计算机的算术运算

part1：加法和减法

数据表示

- 两种常用的数据格式
  - **定点数**
    - 小数点位置固定
    - 定点整数
    - 定点小数
    - 数值范围有限,要求的处理硬件简单
  - **浮点数**
    - 小数点位置浮动
    - 数值范围很大,要求的处理硬件复杂

整数编码

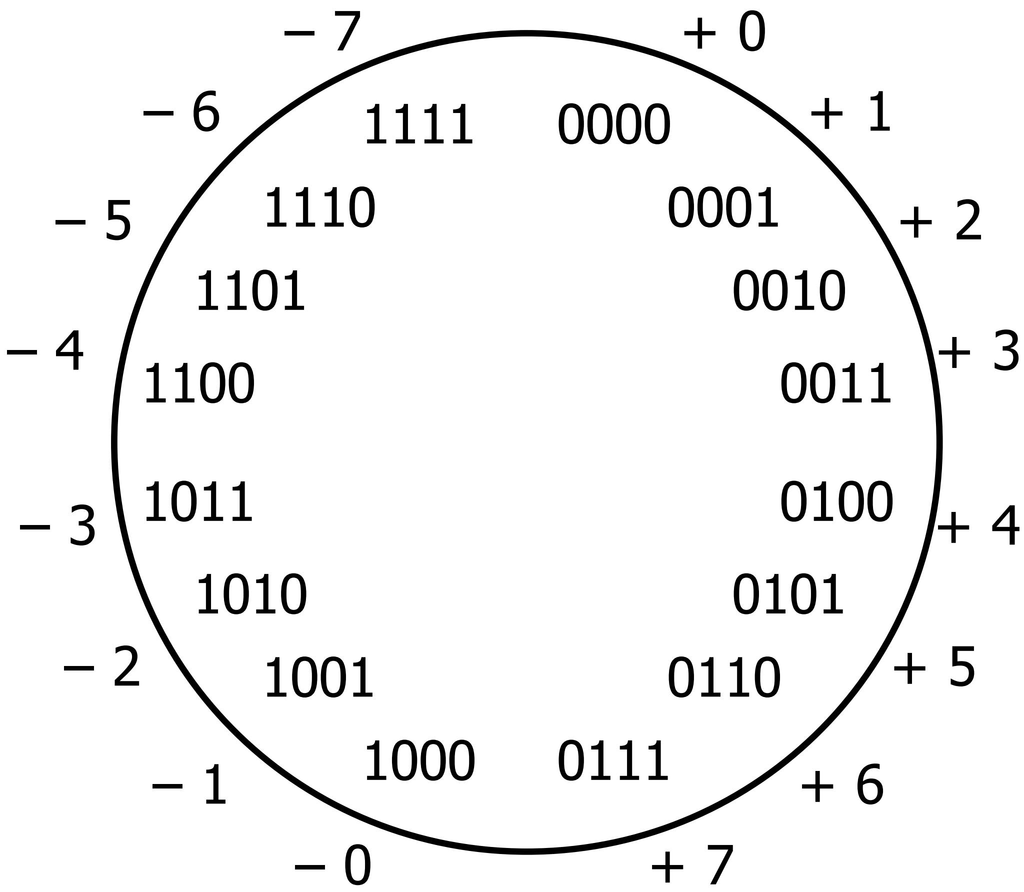
编码	原码	反码	补码
000	**0**	**0**	**0**
001	**1**	**1**	**1**
010	**2**	**2**	**2**
011	**3**	**3**	**3**
100	**-0**	**-3**	**-4**
101	**-1**	**-2**	**-3**
110	**-2**	**-1**	**-2**
111	**-3**	**-0**	**-1**

原码（Sign Magnitude）： 符号位||数的绝对值

反码（One’s Complement）： 符号位||数值按位求反

补码（Two’s Complement）： 反码的最低位+1

原码

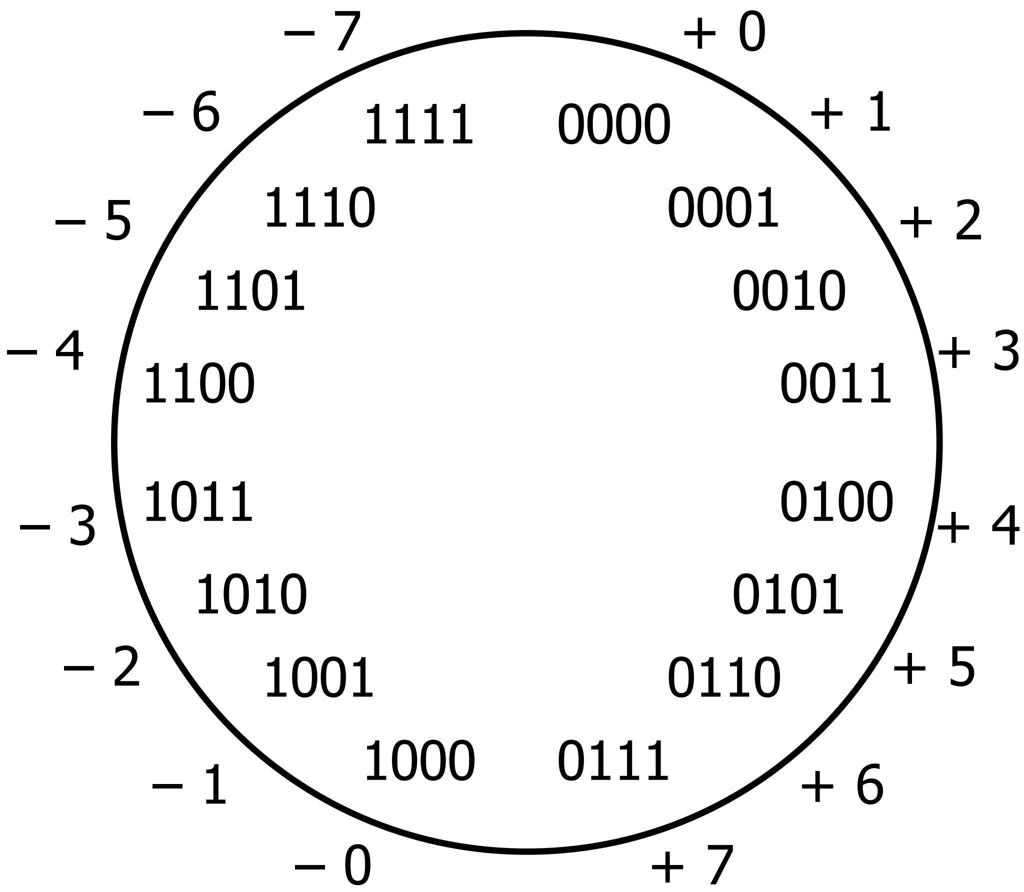


– 7
+ 0
– 6
1111
0000
+ 1
1110
0001
– 5
+ 2
1101
0010
– 4
+ 3
1100
0011
1011
0100
– 3
+ 4
1010
0101
– 2
+ 5
1001
0110
1000
0111
– 1
+ 6
– 0
+ 7

原码

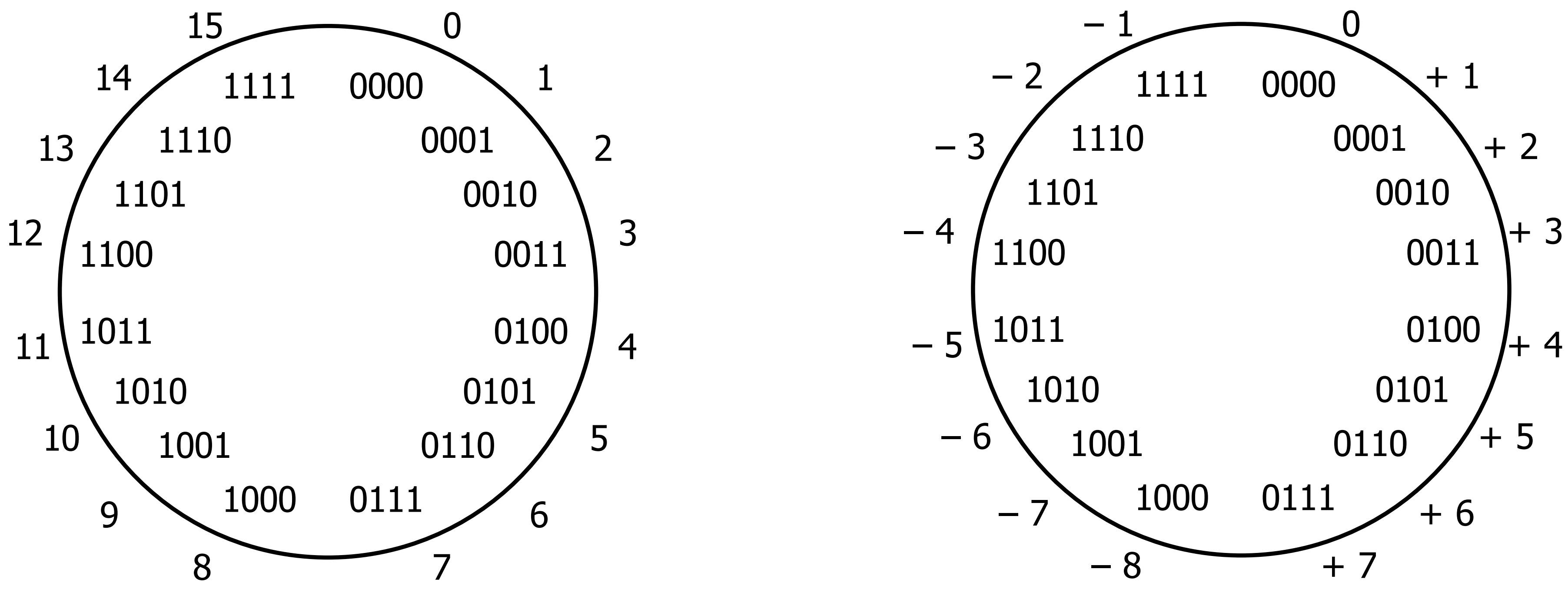
$$4 - 3 \;!= 4 + (-3)$$

$$\begin{array}{r} 0100 \\ +\,1011 \\ \hline \end{array}$$



补码

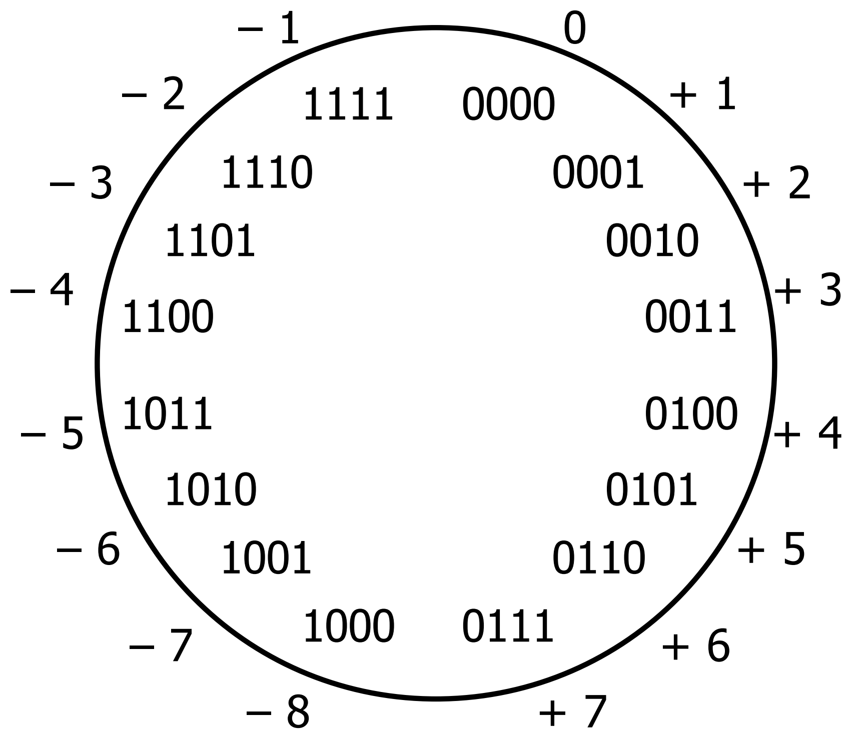
- 无符号 vs 有符号


15
0
14
1111
0000
1
13
1110
0001
2
12
1101
0010
3
1100
0011
11
1011
0100
4
1010
0101
10
1001
0110
5
9
1000
0111
6
8
7
– 1
0
– 2
+ 1
– 3
+ 2
– 4
+ 3
– 5
+ 4
– 6
+ 5
– 7
+ 6
– 8
+ 7

补码运算

2	0010	-2	1110
+ 3	+ 0011	+ -3	+ 1101

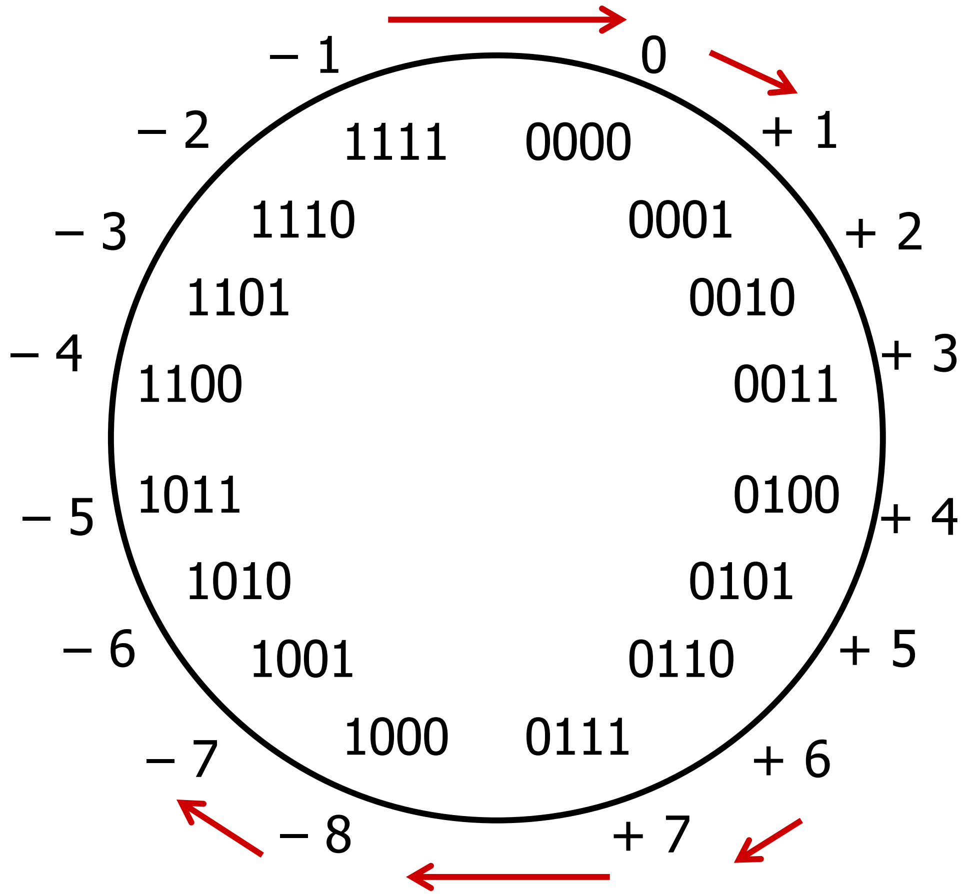
-2	1110	2	0010
+ 3	+ 0011	+ -3	+ 1101



补码运算

-1	1111
+ 2	+ 0010

6	0110
+ 3	+ 0011



思考

Argument$_1$	Op	Argument$_2$	Type	Result
`0`	`==`	`0U`	unsigned	1
`-1`	`<`	`0`	signed	1
`-1`	`<`	`0U`	unsigned	**0**
`2147483647`	`<`	`-2147483648`		
`2147483647U`	`<`	`-2147483648`		
`-1`	`<`	`-2`		
`(unsigned)-1`	`<`	`-2`		
`2147483647`	`<`	`2147483648U`		
`2147483647`	`<`	`(int)2147483648U`		

本讲提纲

- 定点加法原理
- 定点减法原理
- 溢出检测
- 运算器组成

本讲提纲

- 定点加法原理
- 定点减法原理
- 溢出检测
- 运算器组成

原码、反码、补码表示小结

正数的 原码、反码、补码表示均相同，

符号位为 0，数值位同数的真值。

零的原码和反码均有 2个编码，补码只 1个码

负数的 原码、反码、补码表示均不同，

符号位为 1，数值位：原码为数的绝对值

反码为每一位均取反码

补码为反码再在最低位+1

由 $[X]_{补}$ 求 $[-X]_{补}$：每一位取反后再在最低位+1

补码加减法

- 补码加法公式
  - $[x]_补 + [y]_补 = [x+y]_补 \pmod{2^{n+1}}$
- 补码减法公式
  - $[x-y]_补 = [x]_补 - [y]_补 = [x]_补 + [-y]_补 \pmod{2^{n+1}}$

例子：补码加法

- 例：x=+1001,y=+0101，求x+y

- 例：x=+1001,y=-0101，求x+y

例子：补码加法

- 例：x=+1001,y=+0101，求x+y
  - $[x]_{补}=01001$
  - $[y]_{补}=00101$
  - $[x+y]_{补}=01110$
  - $x+y=+1110$

- 例：x=+1001,y=-0101，求x+y
  - $[x]_{补}=01001$
  - $[y]_{补}=11011$
  - $[x+y]_{补}=\underline{1}00110=00110$
  - $x+y=+0110$

例子：补码加法

- 例：x=+1001,y=+0101，求x+y
  - $[x]_{补}=01001$
  - $[y]_{补}=00101$
  - $[x+y]_{补}=01110$
  - $x+y=+1110$

- 例：x=+1001,y=-0101，求x+y
  - $[x]_{补}=01001$
  - $[y]_{补}=11011$
  - $[x+y]_{补}=\underline{1}00110=00110$
  - $x+y=+0110$

补码加法的特点：

（1）符号位要当做数的一部分参加运算

（2）超过模2^{n+1}的进位要丢掉

例子：补码减法

- 例：x=+1101,y=+0110，求x-y

例子：补码减法

- 例：x=+1101,y=+0110，求x-y

$[x]_{补}=01101$

$[y]_{补}=00110$ $[-y]_{补}=11010$

例子：补码减法

- 例：x=+1101,y=+0110，求x-y

$[x]_{补}=01101$

$[y]_{补}=00110 \qquad [-y]_{补}=11010$

$$
\begin{array}{rr}
[x]_{补} & 01101 \\
+\ [-y]_{补} & 11010 \\
\hline
[x-y]_{补} & \boxed{1}\ 00111
\end{array}
$$

$$\boxed{x-y=+\ 0111}$$

本讲提纲

- 定点加法原理
- 定点减法原理
- 溢出检测
- 运算器组成
- 原码乘法原理与乘法器

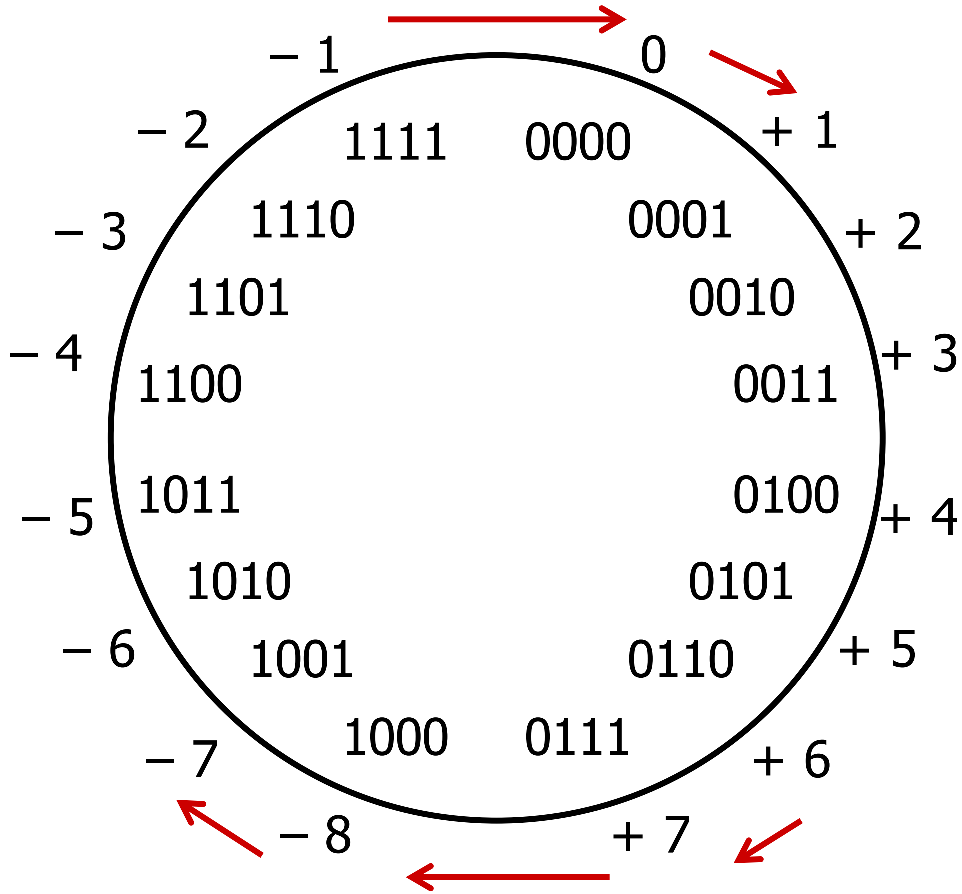
溢出概念和检测方法

- 溢出的概念
  - 定点整数机器中，数的表示范围$|x|<(2^{n-1})$；
  - 在运算过程中，运算结果超出机器字长所能表示的范围的现象，称为“溢出”。
- 可能产生溢出的情况
  - 两正数加，变负数，上溢（大于机器所能表示的最大数）
  - 两负数加，变正数，下溢（小于机器所能表示的最小数）

补码运算

$$\begin{array}{r} -1 \\ +\ 2 \\ \hline \end{array} \qquad \begin{array}{r} 1111 \\ +\ 0010 \\ \hline \end{array}$$

$$\begin{array}{r} 6 \\ +\ 3 \\ \hline \end{array} \qquad \begin{array}{r} {\color{red}0}110 \\ +\ {\color{red}0}011 \\ \hline \end{array}$$



例子：溢出

- 例： $x = +1011$, $y = +1001$,求 $x + y$

- 例： $x = -1101$, $y = -1011$,求 $x + y$

例子：溢出

- 例： $x = +1011,\ y = +1001$,求 $x + y$
  - $[x]_{补} = 01011$
  - $[y]_{补} = 01001$
  - $[x]_{补}+[y]_{补}=10100$

两正数加，变负数，正溢

- 例： $x = -1101,\ y = -1011$,求 $x + y$
  - $[x]_{补} = 10011$
  - $[y]_{补} = 10101$
  - $[x]_{补}+[y]_{补}=01000$

两负数加，变正数，负溢

溢出概念和检测方法

- 溢出检测方法一：双符号位法
  - 参与加减运算的数采用变形补码表示
  - 计算法则
    - 两个符号位都看做数码参加运算
    - 两数进行以模 2^{n+2}的加法，最高符号位上产生的进位丢掉

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

S_{f1}	S_{f2}	检测结果
0	0	正确（正数）
0	1	正溢
1	0	负溢
1	1	正确（负数）

例子：溢出检测

- 例： $x = +1100$, $y = +1000$,求 $x + y$
  - $[x]_{补} = 001100$
  - $[y]_{补} = 001000$
  - $[x]_{补}+[y]_{补}=010100$
- 例： $x = -1100$, $y = -1000$,求 $x + y$
  - $[x]_{补} = 110100$
  - $[y]_{补} = 111000$
  - $[x]_{补}+[y]_{补}=101100$

例子：溢出检测

- 例：$x = +1100,\ y = +1000$,求 $x + y$
  - $[x]_{补} = 001100$
  - $[y]_{补} = 001000$
  - $[x]_{补}+[y]_{补} = 010100$

正溢出

- 例：$x = -1100,\ y = -1000$,求 $x + y$
  - $[x]_{补} = 110100$
  - $[y]_{补} = 111000$
  - $[x]_{补}+[y]_{补} = 101100$

例子：溢出检测

- 例： $x = +1100,\ y = +1000$，求 $x + y$
  - $[x]_{补} = 001100$
  - $[y]_{补} = 001000$
  - $[x]_{补}+[y]_{补}=010100$

正溢出

- 例： $x = -1100,\ y = -1000$，求 $x + y$
  - $[x]_{补} = 110100$
  - $[y]_{补} = 111000$
  - $[x]_{补}+[y]_{补}=101100$

负溢出

溢出概念和检测方法

- 溢出检测方法二：单符号位法
  - 最高有效位产生进位而符号位无进位，正溢
  - 最高有效位无进位而符号位有进位，负溢

C_f	C_0	检测结果
0	**0**	**正确（正数）**
0	1	正溢
1	**0**	**负溢**
1	1	正确（负数）

其中C_f为符号位产生的进位,C_0为最高有效位产生的进位

例子：溢出检测

- 例： $x=+1011$, $y=+1001$,求 $x+y$
  - $[x]_{补}=01011$
  - $[y]_{补}=01001$
  - $[x]_{补}+[y]_{补}=10100$
- 例： $x=-1101$, $y=-1011$,求 $x+y$
  - $[x]_{补}=10011$
  - $[y]_{补}=10101$
  - $[x]_{补}+[y]_{补}=01000$

例子：溢出检测

- 例： $x = +1011$, $y = +1001$,求 $x + y$
  - $[x]_{补} = 01011$
  - $[y]_{补} = 01001$
  - $[x]_{补}+[y]_{补}=10100$

$C_f=0$，$C_0=1$，正溢

- 例： $x = -1101$, $y = -1011$,求 $x + y$
  - $[x]_{补} = 10011$
  - $[y]_{补} = 10101$
  - $[x]_{补}+[y]_{补}=01000$

例子：溢出检测

- 例： $x=+1011$, $y=+1001$,求 $x+y$
  - $[x]_补=01011$
  - $[y]_补=01001$
  - $[x]_补+[y]_补=10100$

$C_f=0$，$C_0=1$，正溢

- 例： $x=-1101$, $y=-1011$,求 $x+y$
  - $[x]_补=10011$
  - $[y]_补=10101$
  - $[x]_补+[y]_补=01000$

$C_f=1$，$C_0=0$，负溢

本讲提纲

- 定点加法原理
- 定点减法原理
- 溢出检测
- 运算器组成

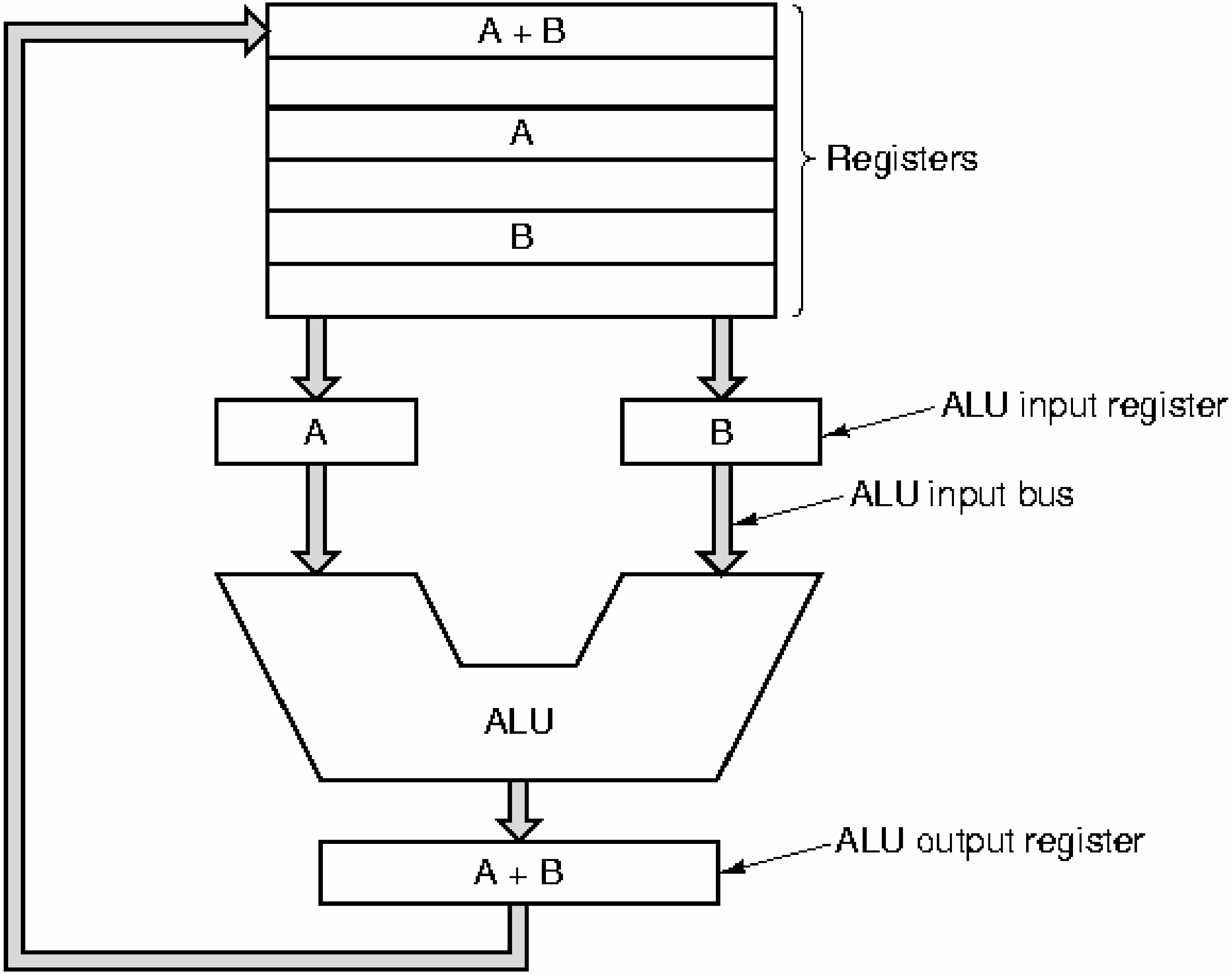
运算器的基本功能

- 完成算术、逻辑运算
  - +、-、×、÷、∧ 、∨、¬。
- 取得操作数
  - 寄存器组、数据总线
- 输出、存放运算结果
  - 寄存器组、数据总线
- 暂存运算的中间结果
  - 移位寄存器
- 得到运算结果的状态
- 理解、响应控制信号

运算器的基础逻辑电路

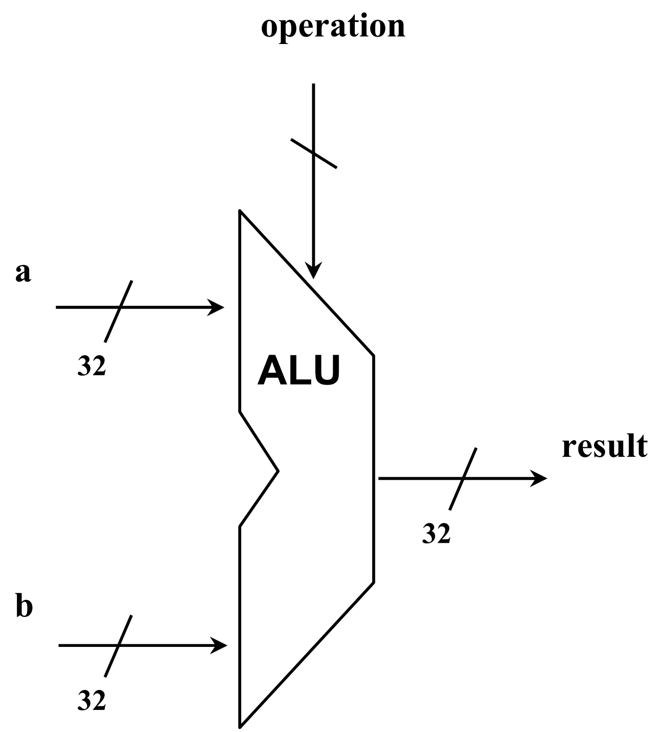
- 逻辑门电路
  - 完成逻辑运算
- 加法器
  - 完成加法运算
- 触发器
  - 保存数据
- 多路选择器、移位器
  - 选择、连通

数据通路


A + B
A
B
Registers
A
B
ALU input register
ALU input bus
ALU
A + B
ALU output register

ALU的功能和设计

- 功能
  - 对操作数完成算术逻辑运算
  - ADD、AND、OR
- 设计
  - 算术运算
    - 加法器
  - 逻辑运算
    - 与门、或门



复习逻辑运算和逻辑门

- 三种基本逻辑运算
  - 与运算 $Y = A \cdot B$
  - 或运算 $Y = A + B$
  - 非运算 $Y = \overline{A}$

表2.2.1 与逻辑真值表

输入		输出
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

表2.2.2 或逻辑真值表

输入		输出
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

表2.2.3 非逻辑真值表

A	Y
0	1
1	0

复习：逻辑运算和逻辑门

复合逻辑运算

- 与非运算 $Y=(AB)'$
- 或非运算 $Y=(A+B)'$
- 与或非运算 $Y=(AB+CD)'$
- 异或运算 $Y=A\oplus B=AB'+A'B$
- 同或运算 $Y=A\odot B=(A\oplus B)'=AB+A'B'$

表2.2.7 同或逻辑真值表

输入		输出
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

表2.2.4 与非逻辑真值表

输入		输出
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

表2.2.4 与非逻辑真值表

输入		输出
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

表2.2.5 或非逻辑真值表

输入		输出
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

表2.2.6 异或逻辑真值表

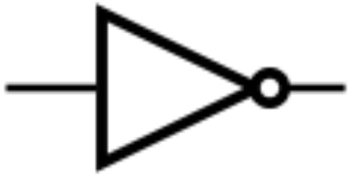
输入		输出
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

复习：逻辑运算和逻辑门

- 逻辑门图形符号

与

或



非

与非

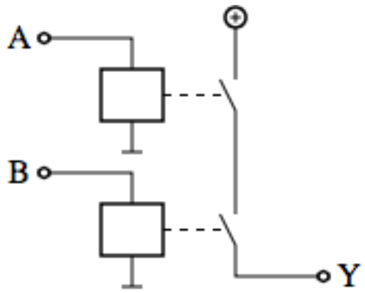
或非

异或

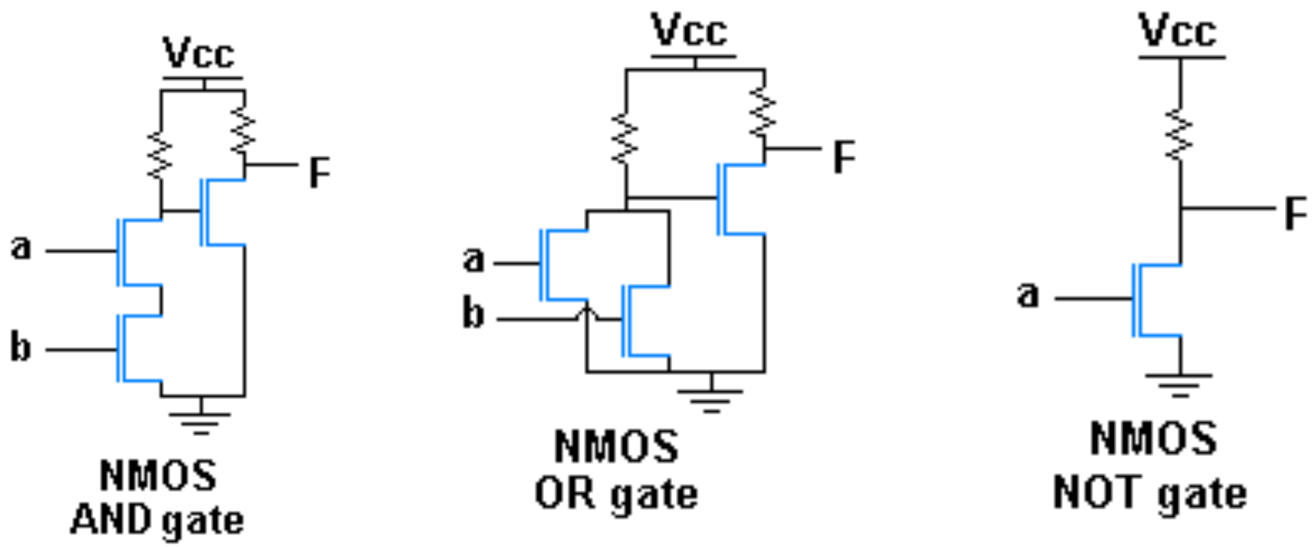
同或

复习：逻辑运算和逻辑门

继电器逻辑

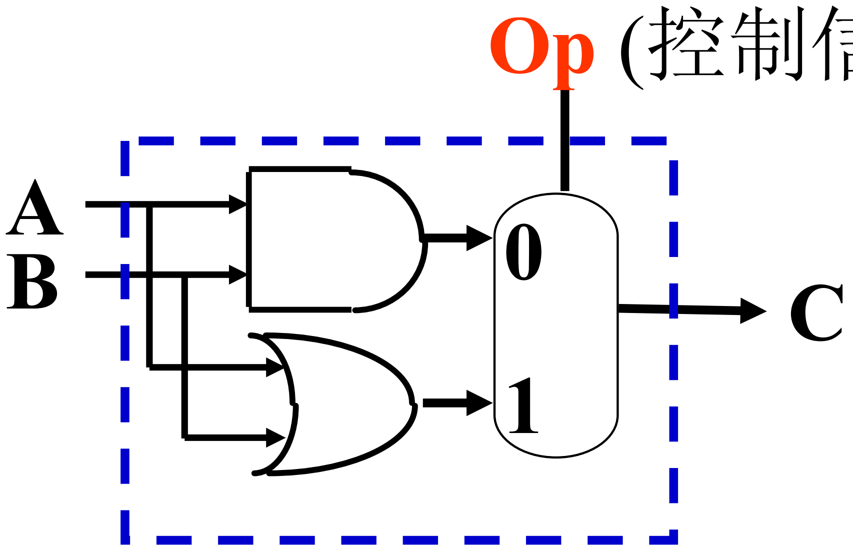


NMOS



1位ALU逻辑运算实现

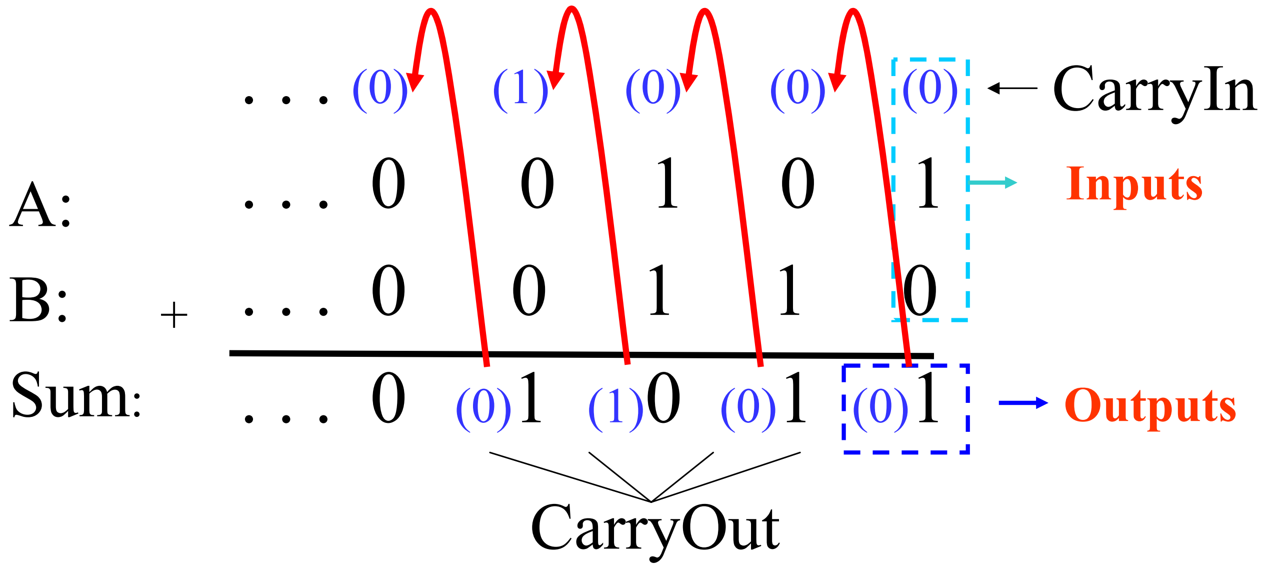
- 直接用逻辑门实现与和或的功能
- 用多路选择器,通过OP控制信号输出结果



功能表

Op	C
0	A and B
1	A or B

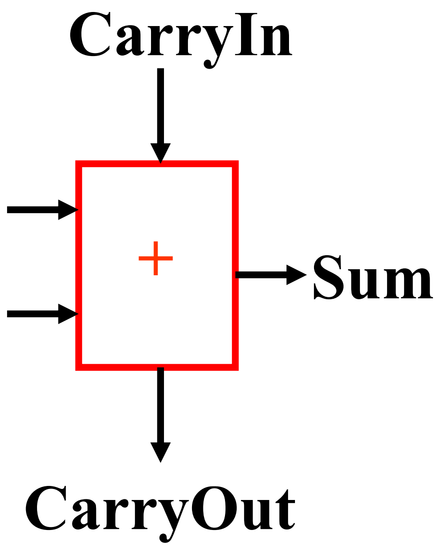
1位ALU加法运算实现



- 1位的加法:
  - 3个输入信号: A_i, B_i, $CarryIn_i$
  - 两个输出信号: Sum_i, $CarryOut_i$
    - ($CarryIn_{i+1} = CarryOut_i$)

1位ALU加法运算实现

Symbol



真值表

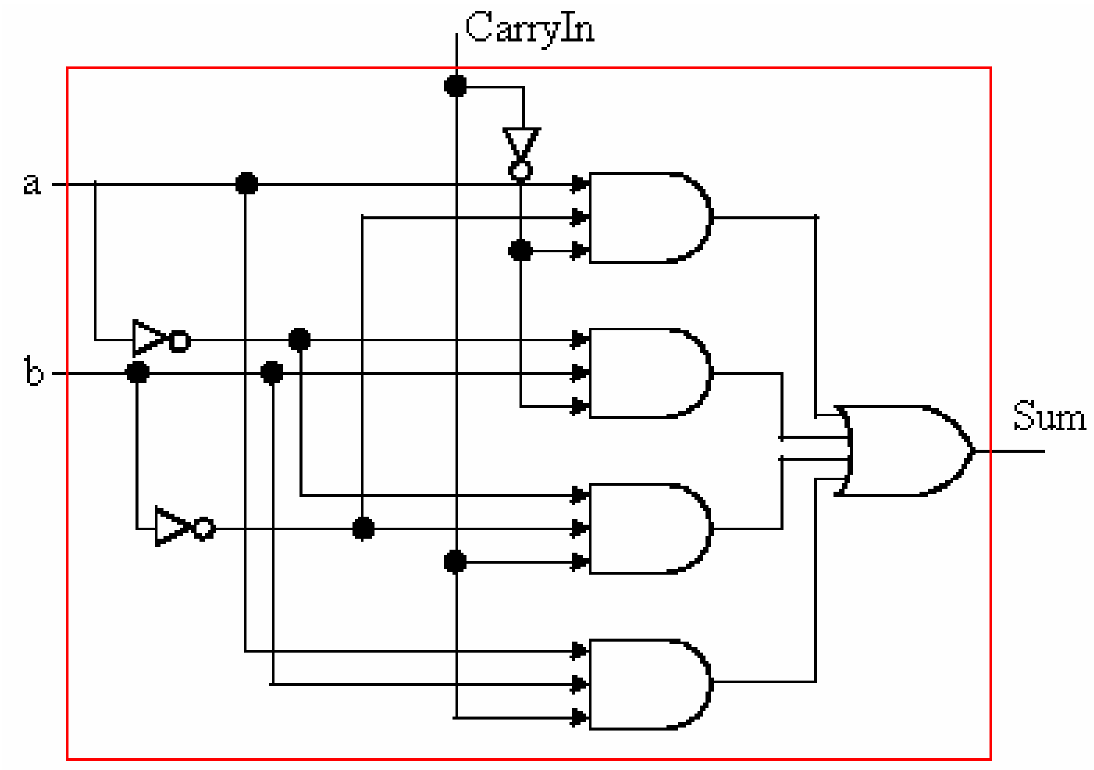
A	B	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

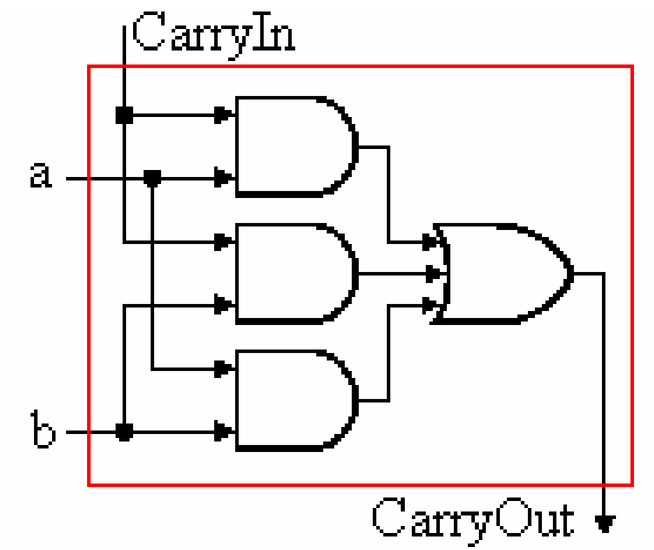
CarryOut = (A’*B*CarryIn) + (A*B’*CarryIn) + (A*B*CarryIn’) + (A*B*CarryIn) = (B*CarryIn) + (A*CarryIn) + (A*B)

Sum = (A’*B’*CarryIn) + (A’*B*CarryIn’) + (A*B’*CarryIn’) + (A*B*CarryIn)

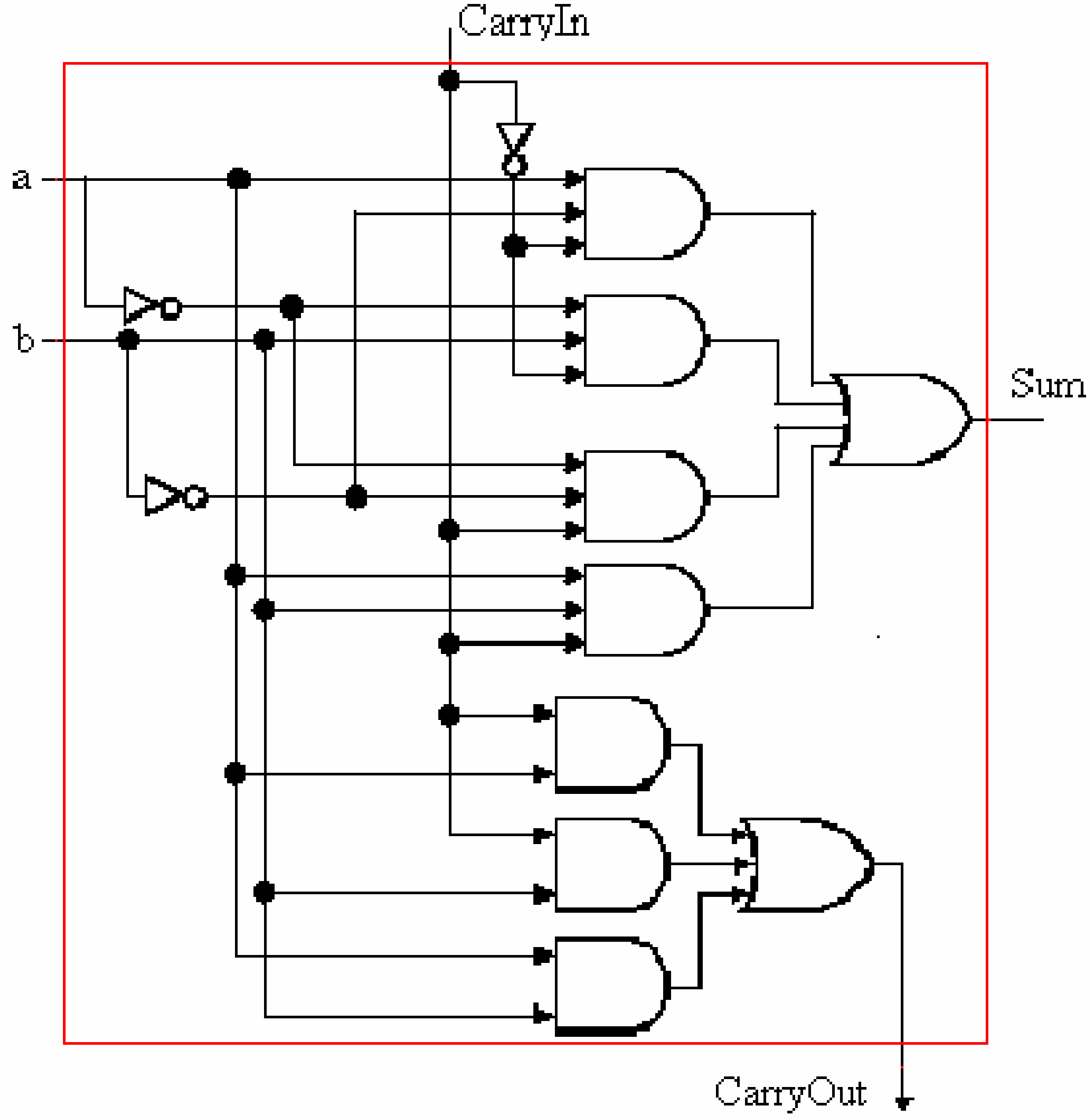
全加器设计与实现

- 用逻辑门实现加法,求Sum
- 用逻辑门求 CarryOut
- 将所有相同的输入连接在一起

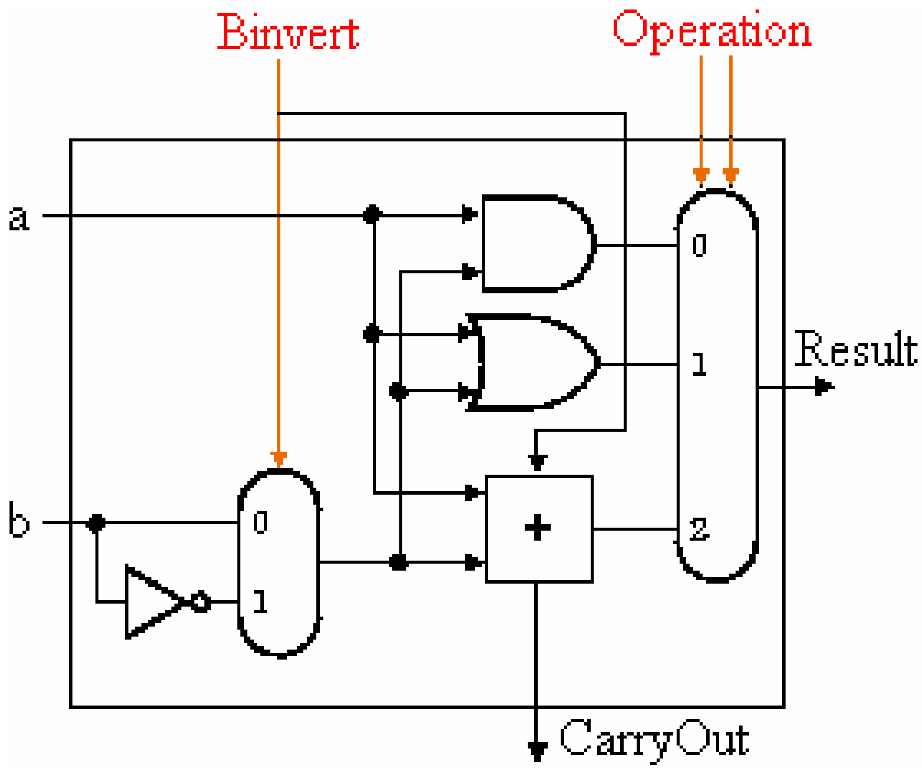




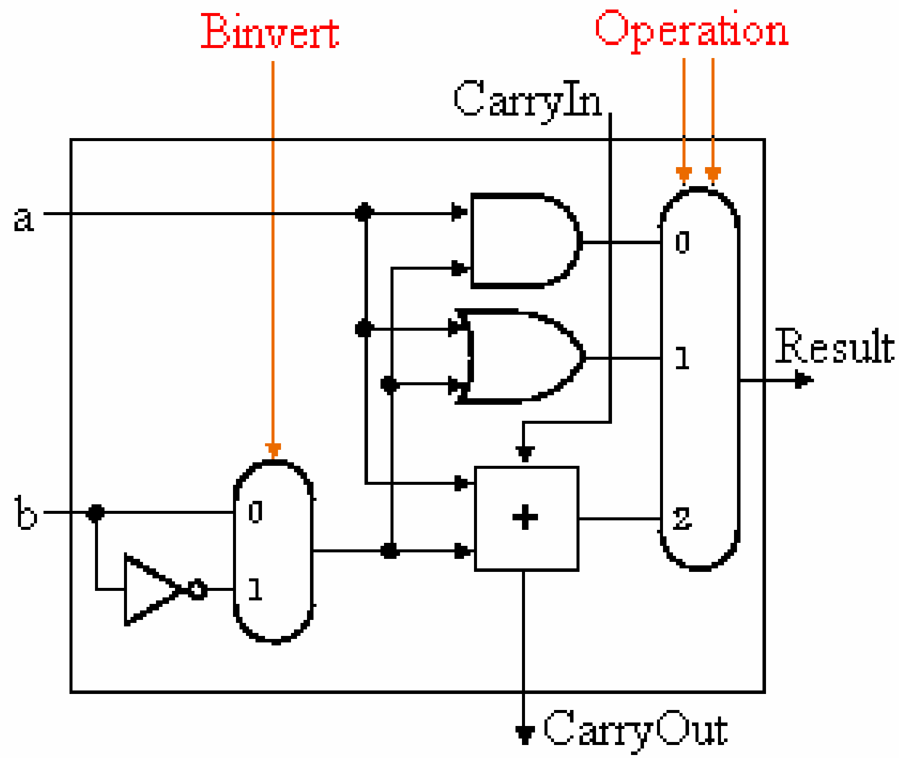
全加器



1位ALU



最低位



其他位

1位ALU的设计过程

- 确定ALU的功能
  - 与、或、加
- 确定ALU的输入参数
- 根据功能要求,得到真值表,并写出逻辑表达式;
- 根据逻辑表达式实现逻辑电路。
- 如何实现4位的ALU呢?

4位ALU实现方式

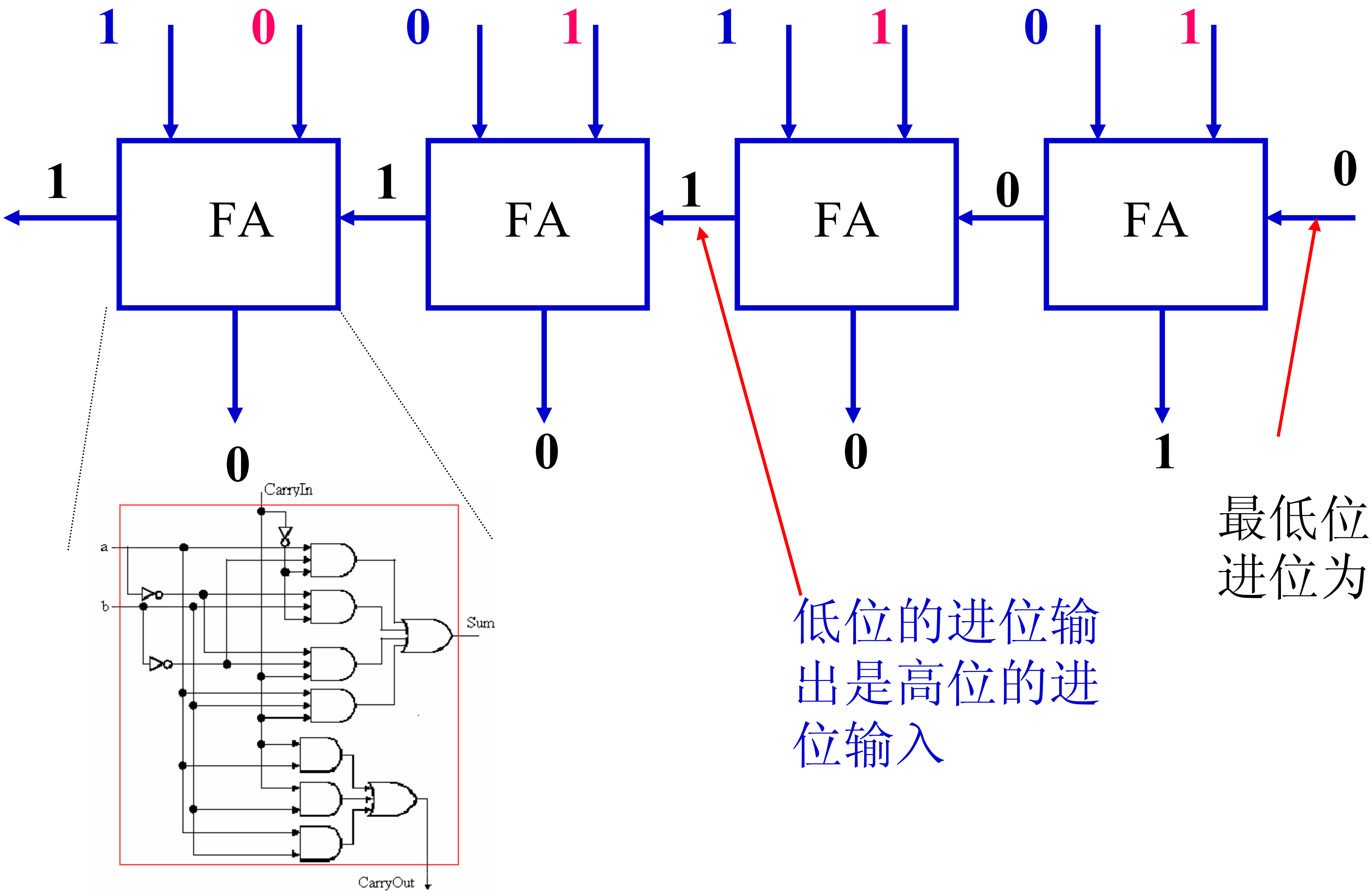
- 思路1:
  - 同1位ALU设计,写真值表,逻辑表达式,实现逻辑电路。

4位ALU实现方式

- 思路1:
  - 同1位ALU设计,写真值表,逻辑表达式,实现逻辑电路。
- 思路2:
  - 用1位ALU串联起来,得到4位的ALU。

$$
\begin{array}{cccccc}
 & & 1 & 1 & 0 & 0 \\
 & & 1 & 0 & 1 & 0 \\
 + & & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 1
\end{array}
$$

4位ALU设计



低位的进位输出是高位的进位输入

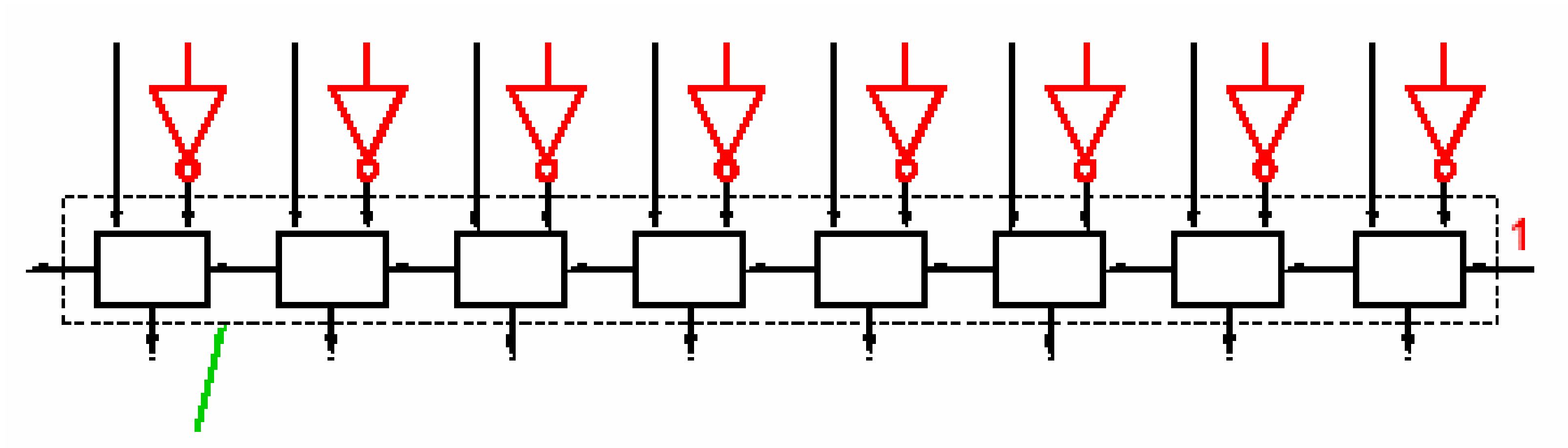
最低位进位为0

补码减法

- 根据算术运算规则:
- $[a-b]_{补}=[a]_{补}+[-b]_{补}$
- [-b]的补码为:将$[b]_{补}$的各位求反,并加1。
- 由此,我们可以用加法器实现减法。

补码减法

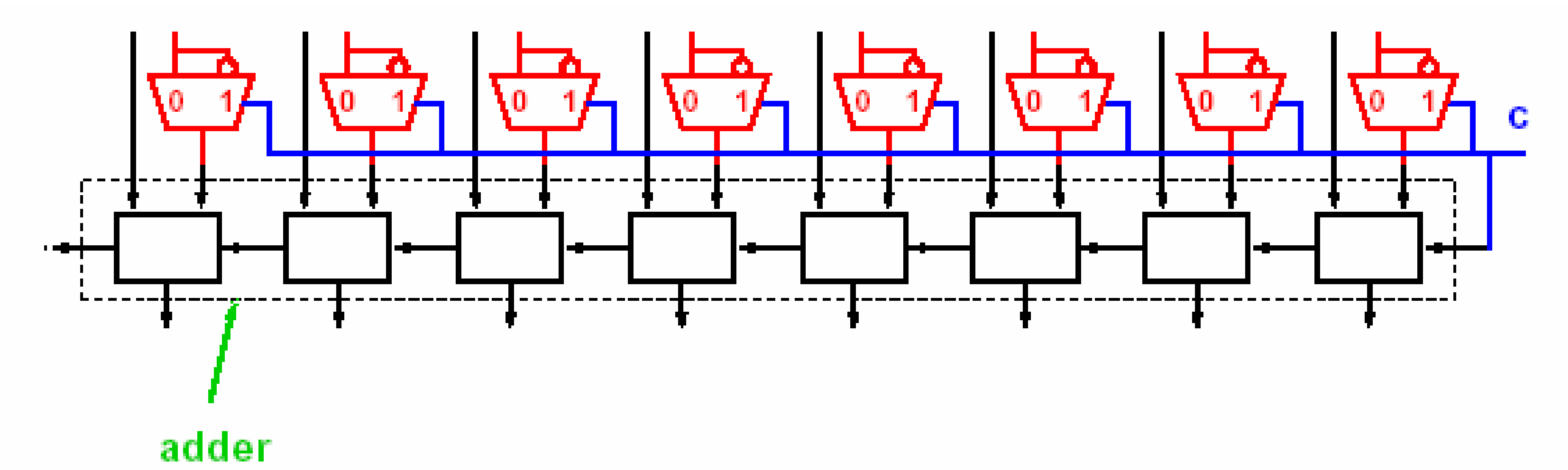
- 根据算术运算规则:
- [a-b]补=[a]补+[-b]补
- [-b]的补码为:将[b]补的各位求反,并加1。
- 由此,我们可以用加法器实现减法。



加法器

将加法和减法相结合

- 给定控制命令C=0,则ALU完成加法a+b;
- C=1,完成减法a-b



n位进位加法器（加法减法组合）