河北师范大学软件学院
Software College of Hebei Normal University

第四章 数值的机器运算

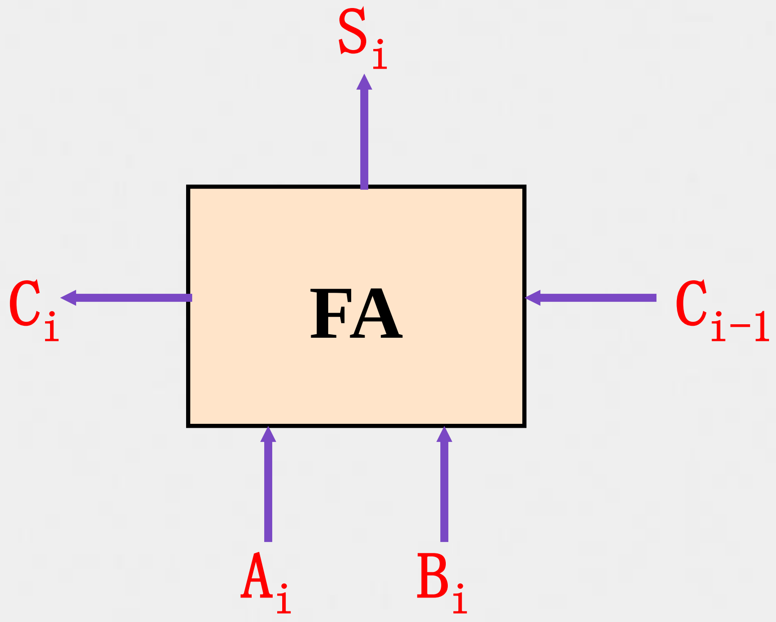
前 言

运算器是计算机进行算术运算和逻辑运算的主要部件，运算器的逻辑结构取决于机器的指令系统、数据表示方法和运算方法等。本章主要讨论数值数据在计算机中实现算术运算和逻辑运算的方法，以及运算部件的基本结构和工作原理。

目 录

4.1.1 加法器

- 加法器是由全加器再配以其他必要的逻辑电路组成的。
- 全加器
  - 基本的加法单元称为全加器，它要求三个输入量：操作数A_i和B_i、低位传来的进位C_{i-1}，并产生两个输出量：本位和S_i、向高位的进位C_i。



4.1.1 加法器

- 全加器真值表

A_i	B_i	C_{i-1}	S_i	C_i
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

4.1.1 加法器

- 全加器的逻辑表达式为：
  - $S_i=A_i \oplus B_i \oplus C_{i-1}$
  - $C_i=A_iB_i+(A_i \oplus B_i)C_{i-1}$
- 串行加法器与并行加法器
  - 在串行加法器中，只有一个全加器，数据逐位串行送入加法器进行运算。
  - 并行加法器由多个全加器组成，其位数的多少取决于机器的字长，数据的各位同时运算。

4.1.2 进位

进位产生函数用G_i表示

进位传递函数用P_i表示

- 进位表达式：
  - $C_i = A_iB_i + (A_i \oplus B_i)C_{i-1}$
  - $\therefore C_i = G_i + P_iC_{i-1}$
  - $G_i = A_iB_i$的含义是：若本位的两个输入均为1，必然要向高位产生进位。
  - $P_i = A_i \oplus B_i$的含义是：当两个输入中有一个为1，低位传来的进位C_{i-1}将超越本位向更高的位传送。

4.1.2 进位的产生与传递

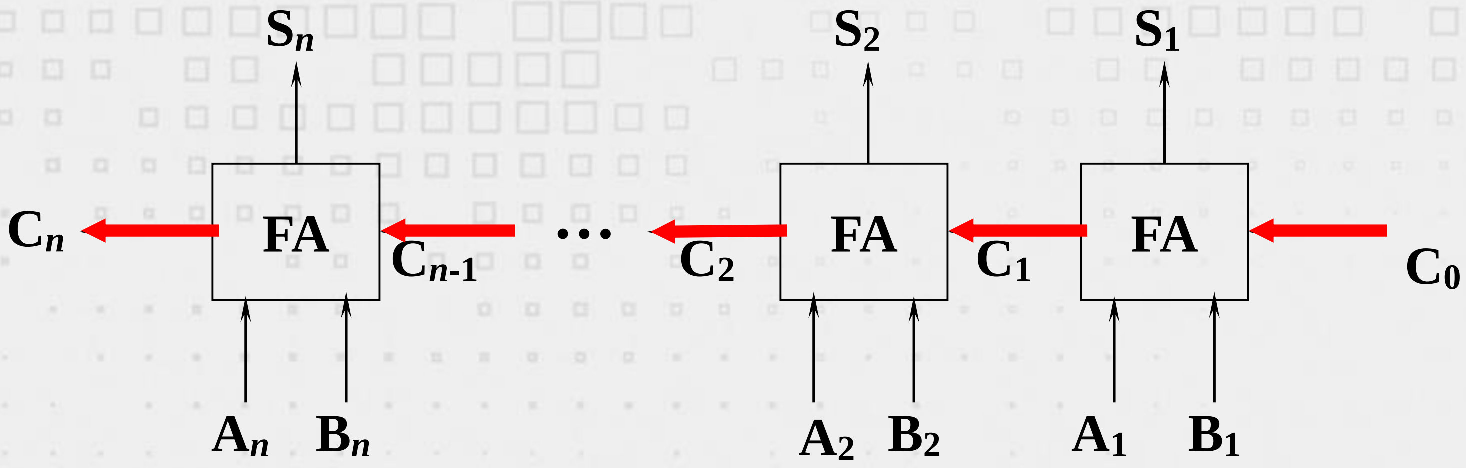
- 把n个全加器串接起来，就可进行两个n位数的相加。串行进位又称行波进位，每一级进位直接依赖于前一级的进位，即进位信号是逐级形成的。

$$C_1=G_1+P_1C_0$$

$$C_2=G_2+P_2C_1$$

$$\vdots$$

$$C_n=G_n+P_nC_{n-1}$$



$$C_1=G_1+P_1C_0$$
$$C_2=G_2+P_2C_1$$
$$\vdots$$
$$C_n=G_n+P_nC_{n-1}$$

- 串行进位链的总延迟时间与字长成正比。假定，将一级门的延迟时间定为ty，从上述公式中可看出，每形成一级进位的延迟时间为2ty。在字长为n位的情况下，若不考虑G_i、P_i的形成时间，从C0→C_n的最长延迟时间为2nty。

4.1.3 并行加法器的快速进位

- 1. 并行进位方式
  - 并行进位又叫先行进位、同时进位，其特点是各级进位信号同时形成。

$$C_1=G_1+P_1C_0$$

$$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$$

$$C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$$

$$C_4=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0$$

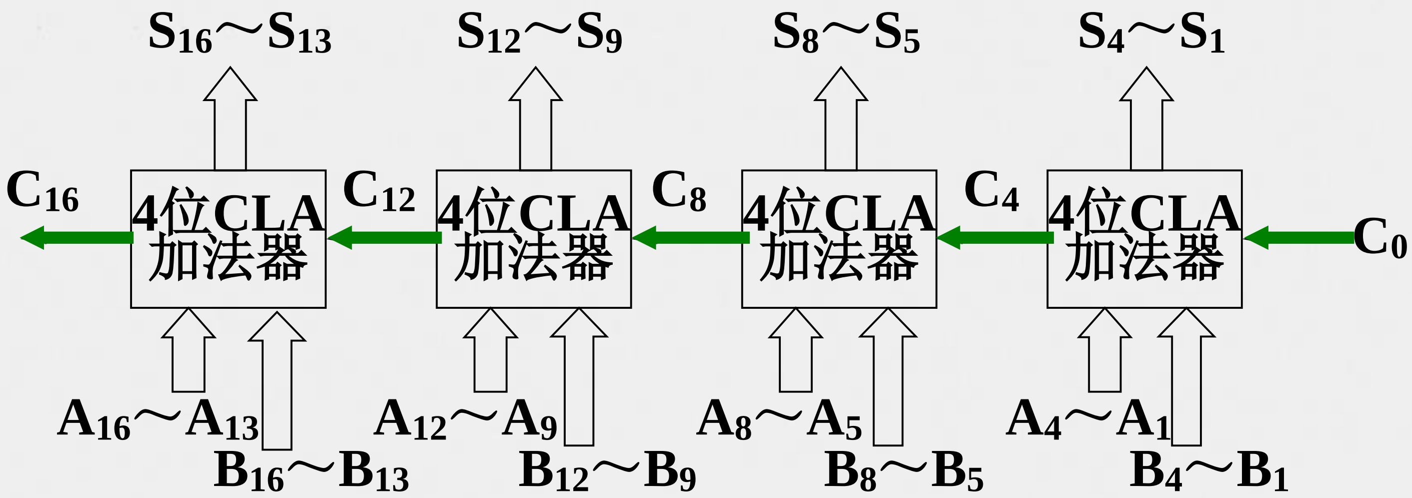
4.1.3

并行加法器的快速进位

- 2. 分组并行进位方式
  - 实际上，通常采用分组并行进位方式。这种进位方式是把n位字长分为若干小组，在组内各位之间实行并行快速进位，在组间既可以采用串行进位方式，也可以采用并行快速进位方式，因此有两种情况。

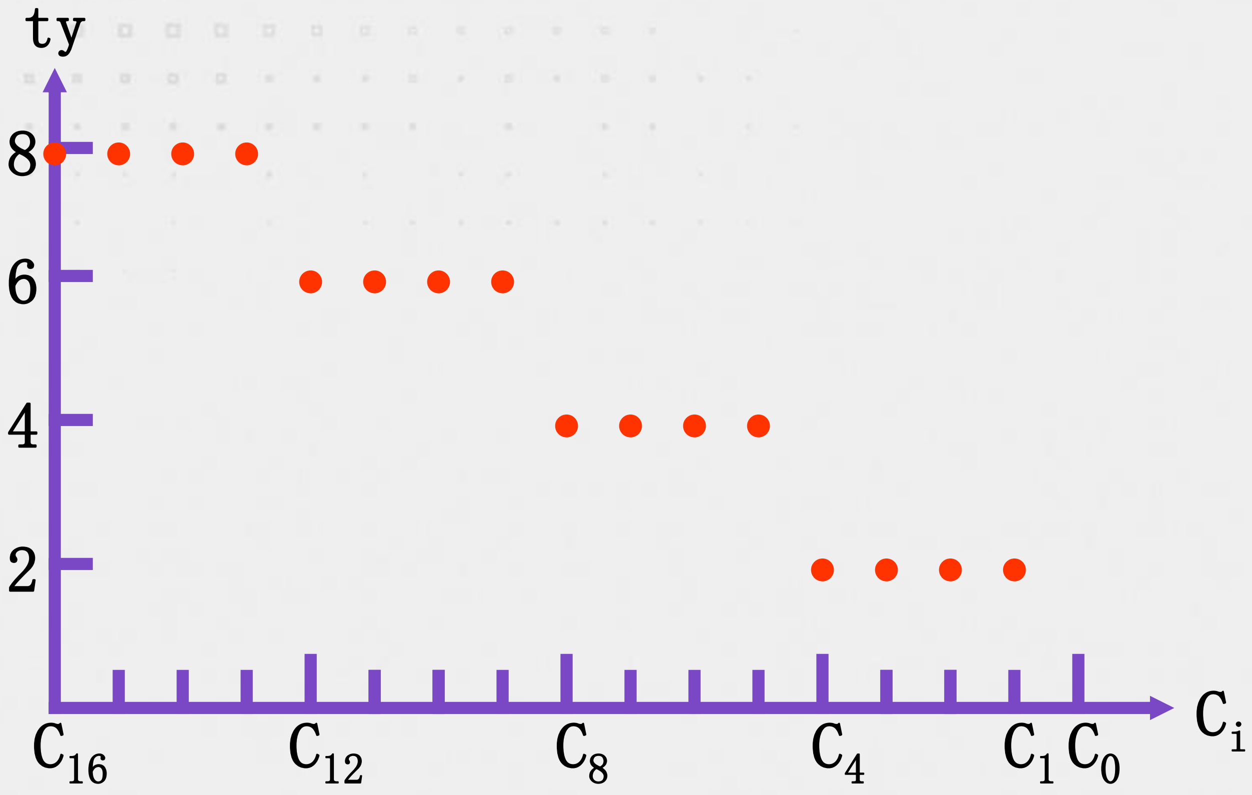
4.1.3 并行加法器的快速进位

- (1)单级先行进位方式
  - 这种进位方式又称为组内并行、组间串行方式。以16位加法器为例，可分为四组，每组四位。



4.1.3 并行加法器的快速进位

- (1)单级先行进位方式

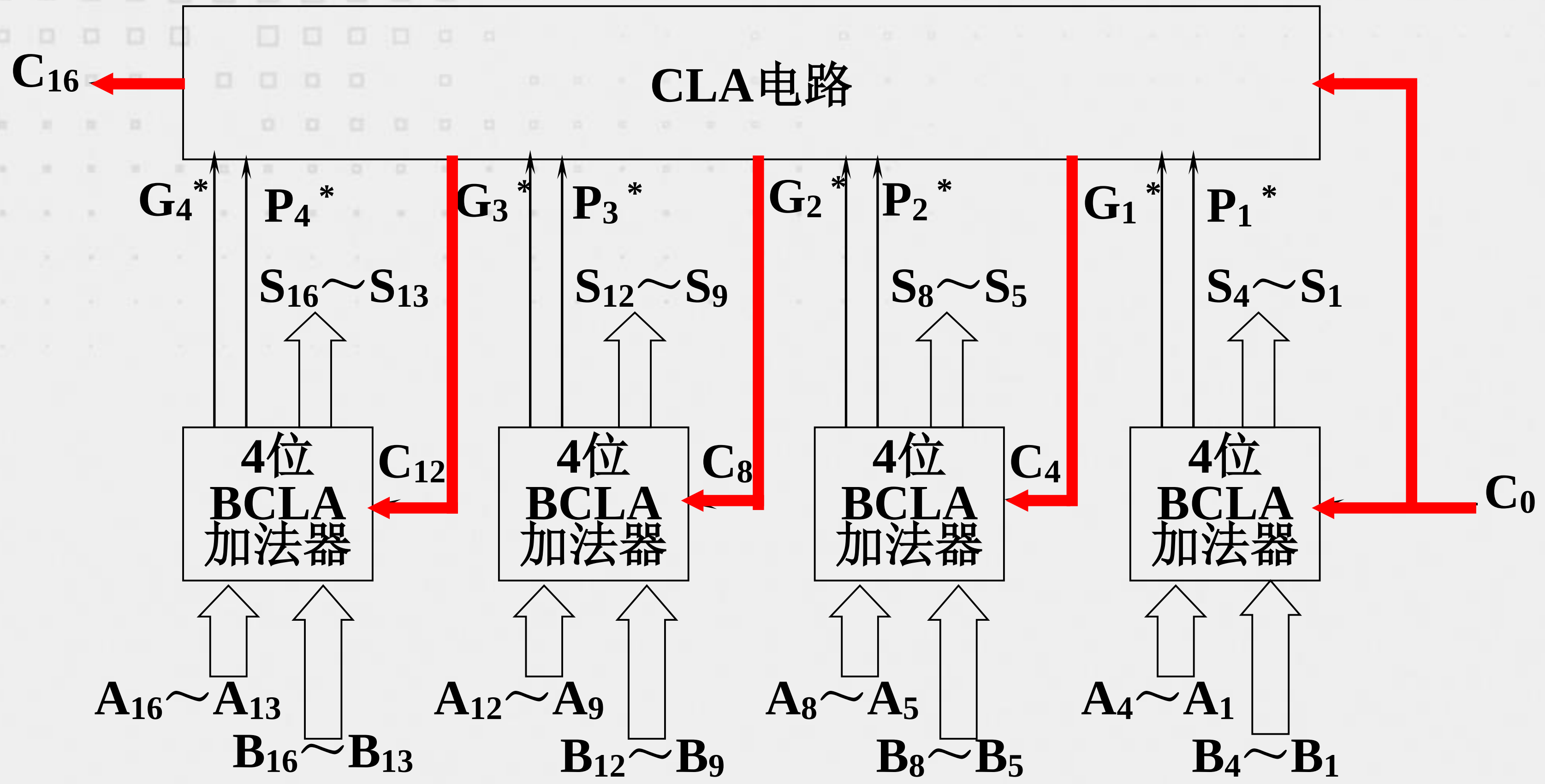


4.1.3 并行加法器的快速进位

- (2)多级先行进位方式
  - 多级先行进位又称组内并行、组间并行进位方式
  - 字长为16位的两级先行进位加法器，第一小组的最高位进位C4：
  - $C_4=\underbrace{G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1}_{组进位产生函数G_1^*}+\underbrace{P_4P_3P_2P_1}_{组进位传递函数P_1^*}C_0=G_1^*+P_1^*C_0$
  - 依次类推：
  - $C_8=G_2^*+P_2^*G_1^*+P_2^*P_1^*C_0$
  - $C_{12}=G_3^*+P_3^*G_2^*+P_3^*P_2^*G_1^*+P_3^*P_2^*P_1^*C_0$
  - $C_{16}=G_4^*+P_4^*G_3^*+P_4^*P_3^*G_2^*+P_4^*P_3^*P_2^*G_1^*+P_4^*P_3^*P_2^*P_1^*C_0$

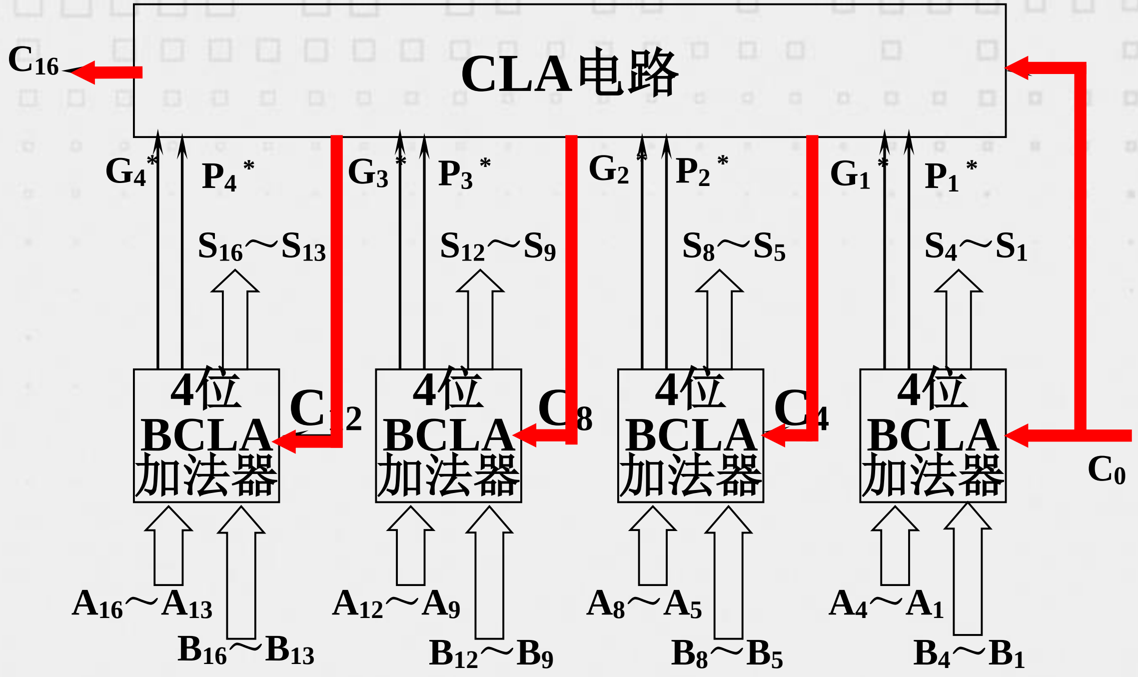
4.1.3 并行加法器的快速进位

- (2)多级先行进位方式



4.1.3 并行加法器的快速进位

- (2)多级先行进位方式
  - 若不考虑G_i、P_i的形成时间，C_0经过2ty产生第1小组的C_1、C_2、C_3及所有组进位产生函数G_i^*和组进位传递函数P_i^*；
  - 再经过2ty，产生C_4、C_8、C_{12}、C_{16}；
  - 最后经过2ty后，才能产生第2、3、4小组内的$C_5 \sim C_7$、$C_9 \sim C_{11}$、$C_{13} \sim C_{15}$。



$C_1=G_1+P_1C_0$

$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$

$C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$

$\cancel{C_4=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0}$

$G_1^*=G_4+P_4G_3+P_4P_3G_2$

$P_1^*=P_4P_3P_2G_1+P_4P_3P_2P_1$

$C_4=G_1^*+P_1^*C_0$

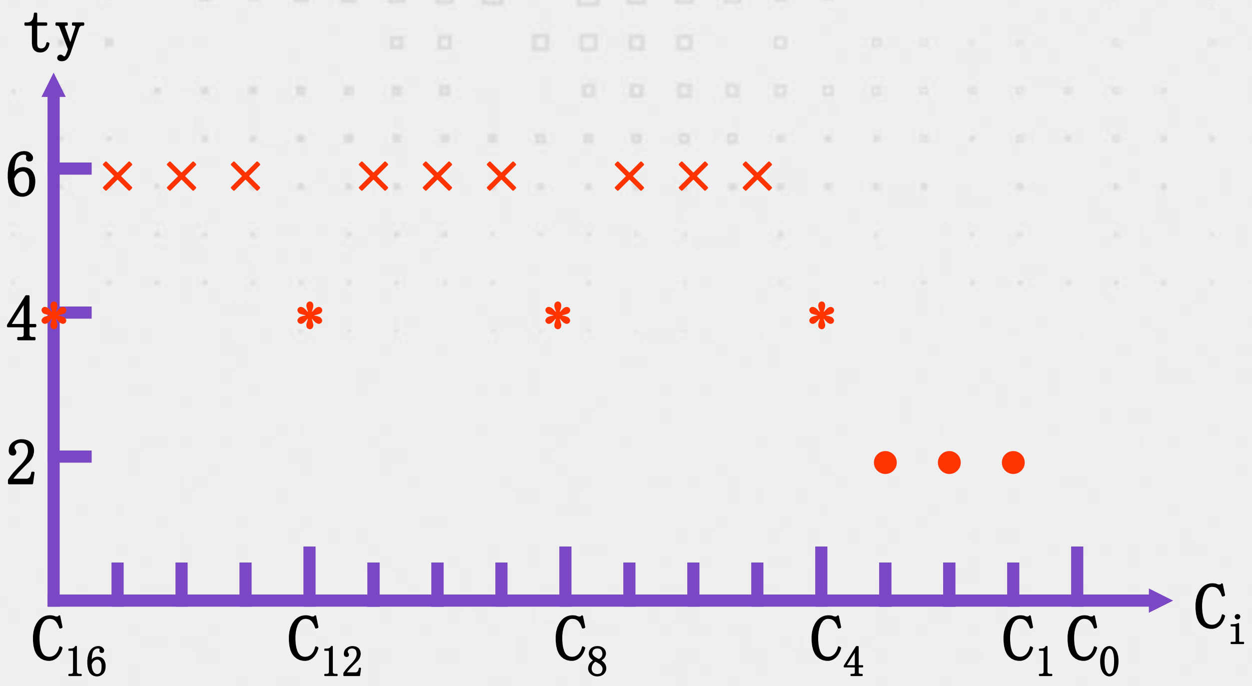
$C_8=G_2^*+P_2^*G_1^*+P_2^*P_1^*C_0$

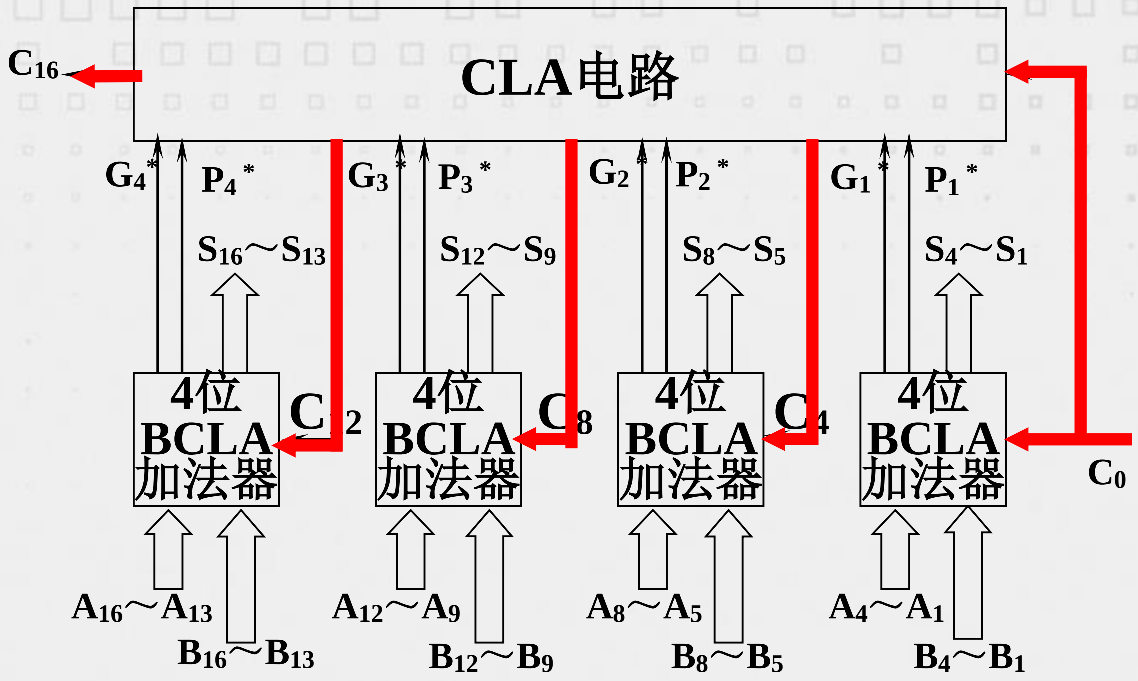
$C_{12}=G_3^*+P_3^*G_2^*+P_3^*P_2^*G_1^*+P_3^*P_2^*P_1^*C_0$

$C_{16}=G_4^*+P_4^*G_3^*+P_4^*P_3^*G_2^*+P_4^*P_3^*P_2^*G_1^*$
$+P_4^*P_3^*P_2^*P_1^*C_0$

4.1.3 并行加法器的快速进位

• (2)多级先行进位方式





$C_4=G_1^*+P_1^*C_0$

$C_8=G_2^*+P_2^*G_1^*+P_2^*P_1^*C_0$

$C_{12}=G_3^*+P_3^*G_2^*+P_3^*P_2^*G_1^*+P_3^*P_2^*P_1^*C_0$

$C_{16}=G_4^*+P_4^*G_3^*+P_4^*P_3^*G_2^*+P_4^*P_3^*P_2^*G_1^*$

$+P_4^*P_3^*P_2^*P_1^*C_0$

目录

4.2.1 原码加减运算

- 对原码表示的两个数进行加减运算时，符号位不参与运算，仅仅是两数的绝对值参与运算。

4.2.2 补码加减运算

- 1.补码加法
  - 两个补码表示的数相加，符号位参加运算，且两数和的补码等于两数补码之和，即
  - $[X+Y]_{补}=[X]_{补}+[Y]_{补}$
- 2.补码减法
  - 根据补码加法公式可推出：
  - $[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}$

4.2.2 补码加减运算

- 已知$[Y]_{补}$求$[-Y]_{补}$的方法是：将$[Y]_{补}$连同符号位一起求反，末位加“1”。
- $[-Y]_{补}$被称为$[Y]_{补}$的机器负数，由$[Y]_{补}$求$[-Y]_{补}$的过程称为对$[Y]_{补}$变补（求补），表示为：
- $[-Y]_{补}=[[Y]_{补}]_{变补}$

4.2.2 补码加减运算

- 要特别注意将“某数的补码表示”与“变补”这两个概念区分开来。
- 一个负数由原码表示转换成补码表示时，符号位不变，仅对数值位的各位变反，末位加“1”。而变补则不论这个数的真值是正是负，一律连同符号位一起变反，末位加“1”。
- $[Y]_{补}$表示的真值如果是正数，则变补后$[-Y]_{补}$所表示真值变为负数，反之亦然。

4.2.2 补码加减运算

- 例1：Y=-0.0110
  - $[Y]_{补}=1.1010$，$[-Y]_{补}=0.0110$
- 例2：Y=0.0110
  - $[Y]_{补}=0.0110$，$[-Y]_{补}=1.1010$

4.2.2 补码加减运算

- 3.补码加减运算规则
  - (1) 参加运算的两个操作数均用补码表示；
  - (2) 符号位作为数的一部分参加运算；
  - (3) 若做加法，则两数直接相加；若做减法，则将被减数与减数的机器负数相加；
  - (4) 运算结果用补码表示。

4.2.2 补码加减运算

- 例1：A=0.1011，B=-0.1110，求：A+B
- $\because [A]_{补}=0.1011$，$[B]_{补}=1.0010$

$$\begin{array}{r} 0.1011 \\ +\ 1.0010 \\ \hline 1.1101 \end{array}$$

- $\therefore [A+B]_{补}=1.1101$，$A+B=-0.0011$

- 例2：A=0.1011，B=-0.0010，求：A-B
- $\because [A]_{补}=0.1011$，$[B]_{补}=1.1110$，$[-B]_{补}=0.0010$

$$\begin{array}{r} 0.1011 \\ +0.0010 \\ \hline 0.1101 \end{array}$$

- $\therefore [A-B]_{补}=0.1101$， $A-B=0.1101$

- 1.溢出的产生
  - 在补码加减运算中，有时会遇到这样的情况：两个同号数相加，结果符号位却相反。
  - 例1：X=1011B=11D，Y=111B=7D

    $[X]_{补}$=0,1011，$[Y]_{补}$=0,0111

    $[X+Y]_{补}$=1,0010，X+Y=-1110B=-14D
  - 两正数相加结果为-14D，显然是错误的。

$$
\begin{array}{r}
0,1\ 0\ 1\ 1 \\
+\ \ 0,0\ 1\ 1\ 1 \\
\hline
1,0\ 0\ 1\ 0
\end{array}
$$

4.2.3 补码的溢出判断与检测

- 1.溢出的产生
  - 在补码加减运算中，有时会遇到这样的情况：两个同号数相加，结果符号位却相反。
  - 例2：X=-1011B=-11D，Y=-111B=-7D

    $[X]_{补}=1,0101$　$[Y]_{补}=1,1001$

    $[X+Y]_{补}=0,1110$，X+Y=1110B=14D
  - 两负数相加结果为14D，显然也是错误的。

$$\begin{array}{r} 1,0\ 1\ 0\ 1 \\ +\ \ 1,1\ 0\ 0\ 1 \\ \hline 0,1\ 1\ 1\ 0 \end{array}$$

4.2.3 补码的溢出判断与检测

- 为什么会发生这种错误呢？原因在于两数相加之和的数值已超过了机器允许的表示范围。
- 字长为n+1位的定点整数（其中一位为符号位），采用补码表示，当运算结果大于2^n-1或小于-2^n时，就产生溢出。

4.2.3 补码的溢出判断与检测

- 设参加运算的两数为X、Y，做加法运算。
- 若X、Y异号，不会溢出。
- 若X、Y同号，运算结果为正且大于所能表示的最大正数或运算结果为负且小于所能表示的最小负数（绝对值最大的负数）时，产生溢出。
- 将两正数相加产生的溢出称为正溢；反之，两负数相加产生的溢出称为负溢。

4.2.3 补码的溢出判断与检测

- 2. 溢出检测方法
  - (1)采用一个符号位
  - (2)采用进位位
  - (3)采用变形补码（双符号位补码）
- 设：被操作数为：$[X]_{补}=X_s,X_1X_2\ldots X_n$

  操作数为：$[Y]_{补}=Y_s,Y_1Y_2\ldots Y_n$

  其和（差）为：$[S]_{补}=S_s,S_1S_2\ldots S_n$

4.2.3 补码的溢出判断与检测

- (1)采用一个符号位
  - 两正数相加，结果为负表明产生正溢；两负数相加，结果为正表明产生负溢。因此可得出采用一个符号位检测溢出的方法：
  - 当$X_s=Y_s=0$，$S_s=1$时，产生正溢。
  - 当$X_s=Y_s=1$，$S_s=0$时，产生负溢。
  - 溢出 = $\overline{X_s}\,\overline{Y_s}S_s+X_sY_s\overline{S_s}$

4.2.3 补码的溢出判断与检测

- (2)采用进位位
  - 两数运算时，产生的进位为C_s,C_1C_2…C_n，
  - 其中：C_s为符号位产生的进位，C_1为最高数值位产生的进位。
  - 两正数相加，当最高有效位产生进位（C_1=1）而符号位不产生进位（C_s=0）时，发生正溢。
  - 两负数相加，当最高有效位没有进位（C_1=0）而符号位产生进位（C_s=1）时，发生负溢。
  - 溢出 = $C_s \oplus C_1$

4.2.3 补码的溢出判断与检测

- (3)采用变形补码（双符号位补码）
  - 双符号位分别用S_{s1}和S_{s2}表示
  - $S_{s1}S_{s2}=00$ 结果为正数，无溢出
  - $S_{s1}S_{s2}=01$ 结果正溢
  - $S_{s1}S_{s2}=10$ 结果负溢
  - $S_{s1}S_{s2}=11$ 结果为负数，无溢出
  - 当两位符号位的值不一致时，表明产生溢出。
  - 溢出$=S_{s1}\oplus S_{s2}$

4.2.3 补码的溢出判断与检测

- 前例中字长为5位，数的表示范围为-16～15，采用变形补码（双符号位）运算，则有：

- 11+7=18（正溢）

$$
\begin{array}{r}
00,1011 \\
+\ 00,0111 \\
\hline
01,0010
\end{array}
$$

- -11+(-7)=-18（负溢）

$$
\begin{array}{r}
11,0101 \\
+\ 11,1001 \\
\hline
10,1110
\end{array}
$$

课题习题

- 1、两补码相加，采用1位符号位，则当（**D**）时，表示结果溢出。

A.最高位有进位　　B.最高位为1

C.最高位进位和次高位进位异或结果为0

D.最高位进位和次高位进位异或结果为1

- 2、在定点补码运算器中，若采用双符号位，当（**B**）时表示结果溢出。

A.双符号位相同　　B.双符号位不同

C.两个正数相加　　D.两个负数相加

目 录

4.3.1 带符号数的移位操作

- 1.原码的移位规则
  - 负数的原码移位后的空出位补0

左移	1	X_2	X_3	···	X_n	0
右移	1	0	X_1	···	X_{n-2}	X_{n-1}

4.3.1 带符号数的移位操作

- 2.补码的移位规则
  - 负数的补码左移后的空出位补0，右移后的空出位补1。

左移	1	X_2	X_3	…	X_n	0
右移	1	1	X_1	…	X_{n-2}	X_{n-1}

4.3.1 带符号数的移位操作

- 3. 移位功能的实现
  - 通常移位操作由移位寄存器来实现。但也有一些计算机不设置专门的移位寄存器，而在加法器的输出端加一个实现直传、左移一位和右移一位的控制逻辑电路（称为移位器）。
  - 分别用2F→L、F→L和F/2→L 这三个不同控制信号选择左移、直传和右移操作。

课堂习题

- 3. 设机器字长8位（含1位符号位），若机器数BAH为原码，则算术左移一位和算术右移一位分别为（ C ）。

A.F4H　EDH　　B.B4H　6DH

C.F4H　9DH　　D.B5H　EDH

- 4. 某字长为8位的计算机中，已知整型变量x、y的机器数分别为$[x]_{补}$=1 1110100，$[y]_{补}$=1 0110000。若整型变量z=2*x+y/2，则z的机器数为（ A ）。

A. 11000000　　B .0 0100100

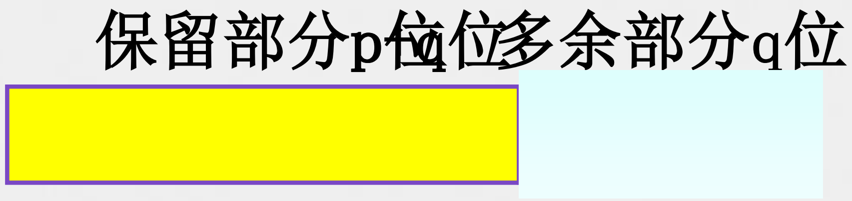
C .1 0101010　　D .溢出

4.3.2 带符号数的舍入操作

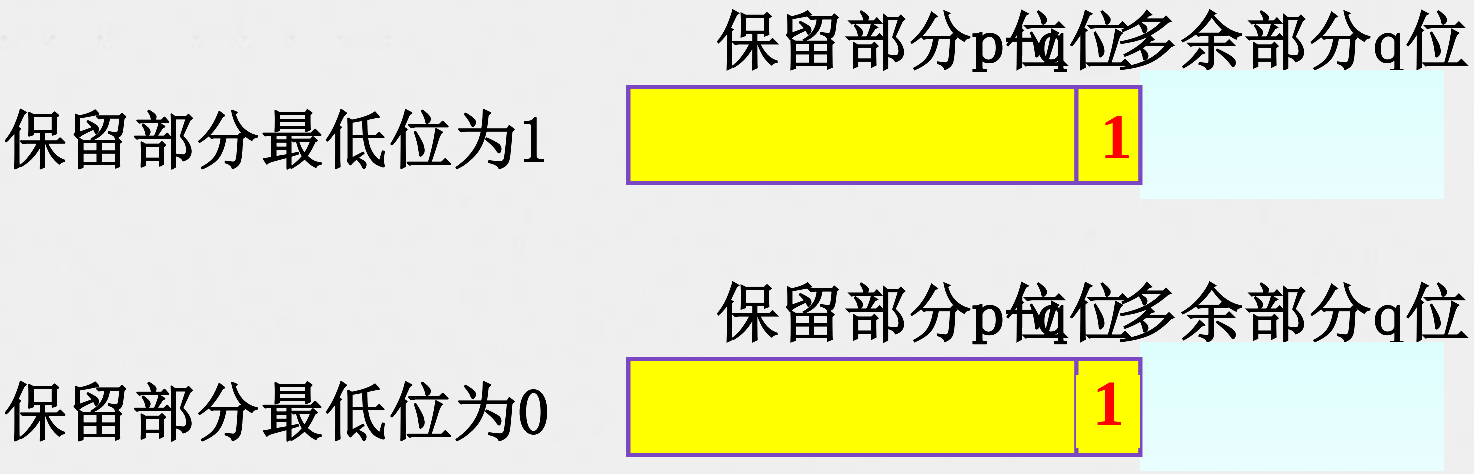
- 1. 恒舍（恒舍法）
- 2. 冯·诺依曼舍入法（恒置1法）
- 3. 下舍上入法（0舍1入）
- 4. 查表舍入法

4.3.2 带符号数的舍入操作

- 1.恒舍（切断）
  - 这是一种最容易实现的舍入方法，无论多余部分q位为何代码，一律舍去，保留部分p位不作任何改变。



- 2. 冯·诺依曼舍入法
  - 这种舍入法又称为恒置1法，即不论多余部分q位为何代码，都把p位的最低位置1。



4.3.2 带符号数的舍入操作

- 3. 下舍上入法
  - 下舍上入就是0舍1入。用将要舍去的q位部分的最高位作为判断标志，如该位为0，则舍去整个q位部分，如该位为1，则在前面的p位部分的最低位上加1。

保留部分p位 多余部分q位

多余部分最高位为0

保留部分p位 多余部分q位

多余部分最高位为1

+ 1

课堂习题

- 5. 如果采用0舍1入法进行舍入处理，则0.01010110011舍去一位后，结果为（ **B** ）。

A.0.0101011001　　B.0.0101011010

C.0.0101011011　　D.0.0101011100

- 6. 如果采用末位恒置1法进行舍入处理，则0.01010110011舍去最后一位后，结果为（ **A** ）。

A.0.0101011001　　B.0.0101011010

C.0.0101011011　　D.0.0101011100

THANK YOU