
数 字 逻 辑

丁 贤 庆

ahhfdxq@163.com

Home work (P38)

☞ 1.3.3 (2) (3)

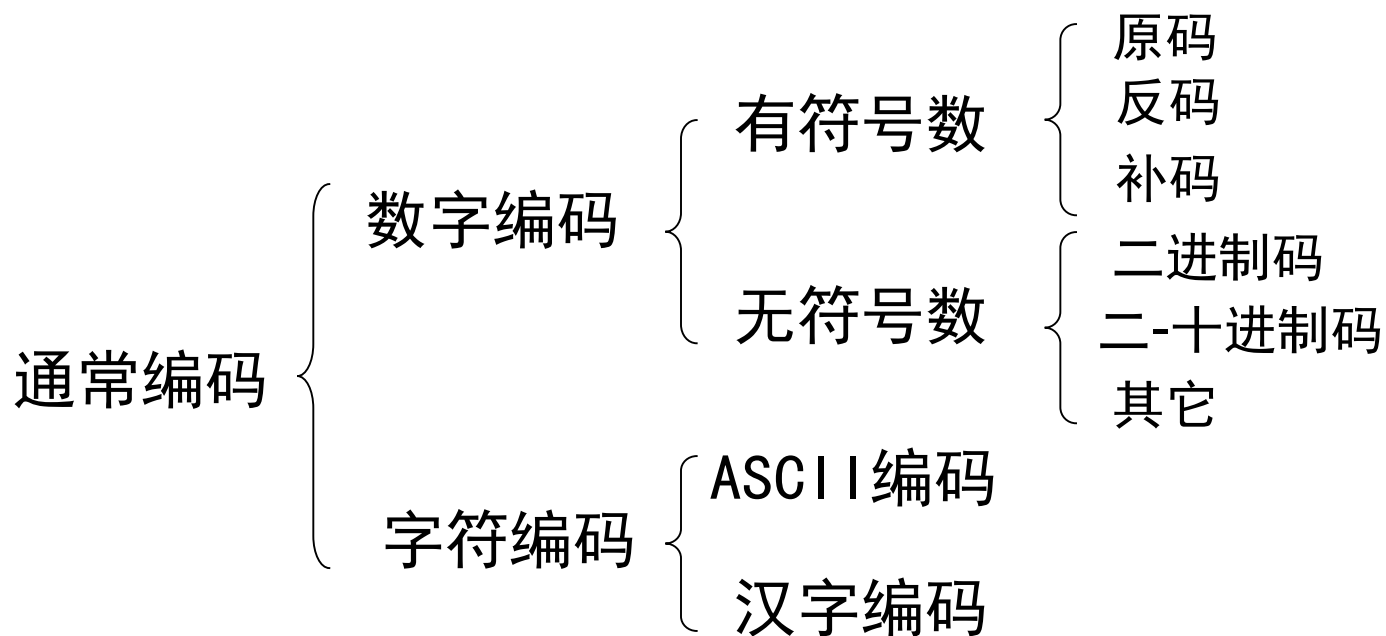
☞ 1.3.4 (2) (3)

☞ 1.4.1 (2) (3)

☞ 1.4.3 (2) (3)

1. 4 二进制编码

给一个信息或符号指定一个具体的二进制码去代表它，这一过程称为二进制编码



有符号数的表示

有符号数采用“符号位+数值位”的形式表示。

符号位(Sign Bit)为“0”时表示正数，为“1”时表示负数，而数值位表示数值的大小。

有符号数有原码、反码和补码三种表示方法。

原码表示：直接在数值位的前面加上一个bit符号位

一般情况下，最高位是符号位，其他是数值位。

有符号数（原码）的表示 (true form)

➤ Signed-Magnitude Representation

Take 8-bit numbers for example:

$$A = \textcolor{red}{0}101\ 1101_2 = +93_{10}$$

$$-A = \textcolor{red}{1}101\ 1101_2 = -93_{10}$$

$$B = \textcolor{red}{1}110\ 1011_2 = -107_{10}$$

$$-B = \textcolor{red}{0}110\ 1011_2 = +107_{10}$$

上面的原码中，红色的bit位为符号位，符号位后面的都是数值位。

Advantage of Signed-Magnitude :

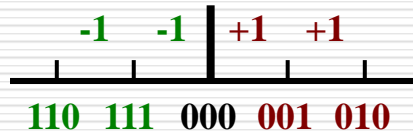
- Negating a number is very easy.
 - Just need to change its sign.

有符号数（原码）的表示

➤ Signed-Magnitude Representation

Usually, the MSB is used as the sign bit. **0: plus, 1: minus**

0:正号 1: 负号



Take 4-bit number for example:

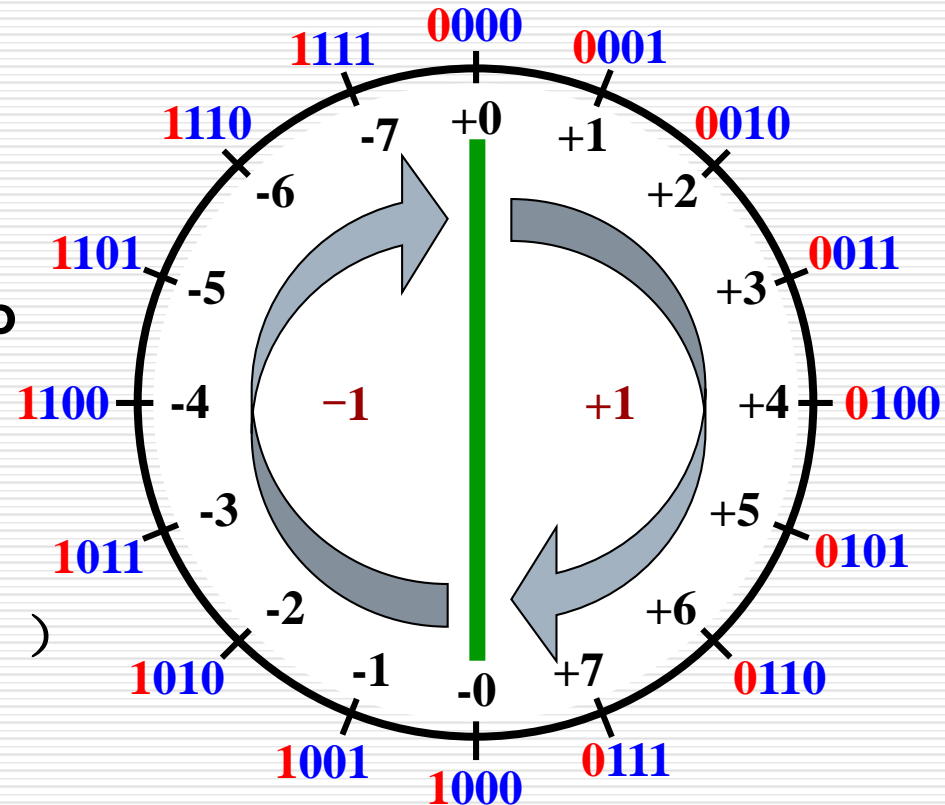
以4位bit数为例

A signed-magnitude number is also called a **True Code** (原码).

It has an equal number of positive and negative integers. (直接在数值位的前面加上一个bit符号位。)

An n -bit integer range is :

$$-(2^{n-1} - 1) \sim +(2^{n-1} - 1)$$



图中红色的bit位为符号位

正数的原码、反码和补码形式相同。

对于“符号位+n-1位数值位”构成的 n 位有符号二进制数 N ，原码能够表示数的范围为 $-(2^{n-1}-1) \sim +(2^{n-1}-1)$ 。

“0”的表示方式有两种： $+0$ 和 -0 。

反码

反码中**符号位**为“0”时表示**正数**，为“1”时表示**负数**。

用反码表示有符号数时，符号位保持不变，数值大小定义为：

$$(N)_{\text{反码}} = \begin{cases} N & (\text{正数时}) \\ (2^n - 1) - N & (\text{负数时}) \end{cases}$$

负数的反码，数值位部分可以看作是原码的数值位部分按位取反得到的。

$$(-6) = (10000110)_{\text{原}} = (11111001)_{\text{反}}$$

$$(+6) = (00000110)_{\text{原}} = (00000110)_{\text{反}}$$

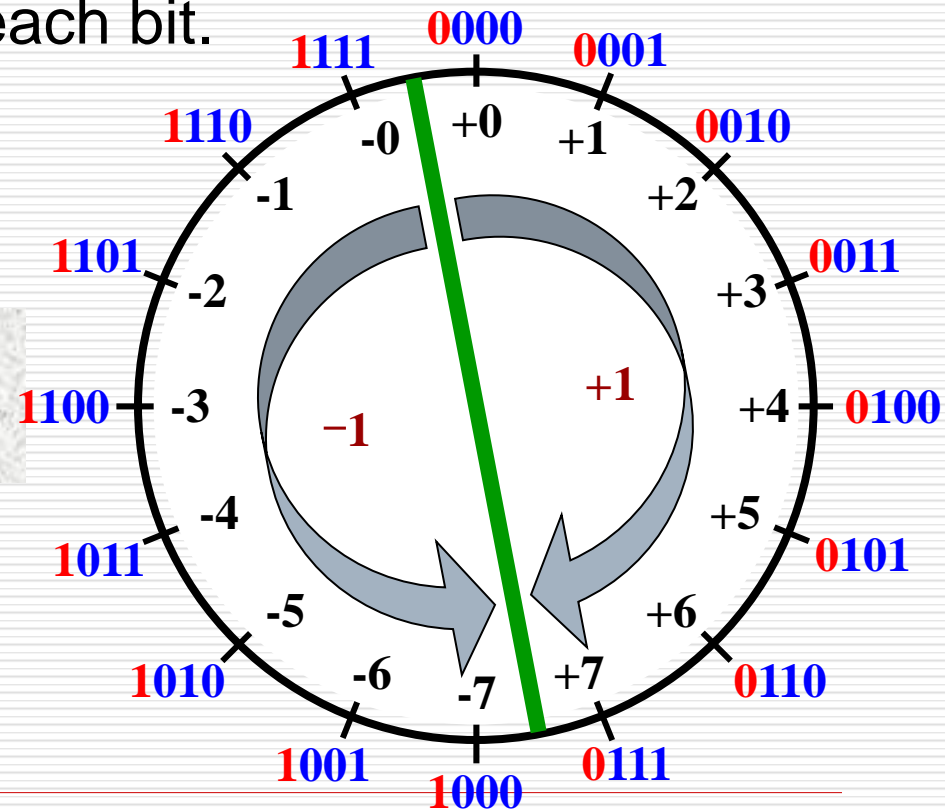
反码

➤ Ones'-Complement Representation —— Bitwise Inversion (按位求反)

Negate a number by inverting each bit.

Take 4-bit number for example:
(以4bit 位数为例)

Can we use *Ones' complement* to represent negative integers?



反码

➤ Ones'-Complement Representation —— Bitwise Inversion (按位求反)

Negate a number by inverting each bit.

Take 4-bit number for example:

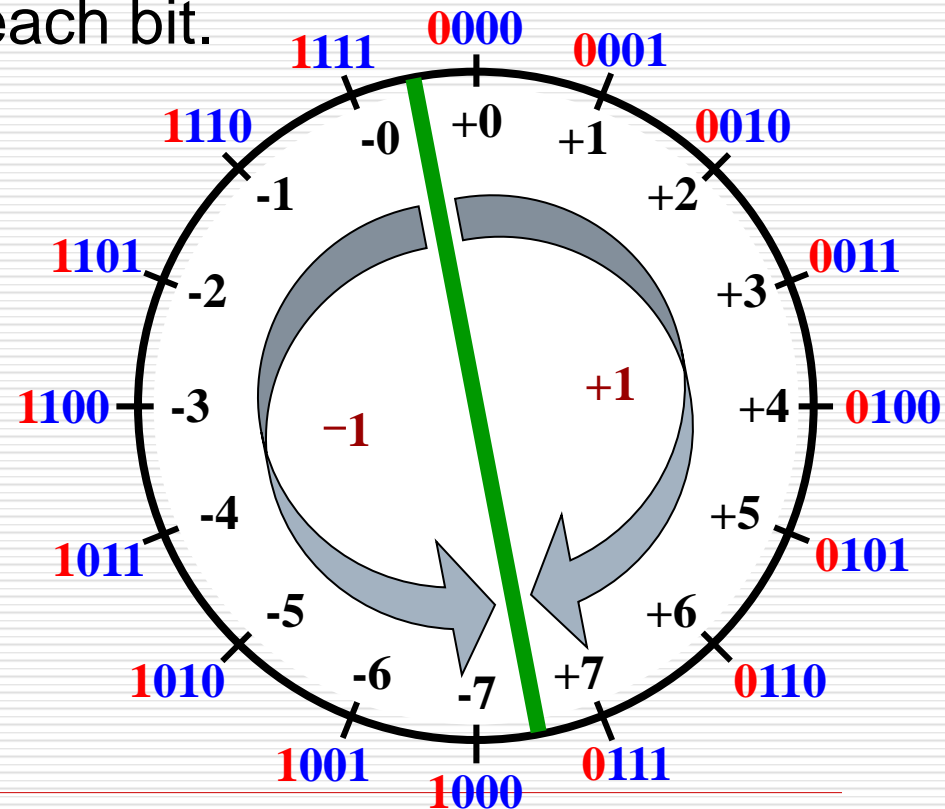
(以4bit 位数为例)

It has an equal number of positive and negative integers.

The MSB can also be used as the sign bit.
0 = plus , 1 = minus

An n-bit integer range is :

$$-(2^{n-1} - 1) \sim +(2^{n-1} - 1)$$



反码计算

Take 8-bit numbers for example: (以8bit 位数为例)

$$(89)_{\text{反}} = 0101\ 1001_2 \quad (-93)_{\text{反}} = 1010\ 0010_2$$

$$(-4)_{\text{反}} = 1111\ 1011_2 \quad (-75)_{\text{反}} = 1011\ 0100_2$$

If $A - B = A + (-B)$?

Example of Addition:

$$89 - 75 = ?$$

$$89 + (-75) = ?$$

Wrong!

0	0	0	0	0	0	0	0	Carry
	0	1	0	1	1	0	0	1
								+89
+	1	0	1	0	0	0	1	0
								-93
	1	1	1	1	1	0	1	1
								-4

Correct!

1	1	1	1	0	0	0	0	Carry
	0	1	0	1	1	0	0	1
								+89
+	1	0	1	1	0	1	0	0
								-75
	0	0	0	0	1	1	0	1
								+13
+							1	
	0	0	0	0	1	1	1	0
								+14

Correct!

If there is a carry out of the MSB, add 1 to the result.

Subtractor is not necessary !

补码 (complement)

正数的补码，与正数的原码、反码形式相同。

Negate a number by inverting each bit. then add 1;

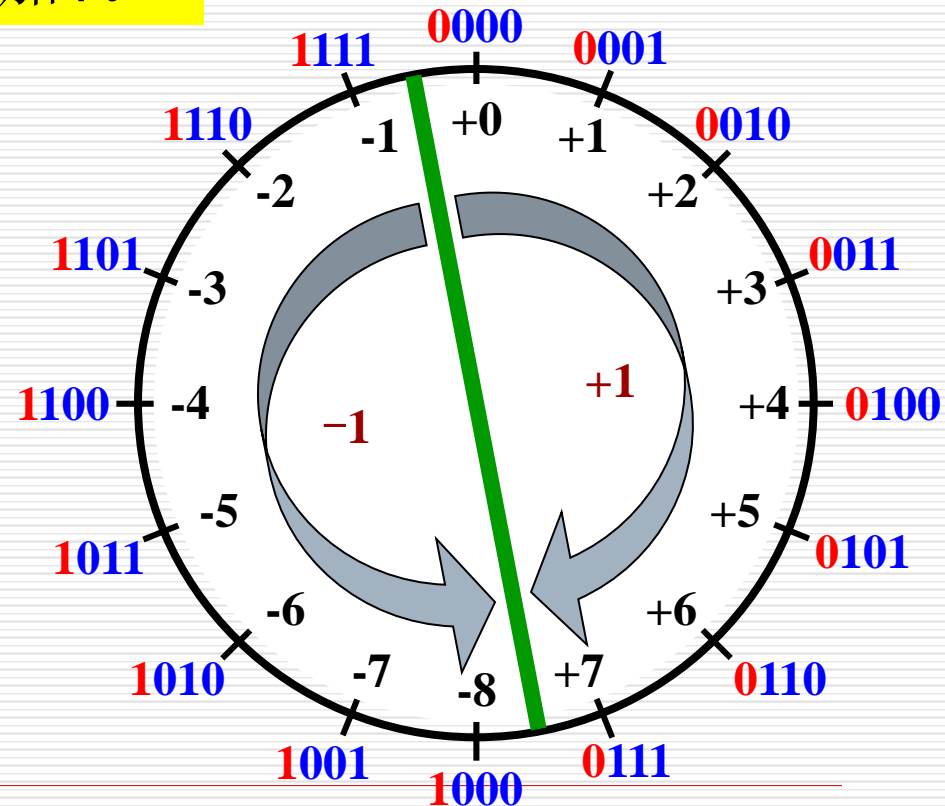
负数的补码就是负数的反码部分加1。

Take 4-bit number for example:

(以4bit 位数为例)

Can we use two's complement to represent negative integers?

$$((N)_{\text{补码}})_{\text{补码}} = (N)_{\text{原码}}$$



补码

用电路实现减法运算较复杂（需先比较大小才能减），而实现加法相对容易，为了将减法运算转换为加法运算，引入了补码。下面以手表为例说明补码运算的原理。

早上7点起床发现手表停在11点。调表的方法有两种（如图1-5）：

- ◆ 第一种方法(减法)：回拨4格，即 $11-4=7$ ；
- ◆ 第二种方法(加法)：向前拨8格，即 $11+8=(12)+7$ ；

说明：在忽略进位的情况下，可以用加法运算代替减法运算。

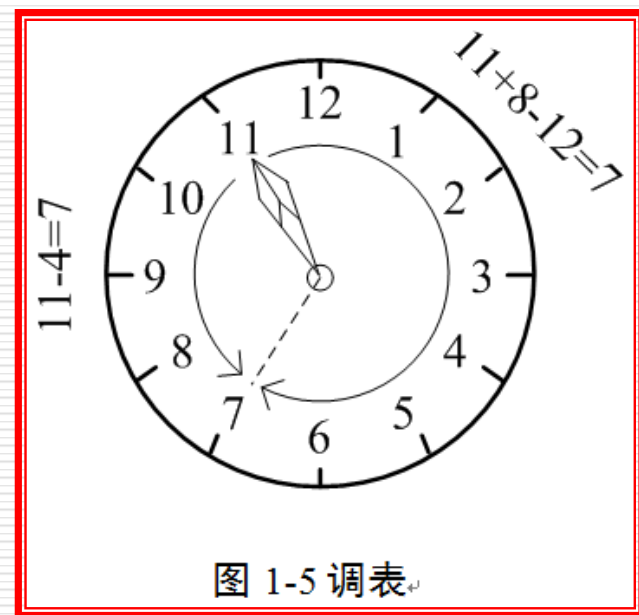


图 1-5 调表

问题：如何从4找到8？

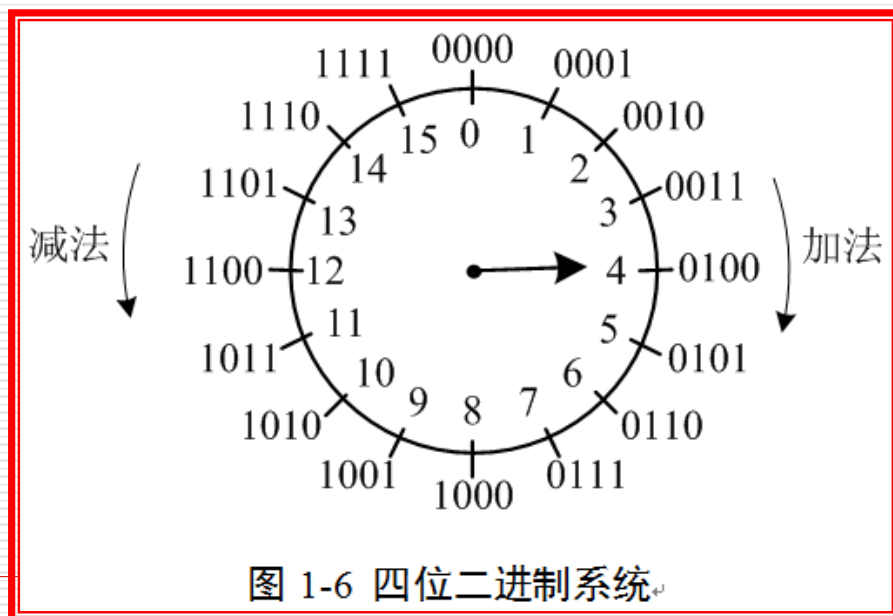
$4+8=12$ ，恰好为**表盘的模**（也称为**进制**、**容量**）。对于模12来说，8为4的**补码**。

对于二进制系统也是同样的道理。以**四位二进制（模16）**系统为例，如图1-6所示，若要做减法运算 $1011-0111$ （十进制**11-7**）时，首先应找到

0111的补码。因为

$7+9=16$ ，所以

$1011-0111$ 可以用**加法运算** $1011+1001$ （十进制**11+9**）代替，即对于模16运算，1001为0111的**补码**。



补码

补码又称为**对2的补码**（2's complement）。
用补码表示有符号数时，符号位保持不变，数值大小定义为

$$(N)_{\text{补码}} = \begin{cases} N & (\text{正数时}) \\ 2^n - N & (\text{负数时}) \end{cases}$$

正数的原码、反码和补码形式相同。

求负数的补码，一般方法是：先求出负数的反码（将原码的数值位逐位求反），然后在最低位加1即可得到补码，即

$$(N)_{\text{补码}} = (N)_{\text{反码}} + 1 \quad (\text{负数时})$$

问题：怎样由补码得到原码？

$$((N)_{\text{补码}})_{\text{补码}} = (N)_{\text{原码}}$$

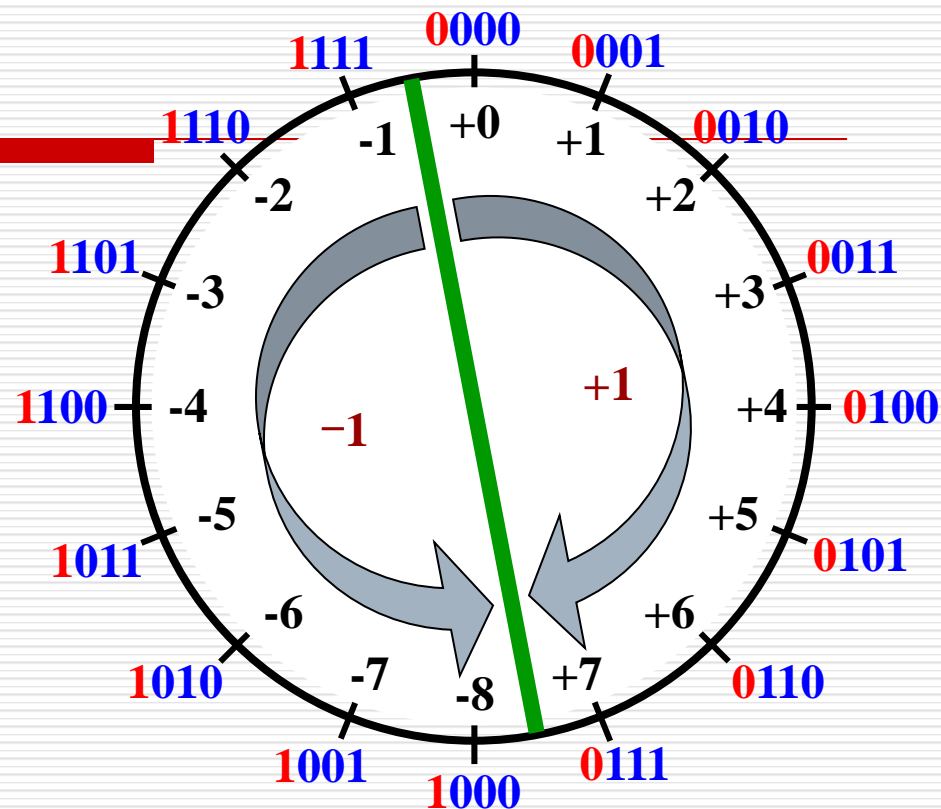
补码

Take 4-bit number for example:

(以4bit 位数为例)

The MSB can also be used as the sign bit.
0 = plus , 1 = minus

An n-bit integer range is :
 $-(2^{n-1}) \sim +(2^{n-1} - 1)$



若 $(N)_{\text{补码}} = 1000$ 求 $(N)_{\text{原码}} = ?$

解答：由于符号位为1，所以N必为负数。其数值位的求法如下：
将数值位000对应位求反，然后加1，得到数值部分值为1000，也就是8，所以1000是-8的补码

例 用**二进制补码**计算13+10、13-10、-13+10和-13-10。

解：由于13+10=23，故数值大小需要用5位二进制数表示。用补码运算时，需要再加上1位符号位，所以需要**用6位有符号二进制数**运算。

$$\begin{array}{r} +13 \quad 0 \ 01101 \\ +10 \quad 0 \ 01010 \\ \hline +23 \quad 0 \ 10111 \end{array}$$

$$\begin{array}{r} -13 \quad 1 \ 10011 \\ +10 \quad 0 \ 01010 \\ \hline -3 \quad 1 \ 11101 \end{array}$$

$$\begin{array}{r} +13 \quad 0 \ 01101 \\ -10 \quad 1 \ 10110 \\ \hline +3 \quad (1) \ 0 \ 00011 \end{array}$$

$$\begin{array}{r} -13 \quad 1 \ 10011 \\ -10 \quad 1 \ 10110 \\ \hline -23 \quad (1) \ 1 \ 01001 \end{array}$$

应用补码可以将减法转化成加法，而**乘法**可以用移位相加实现，**除法**运算可以用移位相减实现，故**加、减、乘、除**，全部可以用**移位和相加**这两种操作实现，简化了电路结构。

补码

6位有符号二进制数系统里：

Example of Addition:

有了补码就可以很容易的实现正数与负数的运算了。

0	0	0	1	0	0	Carry
	0	0	0	1	0	+2
+	0	0	0	1	1	+3
<hr/>						
	0	0	1	0	1	+5

Correct!

1	1	1	1	1	0	Carry
	1	1	0	1	1	-5
+	1	1	1	0	1	-3
<hr/>						
	1	1	0	0	0	-8

Correct!

0	0	0	0	1	0	Carry
	0	0	1	0	1	+5
+	1	1	0	0	1	-7
<hr/>						
	1	1	1	1	0	-2

Correct!

A negative integer is just the **Complement** of its positive integer !

$$A - B = A + \text{Module} - B = A + (-B)$$

∴ Subtractor is not necessary !

8位二进制数与无符号数和有符号数三种表示方法对照表

8位二进制数	无符号数	有符号数		
		原码	反码	补码
0000_0000	0	+0	+0	+0
0000_0001	1	+1	+1	+1
...
0111_1101	125	+125	+125	+125
0111_1110	126	+126	+126	+126
0111_1111	127	+127	+127	+127
1000_0000	128	-0	-127	-128
1000_0001	129	-1	-126	-127
1000_0010	130	-2	-125	-126
...
1111_1110	254	-126	-1	-2
1111_1111	255	-127	-0	-1

1.4.1 二—十进制码进制码

（BCD码——Binary Code Decimal）用4位二进制数来表示一位十进制数中的0~9十个数码。

4位二进制数有16种组合，可从这16种组合中选择10种组合分别来表示十进制的0~9十个数。选哪10种组合，有多种方案，这就形成了不同的BCD码。

几种常见的BCD码

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

十进制数	有权码			无权码	
	8421码	2421码	5421码	余3码	余3循环码
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	1011	1000	1000	1100
6	0110	1100	1001	1001	1101
7	0111	1101	1010	1010	1111
8	1000	1110	1011	1011	1110
9	1001	1111	1100	1100	1010

几种常见的BCD码

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

十进制数	有权码			无权码	
	8421码	2421码	5421码	余3码	余3循环码
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	1011	1000	1000	1100
6	0110	1100	1001	1001	1101
7	0111	1101	1010	1010	1111
8	1000	1110	1011	1011	1110
9	1001	1111	1100	1100	1010

几种常见的BCD码

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

十进制数	有权码			无权码	
	8421码	2421码	5421码	余3码	余3循环码
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	1011	1000	1000	1100
6	0110	1100	1001	1001	1101
7	0111	1101	1010	1010	1111
8	1000	1110	1011	1011	1110
9	1001	1111	1100	1100	1010

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

几种常见的BCD码

十进制数	有权码			无权码	
	8421码	2421码	5421码	余3码	余3循环码
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	1011	1000	1000	1100
6	0110	1100	1001	1001	1101
7	0111	1101	1010	1010	1111
8	1000	1110	1011	1011	1110
9	1001	1111	1100	1100	1010

各种编码的特点

- ◆ 有权码：编码与所表示的十进制数之间的转换容易。

$$(1001\ 0000)_{8421\text{BCD}} = (90)_D$$

- ◆ 余3码：0和9,1和8.....6和4的余3码互为反码，这对于求取对10的补码很方便。如将两个余3码相加的和是十进制的10时，正好是二进制的16，于是可从高位自动产生进位信号。
- ◆ 余3码循环码：相邻的两个代码之间仅一位的状态不同。按余3码循环码组成计数器时，每次转换过程只有一个触发器翻转，译码时不会发生竞争一冒险现象。

用BCD代码表示十进制数

对于一个多位的十进制数，需要有与十进制位数相同的几组BCD代码来表示。例如：

$$\begin{aligned}(463.5)_{10} &= \left[\begin{array}{ccc} \text{0100} & \text{0110} & \text{0011} \\ 4 & 6 & 3 \end{array} \right]_{8421\text{BCD}} \\ (863.2)_{10} &= \left[\begin{array}{ccc} \text{1110} & \text{1100} & \text{0011} \\ 8 & 6 & 3 \end{array} \right]_{2421\text{BCD}}\end{aligned}$$

不能省略！

不能省略！

求BCD代码表示的十进制数

对于有权BCD码，可以根据位权展开求得所代表的十进制数。例如：

$$(0111)_{8421\text{BCD}} = 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = (7)_{10}$$

$$(1101)_{2421\text{BCD}} = 1 \times 2 + 1 \times 4 + 0 \times 2 + 1 \times 1 = (7)_{10}$$

1.4.2 格雷码

3 位二进制码与格雷码的对照表

□ 格雷码是一种常见的无权码。

□ 特点是：任何两个相邻代码之间仅有一位取值不同。

□ 其首、尾两个代码之间也有一位不同，因此，格雷码又称为循环码。

十进制数	二进制码 $b_2b_1b_0$	格雷码 $G_2G_1G_0$
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

1.4.2 格雷码

3 位二进制码与格雷码的对照表

□ 格雷码另一个特点：

最高位的0和1只改变一次；若以最高位的0和1之间的交界为轴，其他位的代码是上下对称的。所以，格雷码又是反射码。

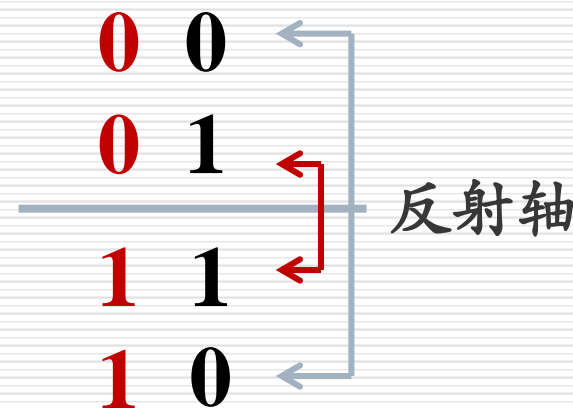
十进制数	二进制码 $b_2b_1b_0$	格雷码 $G_2G_1G_0$
0	000	000
1	001	001
2	010	011
3	011	<u>010</u>
4	100	110
5	101	111
6	110	101
7	111	100

1.4.2 格雷码

□ 构造格雷码的方法一：

2 位格雷码

- 一位格雷码有两个代码：0 和 1。
- $(n+1)$ 位格雷码中的前 2^n 个代码是将 n 位格雷码，按顺序排列，最高位补0。
- $(n+1)$ 位格雷码中的后 2^n 个代码是将 n 位格雷码，按倒序排列，最高位补1。



1.4.2 格雷码

□ 格雷码构成方法

- 一位格雷码有两个代码：0 和 1。
- $(n+1)$ 位格雷码中的前 2^n 个代码是将 n 位格雷码，按顺序排列，最高位补0。
- $(n+1)$ 位格雷码中的后 2^n 个代码是将 n 位格雷码，按倒序排列，最高位补1。

3 位格雷码

0	0	0	反射轴
0	0	1	
0	1	1	反射轴
0	1	0	
1	1	0	
1	1	1	
1	0	1	
1	0	0	

4 位二进制码与格雷码的对照表

二进制码 $b_3b_2b_1b_0$	格雷码 $G_3G_2G_1G_0$
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Gray Code

格雷码

Gray Code Constructing

构造格雷码-----异或运算实现

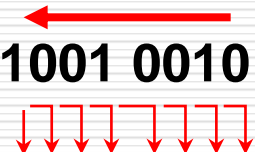
From binary number to Gray code

The width is same, the MSB is same;


From **right to left**, if **a bit in binary number is same as its right bit**, the **gray bit is 0**. If it is different, **the gray bit is 1**.

Examples:

Binary number: 1001 0010
Gray codes: 1101 1011



二进制码 1001 0010
格雷码 1101 1011



Gray Code

Translation between Gray Code and Binary Code

Binary code: $b_{n-1} b_{n-2} b_{n-3} \dots b_1 b_0$

Gray code: $g_{n-1} g_{n-2} g_{n-3} \dots g_1 g_0$

$$g_i = b_{i+1} \oplus b_i \quad g_{n-1} = b_{n-1} \oplus 0 = b_{n-1} \quad (n-1 \text{ 是最高位})$$

5的二进制码是: 0101 \rightarrow 5的格雷码是: 0111

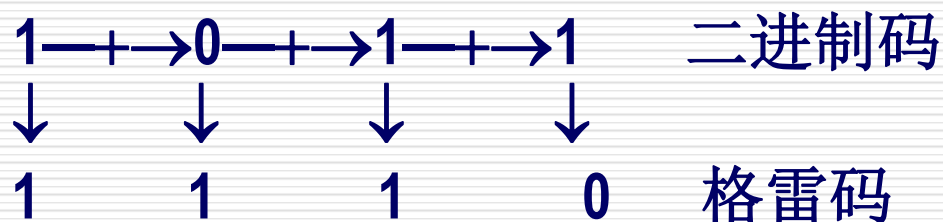
$$\because A = B \oplus C \rightarrow C = A \oplus B \rightarrow B = A \oplus C$$

$$\therefore b_i = b_{i+1} \oplus g_i \quad b_{n-1} = g_{n-1} \oplus 0 = g_{n-1} \quad (n-1 \text{ 是最高位})$$

7的格雷码是: 0100 \rightarrow 7的二进制码是: 0111

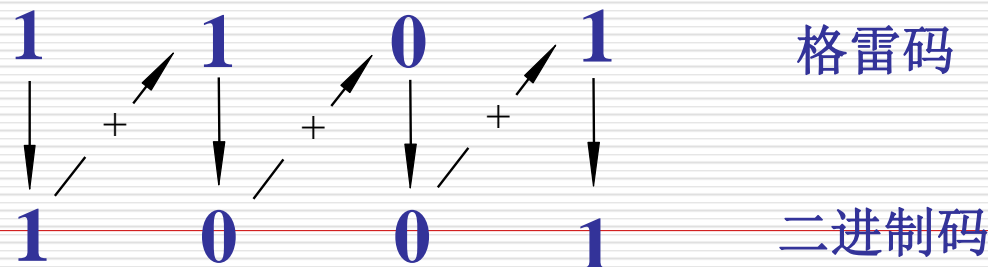
□ 二进制码到格雷码的转换

- (1) 格雷码的最高位（最左边）与二进制码的最高位相同。
- (2) 从左到右，逐一将二进制码相邻的两位相加（舍去进位），作为格雷码的下一位。



□ 格雷码到二进制码的转换

- (1) 二进制码的最高位（最左边）与格雷码的最高位相同。
- (2) 将产生的每一位二进制码，与下一位相邻的格雷码相加（舍去进位），作为二进制码的下一位。



奇偶校验码

信息位 + 1位校验位。

具有检错能力，能发现奇数个代码位同时出错的情况。

构成：信息位（可以是任一种二进制代码）及一位校验位。

校验位数码的编码方式：

“奇校验”时，使校验位和信息位所组成的每组代码中含有奇数个1；

“偶校验”时，使校验位和信息位所组成的每组代码中含有偶数个1。

二进制码1001 0010



校验位

$$(463.5)_{10} = (\quad ? \quad)_{8421\text{BCD}}$$

- ☐ A 10001100011.0101
- ☒ B 010001100011.0101
- ☐ C 10001100011.1000
- ☐ D 010001100011.1010

提交

上一题的解答：

$$(463.5)_{10} = (\quad ? \quad)_{8421\text{BCD}}$$

$$(463.5)_{10} = \left[\begin{array}{c} \boxed{0}100 \\ \hline 4 \end{array} \quad \begin{array}{c} 0110 \\ \hline 6 \end{array} \quad \begin{array}{c} 0011 \\ \hline 3 \end{array} . \quad \begin{array}{c} 0101 \\ \hline 5 \end{array} \right]_{8421\text{BCD}}$$

$$(863.2)_{10} = \left[\begin{array}{c} 1111 \\ \hline 8 \end{array} \quad \begin{array}{c} 1100 \\ \hline 6 \end{array} \quad \begin{array}{c} 0011 \\ \hline 3 \end{array} . \quad \begin{array}{c} 0010 \\ \hline 2 \end{array} \right]_{2421\text{BCD}}$$

不能省略！

不能省略！

ASCII 码

ASCII码为美国信息交换标准代码，由七位二进制代码（ $b_7b_6b_5b_4b_3b_2b_1$ ）组成。

- ◆ 它共有128个代码，可以表示大、小写英文字母、十进制数、标点符号、运算符号、控制符号等，普遍用于计算机的键盘指令输入和数据等，如表所示。

ASCII码

表 ASCII 码表

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	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	BEL	ETB	‘	7	G	W	g	w

键盘上的“0”按键对应于30H，“A”按键对应于41H，
“a”按键对应于61H。

ASCII码表（续）

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	001	010	011	100	101	110	111
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

键盘上的“回车”按键对应于0DH,就是十进制数13。

在七位ASCII码前补加一位“1”，则构成我国汉字编码使用的扩展ASCII码($1b_7b_6b_5b_4b_3b_2b_1$)。用两个扩展ASCII码编码一个汉字，最多可编码 $128 \times 128 = 16384$ 个汉字或字符。

1.5 二值逻辑变量与基本逻辑运算

◆ 逻辑运算： 当0和1表示逻辑状态时，两个二进制数码按照某种特定的因果关系进行的运算。

逻辑运算使用的数学工具是逻辑代数。

◆ 逻辑代数与普通代数:与普通代数不同,逻辑代数中的变量只有0和1两个可取值，它们分别用来表示完全两个对立的逻辑状态。

◆ 在逻辑代数中，有与、或、非三种基本的逻辑运算。

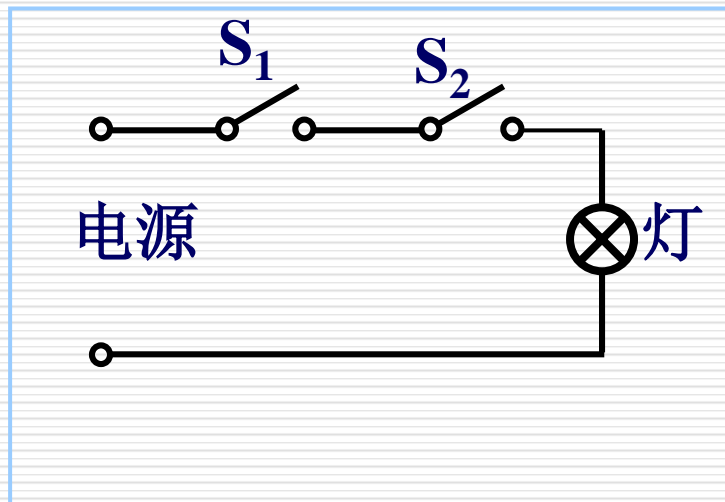
◆ 逻辑运算的描述方式： 逻辑代数表达式、真值表、逻辑图、卡诺图、波形图和硬件描述语言（HDL）等。

1. 与运算

(1) 与逻辑:只有当决定某一事件的条件全部具备时,这一事件才会发生。这种因果关系称为与逻辑关系。

与逻辑举例

电路状态表

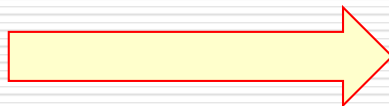


开关 S_1	开关 S_2	灯
断	断	灭
断	合	灭
合	断	灭
合	合	亮

1. 与运算

与逻辑举例状态表

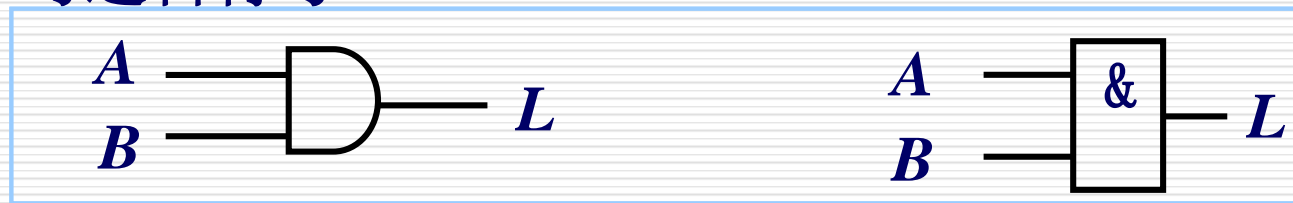
开关S ₁	开关S ₂	灯	开关断开记为: 0 开关合上记为: 1
断	断	灭	
断	合	灭	
合	断	灭	
合	合	亮	灯灭记为: 0 灯亮记为: 1



逻辑真值表

A	B	L
0	0	0
0	1	0
1	0	0
1	1	1

与逻辑符号



美国标准

国际标准

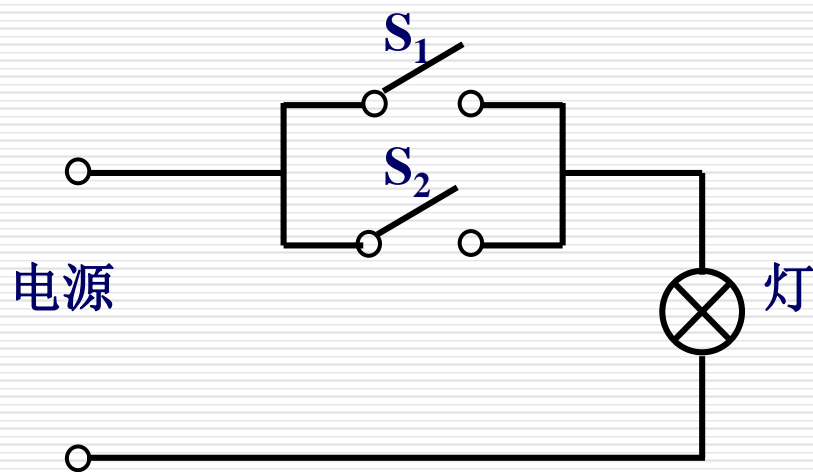
逻辑表达式

与逻辑: $L = A \cdot B = AB$

2、或运算

只要在决定某一事件的各种条件中，**有一个或几个条件具备**时，这一事件就会发生。这种因果关系称为或逻辑关系。

或逻辑举例



电路状态表

开关 S_1	开关 S_2	灯
断	断	灭
断	合	亮
合	断	亮
合	合	亮

2、或运算

或逻辑举例状态表

开关 S_1	开关 S_2	灯
断	断	灭
断	合	灭
合	断	灭
合	合	亮

逻辑真值表

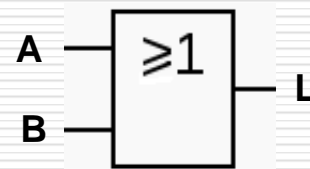
A	B	L
0	0	0
0	1	1
1	0	1
1	1	1



或逻辑符号



美国标准



国际标准

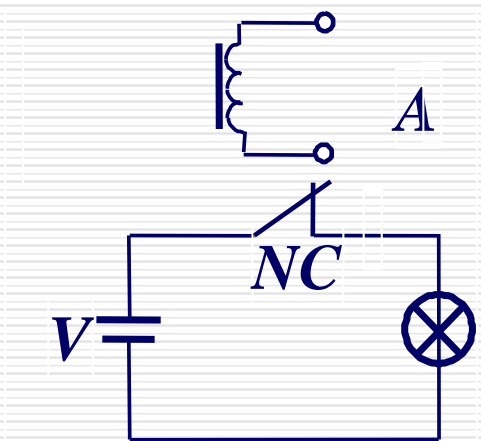
逻辑表达式

或逻辑: $L = A + B$

3. 非运算

事件发生的**条件具备**时，事件不会发生；事件发生的条件不具备时，事件发生。这种因果关系称为非逻辑关系。

非逻辑举例



非逻辑举例状态表

A	灯
不通电	亮
通电	灭

3. 非运算

非逻辑举例状态表

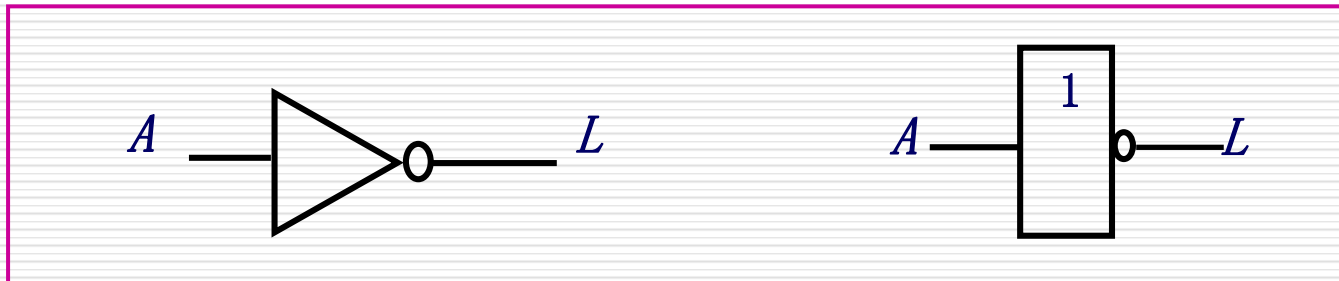
A	灯
不通电	亮
通电	灭



非逻辑真值表

A	L
0	1
1	0

非逻辑符号



逻辑表达式

$$L = \overline{A}$$

对于一个非零的 n 位二进制数，以下说法错误的是（ ）

- ☐ A 如果该数是负数，则它的反码等于 $(n \text{ 个 } 1)_2$ ，减去该数
- ☐ B 如果该数是负数，则它的补码等于 $(n \text{ 个 } 1)_2$ ，减去该数，加1
- ☒ C 如果该数是负数，则它的补码等于 $(1 \text{ 后面 } n \text{ 个 } 0)_2$ ，减去该数。减1
- ☐ D 如果该数是负数，则它的补码等于 2^n ，减去该数

提交

4. 几种常用复合逻辑运算

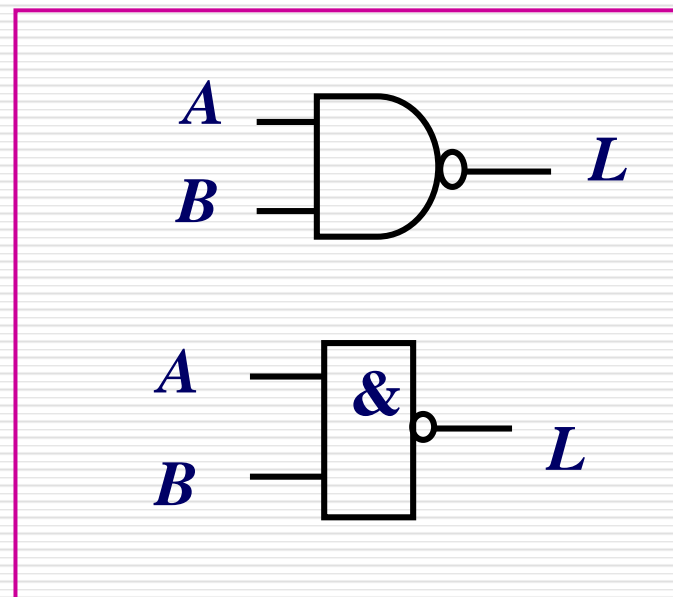
1)与非运算

两输入变量与非
逻辑真值表

A	B	L
0	0	1
0	1	1
1	0	1
1	1	0

与非逻辑表达式

与非逻辑符号



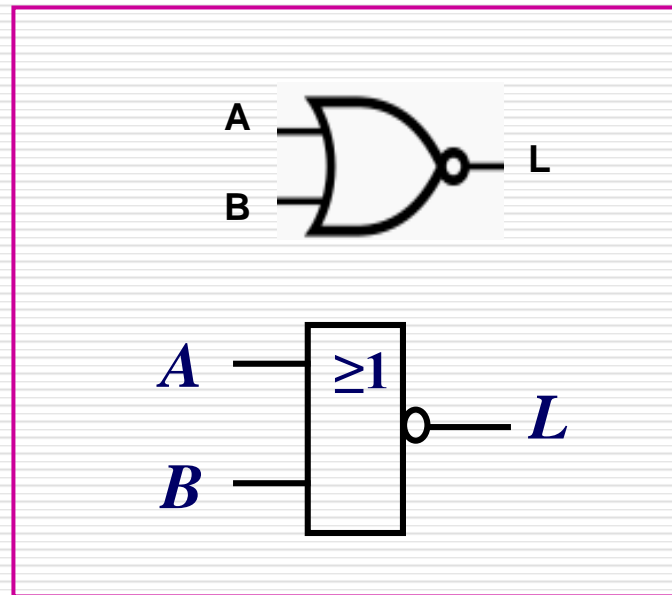
$$L = \overline{A \cdot B}$$

2)或非运算

或非逻辑符号

两输入变量或非
逻辑真值表

A	B	L
0	0	1
0	1	0
1	0	0
1	1	0



或非逻辑表达式

$$L = \overline{A+B}$$

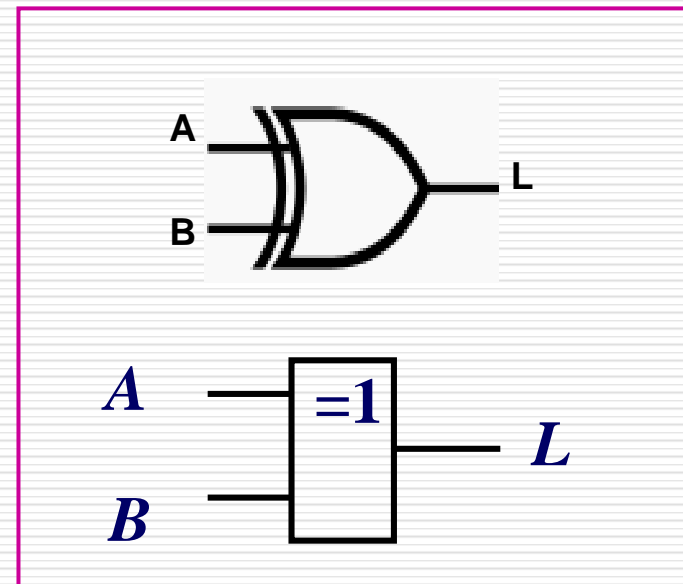
3) 异或逻辑

若两个输入变量的值相异，输出为1，否则为0。

异或逻辑真值表

A	B	L
0	0	0
0	1	1
1	0	1
1	1	0

异或逻辑符号



异或逻辑表达式 $L = A \oplus B = \bar{A}B + A\bar{B}$

4) 同或运算

若两个输入变量的值相同，输出为1，否则为0。

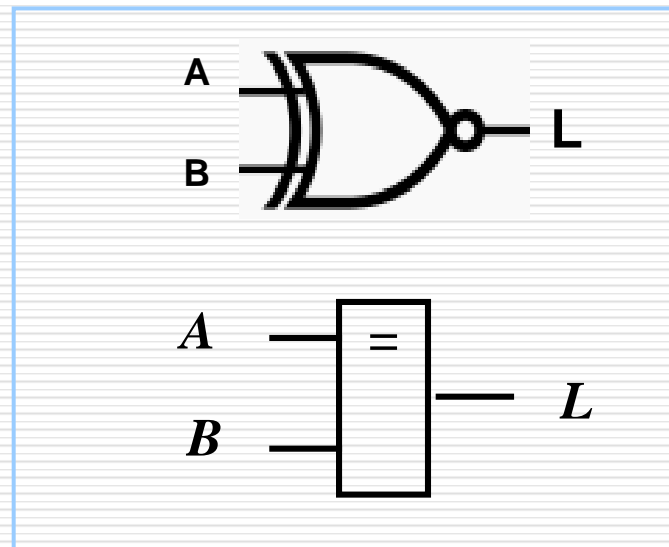
同或逻辑真值表

A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

同或逻辑表达式

$$L = AB + \overline{A}\overline{B} = A \odot B$$

同或逻辑逻辑符号

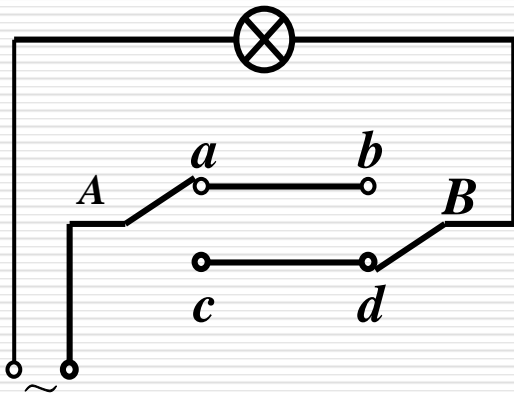


1.6 逻辑函数的建立及其表示方法

1.6.1 逻辑函数几种表示方法

1. 真值表表示

楼道灯开关示意图



确定变量、函数，并赋值

开关： 变量 A、B

灯 ： 函数 L

A、B: 向上—1 向下--0

L : 亮---1; 灭---0

逻辑抽象，列出真值表
开关状态表

开关 A	开关 B	灯
下	下	亮
下	上	灭
上	下	灭
上	上	亮

逻辑真值表

A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

2、逻辑函数表达式表示。

逻辑表达式是用与、或、非等运算组合起来，表示逻辑函数与逻辑变量之间关系的逻辑代数式。

例：已知某逻辑函数的真值表，试写出对应的逻辑函数表达式。

逻辑真值表

A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

$$L = \overline{A} \overline{B}$$

可以将 $A=0, B=0$ 代入上式验证 $L=1$

$$L = A B$$

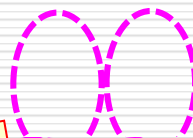
可以将 $A=1, B=1$ 代入上式验证 $L=1$

3. 逻辑图表示方法

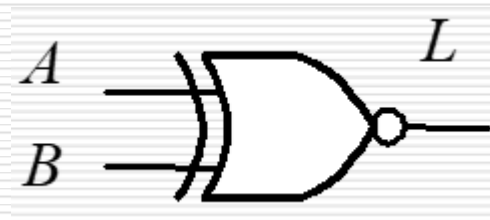
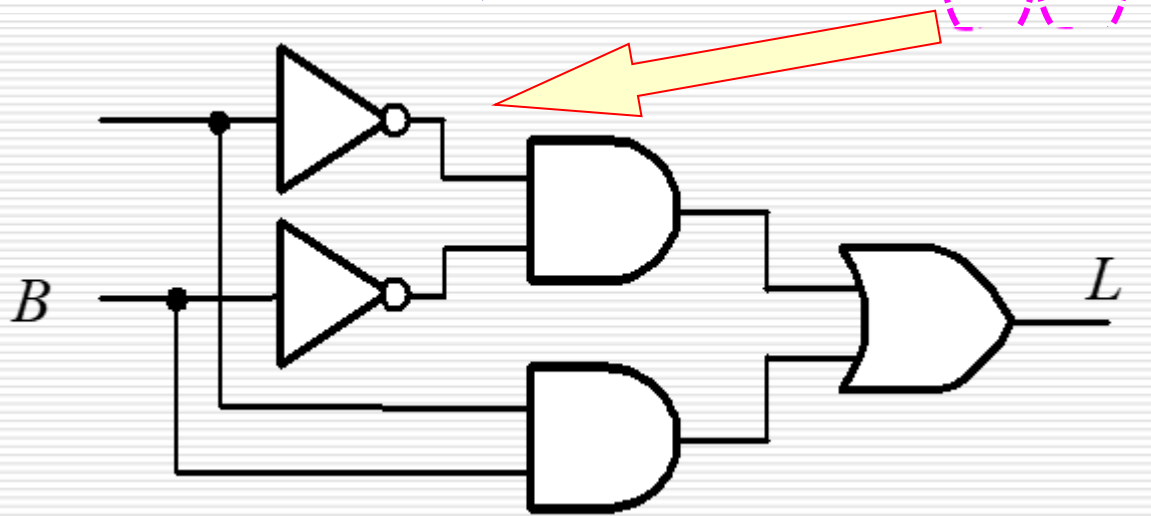
◆ 用与、或、非等逻辑符号表示逻辑函数中各变量之间的逻辑关系所得到的图形称为**逻辑图**。

◆ 将逻辑函数式中所有的与、或、非运算符号用相应的逻辑符号代替，并按照逻辑运算的先后次序将这些逻辑符号连接起来，就得到图电路所对应的逻辑图

例：已知某逻辑函数表达式为



，试画出其逻辑图

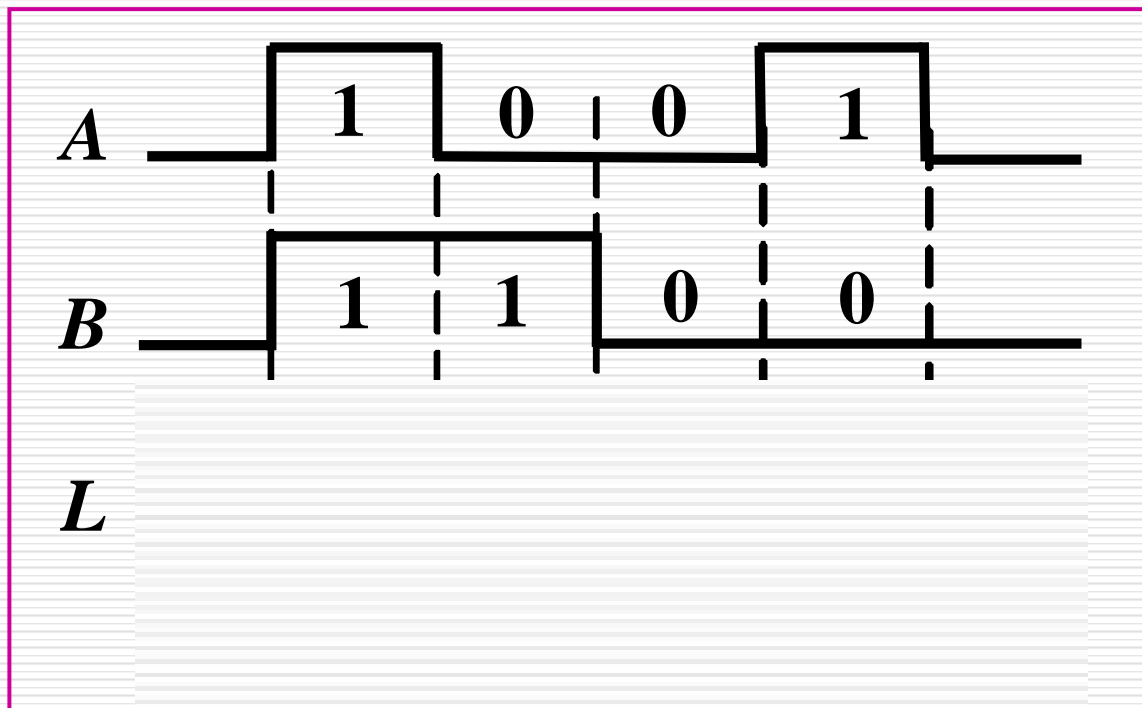


4. 波形图表示方法

用**输入端**在不同逻辑信号**作用下**所对应的输出信号的**波形图**，表示电路的逻辑关系。

真值表

<i>A</i>	<i>B</i>	<i>L</i>
0	0	1
0	1	0
1	0	0
1	1	1



1.6.2 逻辑函数表示方法之间的转换

逻辑函数的真值表、逻辑函数表达式、逻辑图、波形图、卡诺图及HDL描述之间可以相互转换。这里介绍两种转换。

1.真值表到逻辑图的转换

真值表如右表。

转换步骤：

(1)根据真值表写出逻辑表达式

$$L = \overline{A} B C + A B \overline{C}$$

(2)化简逻辑表达式（第2章介绍）

上式不需要简化

$$L = \overline{A} B C$$

$$L = A B \overline{C}$$

<i>A</i>	<i>B</i>	<i>C</i>	<i>L</i>
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(3)根据与或逻辑表达式画逻辑图

$$L : \overline{A} B C + A \overline{B} \overline{C}$$

用与、或、非符号代替相应的逻辑符号，注意运算次序。

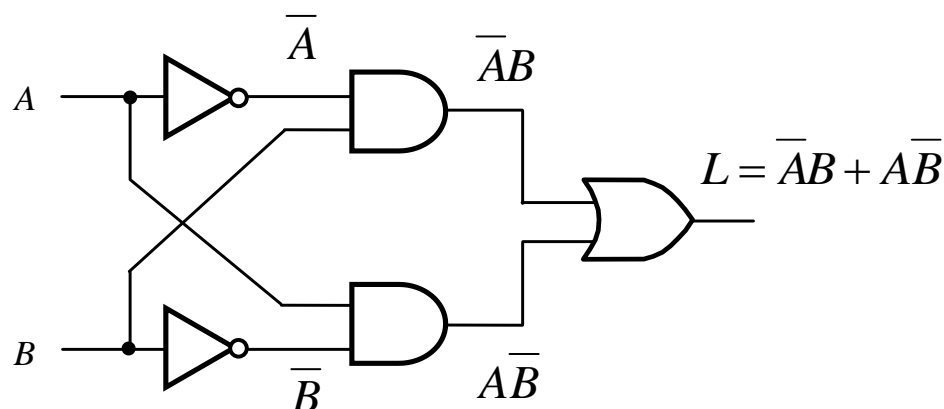
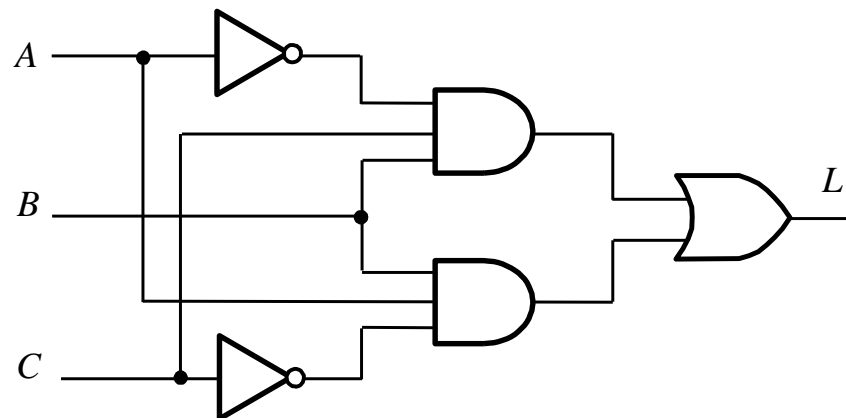
2. 逻辑图到真值表的转换

转换步骤：

(1)根据逻辑图逐级写出表达式

(2)化简变换求最简与或式

(3)将输入变量的所有取值逐一代入表达式得真值表



A	B	L
0	0	0
0	1	1
1	0	1
1	1	0

第二章

逻辑代数与硬件描述语言基础

2.1 逻辑代数的基本定理和规则

2.1.1 逻辑代数的基本定律和恒等式

2.1.2 逻辑代数的基本规则

2.1.1 逻辑代数的基本定律和恒等式

1、基本公式

$$0、1律: A + 0 = A \quad A + 1 = 1 \quad A \cdot 1 = A \quad A \cdot 0 = 0$$

$$互补律: A + \overline{A} = 1 \quad A \cdot \overline{A} = 0$$

$$交换律: A + B = B + A \quad A \cdot B = B \cdot A$$

$$结合律: A + B + C = (A + B) + C \quad A \cdot B \cdot C = (A \cdot B) \cdot C$$

$$分配律: A (B + C) = AB + AC \quad A + BC = (A + B)(A + C)$$

重叠律:

$$A + A = A$$

$$A \cdot A = A$$

反演律(摩根定理): $\overline{A + B} = \overline{A} \cdot \overline{B}$ $\overline{AB} = \overline{A} + \overline{B}$

吸收律

$$A + A \cdot B = A$$

$$A \cdot (A + B) = A$$

$$A + \overline{A} \cdot B = A + B$$

$$(A + B) \cdot (A + C) = A + BC$$

其它常用恒等式

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

$$AB + \overline{A}C + BCD = AB + \overline{A}C$$

2、基本公式的证明（真值表证明法）

例 证明： $A + \bar{A} \cdot B = A + B$

列出等式、右边的函数值的真值表

A	B	\bar{A}	$\bar{A} \cdot B$	$A + \bar{A}B$	$A + B$
0	0				
0	1				
1	0				
1	1				

例：试化简下列逻辑函数 $L = (A + B)(\bar{A} + B)$

$$L = A\bar{A} + AB + B\bar{A} + BB \text{ (分配律)}$$

$$= 0 + AB + B\bar{A} + B \quad (A \cdot \bar{A} = 0, A \cdot A = A)$$

$$= AB + B\bar{A} + B \quad (A + 0 = A)$$

$$= B(A + \bar{A} + 1) \quad [AB + AC = A(B + C)]$$

$$= B \cdot 1 = B \quad (A + 1 = 1, A \cdot 1 = A)$$

2.1.2 逻辑代数的基本规则

1. 代入规则 : 在包含变量A逻辑等式中, 如果用另一个函数式代入式中所有A的位置, 则等式仍然成立。这一规则称为代入规则。

例: $B(A + C) = BA + BC$,

用 $A + D$ 代替 A , 得

$$B[(A + D) + C] = B(A + D) + BC = BA + BD + BC$$

代入规则可以扩展所有基本公式或定律的应用范围

2. 反演规则:

对于任意一个逻辑表达式 L ，若将其中所有的与 (\cdot) 换成或 $(+)$ ，或 $(+)$ 换成与 (\cdot) ；原变量换为反变量，反变量换为原变量；将1换成0，0换成1；则得到的结果就是原函数的反函数 \bar{L} 。

- Swapping $+$ and \cdot (与或交换)
- Complementing all variables (变量取反)
- 遵循原来的运算优先 (Priority) 次序
- 不属于单个变量上的反号应保留不变，即多个变量上的反号保留不变。

例2.1.1 试求 $L = \bar{A}\bar{B} + CD + 0$ 的非函数

解：按照反演规则，得

$$\bar{L} = (A + B) \cdot (\bar{C} + \bar{D}) \cdot 1 = (A + B)(\bar{C} + \bar{D})$$

➤ Complement Rules (反演规则)

Example 1: Write the Complement function for each of the following logic functions. (写出下面函数的反函数)

$$F1 = A \cdot (B + C) + C \cdot D$$

$$\overline{F1} = ?$$

$$= (\bar{A} + \bar{B} \cdot \bar{C}) \cdot (\bar{C} + \bar{D})$$

$$F2 = \overline{A \cdot B} + C \cdot D \cdot \bar{E}$$

$$\overline{F2} = ?$$

$$= (\bar{A} + \bar{B}) \cdot (\bar{C} + \bar{D} + E)$$

$$= A \cdot B \cdot (\bar{C} + \bar{D} + E)$$

➤ Complement Rules (反演规则)

Example 2: Prove $\overline{(A \cdot B + \bar{A} \cdot C)} = A \cdot \bar{B} + \bar{A} \cdot \bar{C}$
证明

$$\begin{aligned} & \overline{A \cdot B + \bar{A} \cdot C} \\ &= (\bar{A} + \bar{B}) \cdot (A + \bar{C}) \\ &= \bar{A} \cdot A + A \cdot \bar{B} + \bar{A} \cdot \bar{C} + \bar{B} \cdot \bar{C} \\ &= A \cdot \bar{B} + \bar{A} \cdot \bar{C} + \bar{B} \cdot \bar{C} \\ &= A \cdot \bar{B} + \bar{A} \cdot \bar{C} \end{aligned}$$

$$\text{令 } F = (A \cdot B + \bar{A} \cdot C)$$



$$\bar{F} = (\bar{A} + \bar{B}) \cdot (A + \bar{C})$$

3. 对偶规则:

对于任何逻辑函数式，若将其中的与（ \cdot ）换成或（ $+$ ），或（ $+$ ）换成与（ \cdot ）；并将1换成0，0换成1；那么，所得的新的函数式就是 L 的对偶式，记作 L' 。

•对偶规则

$\cdot \Leftrightarrow +$ $0 \Leftrightarrow 1$ 所有变量不变，所有运算符变化

变换时不能破坏原来的运算顺序（优先级）。也就是说，为了保持运算的先后顺序不变，需要增加括号。

例：逻辑函数 $L = (A + \bar{B})(A + C)$ 的对偶式为

$$L' = \bar{A}\bar{B} + AC$$

•对偶原理 (Principle of Duality)

若两逻辑式相等，则它们的对偶式也相等。