法求x×y=? x÷y=?**

  解： **Ex=2, Ey=3, Mx=+0.4, My=+0.2**

$$x\times y=10^{(Ex+Ey)}\times(Mx\times My)=10^{2+3}\times(0.4\times 0.2)$$
$$=8000$$
$$x\div y=10^{(Ex-Ey)}\times(Mx\div My)=10^{2-3}\times(0.4\div 0.2)$$
$$=0.2$$

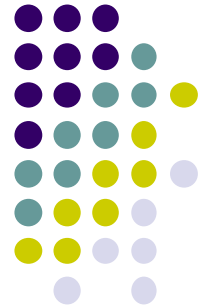
# 2.6.3 浮点运算流水线



1、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

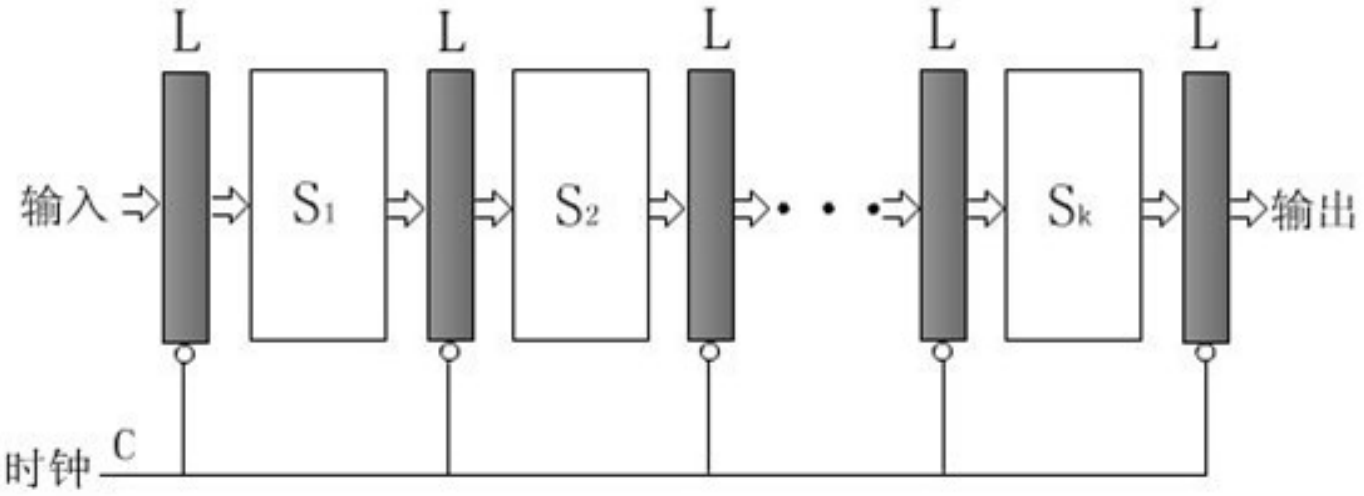
# 2.6.3 浮点运算流水线



## 2、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

# 流水线原理

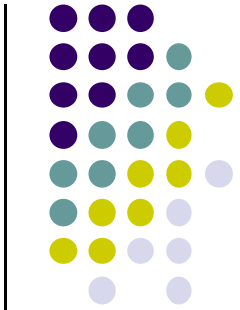



设过程段 $S_i$所需的时间为 $\tau_i$，缓冲寄存器的延时为 $\tau_l$，线性流水线的时钟周期定义为

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

流水线处理的频率为 $f = 1/\tau$。

# 流水线原理



- 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$T_k = k + (n-1)$，

  所需要的时间为： $T = T_k \times \tau$

  而同时，顺序完成的时间为：$T = n \times k \times \tau$

- k级线性流水线的加速比：

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

# 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

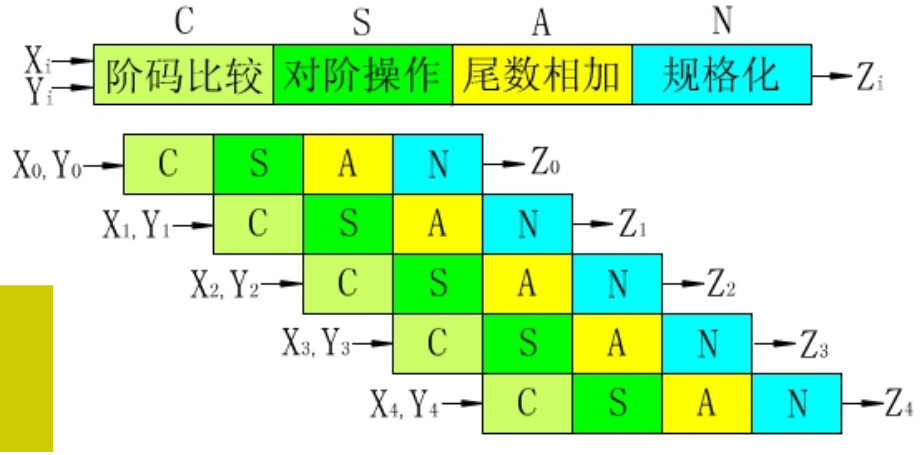
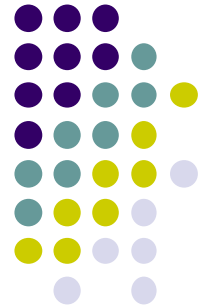




图2.21　向量加法计算的流水时空图

# 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）

# 2.6.4 浮点运算器实例

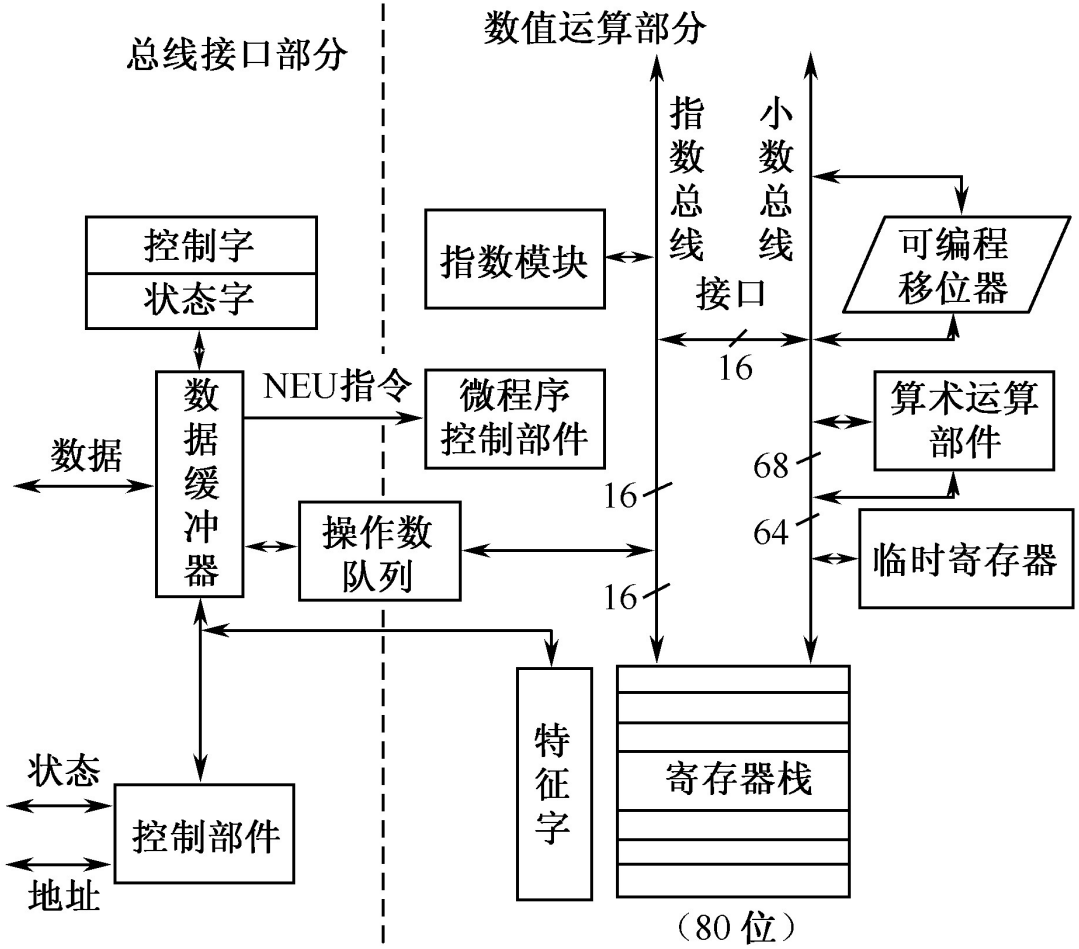
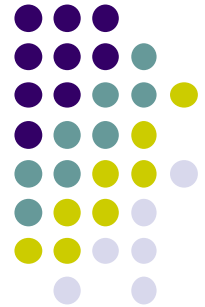



图2.23 80x87浮点运算器逻辑框图

# 本章小结



- 一个定点数由符号位和数值域两部分组成。按小数点位置不同，定点数有纯小数和纯整数两种表示方法。
- 按IEEE754标准，一个浮点数由符号位S、阶码E、尾数M三个域组成。其中阶码E的值等于指数的真值e加上一个固定偏移值。
- 为了使计算机能直接处理十进制形式的数据，采用两种表示形式：1、字符串形式，主要用在非数值计算的应用领域；2、压缩的十进制数串形式，用于直接完成十进制数的算术运算。

# 本章小结

- 数的真值变成机器码时有四种表示方法：原码表示法、反码表示法、补码表示法、移码表示法。其中移码主要用于表示浮点数的阶码E，以利于比较两个指数的大小和对阶操作。
- 字符信息属于符号数据，是处理非数值领域的问题。国际上采用的字符系统是七位的ASCII码。
- 直接使用西文标准键盘输入汉字，进行处理，并显示打印汉字，是一项重大成就。为此要解决汉字的输入编码、汉字内码、字模码等三种不同用途的编码。

# 本章小结

- 为运算器构造的简单性，运算方法中算术运算通常采用补码加减法，原码乘除法或补码乘除法。为了运算器的高速性和控制的简单性，采用了先行进位、阵列乘除法、流水线等并行技术措施。运算方法和运算器是本章的重点。
- 定点运算器和浮点运算器的结构复杂程度有所不同。早期微型机中浮点运算器放在CPU芯片外，随着高密度集成电路技术的发展，现已移至CPU内部。