2　数字逻辑基础

2.1　数制

2.2　码制

2.3　算术运算与逻辑运算

2.4　逻辑函数及其表示方法

2.5　逻辑函数化简与变换

2.1 数制

2.1.1 几种常用数制

数制是进位计数制的简称，也叫位置计数制。

基本概念：

(1) 进位基（模）数（Radix/Base)

使用数码符号的总数，简称为“基”或模 。

例如十进制，数码符号为0，1，2，…，9，共10个，故其进位基数R=10。

(2) 数位的权值(Weight)

某个数位上数码为1时所表征的数值，称为该数位的权值，简称“权”。

各个数位的权值均可表示成R^i的形式，例如十进制，各位的权值为10^i。

按权展开式

$$(333.33)_D = 3\times 10^2 + 3\times 10^1 + 3\times 10^0 + 3\times 10^{-1} + 3\times 10^{-2}$$

权　权　权　权　权

位置计数法

推广到任意进制：

1）有R个数字符号，数码d_i从0 ~ R-1。

2）基数R，逢R进一。

3）不同数位上的数具有不同的权值R^i。

4）任意一个R进制数，都可按其权位展成多项式的形式：

$$(D)_R = (d_{n-1} \cdots d_1 d_0 . d_{-1} \cdots d_{-m})_R$$

$$= d_{n-1}R^{n-1} + \cdots + d_1R^1 + d_0R^0 + d_{-1}R^{-1} + \cdots + d_{-m}R^{-m}$$

$$= \sum_{i=-m}^{n-1} d_i R^i$$

常用数制

1. 十进制(Decimal)

十进制数人们最熟悉， 但机器实现起来困难。

十进制的十个数码，必须由十个不同的而且能严格区分的电路状态与之对应，这样将在技术上带来许多困难，而且也不经济，因此在计数电路中一般不直接采用十进制。

一个 开关的通、断两个状态可以分别表示0、1。数字系统中使用二进制比较方便。

2. 二进制(Binary)

a. 每位有2个不同的数码0、 1。

b. 基数R=2，即遵循“逢二进一”的计数规则。

c. 各位的权值为2^i。

$$(1011.01)_B = \underset{8}{1\times 2^3} + \underset{4}{0\times 2^2} + \underset{2}{1\times 2^1} + \underset{1}{1\times 2^0} + 0\times 2^{-1} + 1\times 2^{-2}$$

字节(byte)： 8位二进制数称为一个字节， 如10111101.

字（word)： 微型计算机系统中，16位（bit)定义为一个字，32位称为双字。

Kilo 1K = 1024 = 2^{10}
Mega 1M = 1024K
Giga 1G = 1024M
Tera 1T = 1024G
Peta 1P = 1024T

注意：

数字系统中1K (1024) 不同于物理学中1k (1000)

3. 十六进制(Hexadecimal)

a. 每位有16个不同的数码0、 1、 2、 …、 9、A、B、C、D、E、F。

b. 基数R=16，即遵循“逢十六进一”的计数规则。

c. 各位的权值为16^i。

$$(BD2.3C)_H=B\times16^2+D\times16^1+2\times16^0+3\times16^{-1}+C\times16^{-2}$$

常用数制对照表

十进制（D）	二进制（B）	八进制（O）	十六进制（H）
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

2.1.2 数制间的转换

转换原则:

转换前后整数部分和小数部分必须分别相等!

1. 非十进制数转换成十进制数。

多项式法:

首先把非十进制数写成按权展开的多项式，然后按十进制数的计数规则求其和。所得结果就是其所对应的十进制数。

[例]

$$(DE)_H = (13\times16^1 + 14\times16^0)_D$$
$$= (208+14)_D$$
$$= 222_D$$

$$(110101.101)_B$$
$$= (1\times2^5 + 1\times2^4 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3})_D$$
$$= 53.625_D$$

2. 十进制数转换为其它非十进制的数

整数和小数部分需分别转换。

(1) 整数部分的转换（除基取余法）

$$(D)_{\mathrm{B}} = d_{\mathrm{k}-1}\times 2^{\mathrm{k}-1} + d_{\mathrm{k}-2}\times 2^{\mathrm{k}-2} + \cdots + d_1\times 2^1 + d_0\times 2^0$$

$$= 2(d_{\mathrm{k}-1}\times 2^{\mathrm{k}-2} + d_{\mathrm{k}-2}\times 2^{\mathrm{k}-3} + \cdots + d_2\times 2^1 + d_1) + d_0$$

如果将上式两边同除以2，所得的商为

$$(D)_{\mathrm{n\text{-}1}} = d_{\mathrm{k}-1}\times 2^{\mathrm{k}-2} + d_{\mathrm{k}-2}\times 2^{\mathrm{k}-3} + \cdots + d_2\times 2^1 + d_1$$

余数就是d_0；

若将$(D)_{\mathrm{n-1}}$两边再同时除以2，则所得余数是d_1。重复上述过程，直到商为0，就可得二进制数的数码d_0、d_1、…、d_{n-1}。

十进制整数N转换成R进制数的步骤：

a. 将N除以R，记下所得的商和余数。

b. 将上一步所得的商再除以R，记下所得商和余数。

c. 重复做b，直到商为0。

d. 将各个余数按照和运算过程相反的顺序排列起来，即为R进制的数。

[例] 将$(89)_D$转换为二进制数。

即$(89)_D = (1011001)_B$

[解]

		余数	
2	89		
2	44	…… 1 …… d_0	LSB (Least Significant Bit)
2	22	…… 0 …… d_1	
2	11	…… 0 …… d_2	
2	5	…… 1 …… d_3	
2	2	…… 1 …… d_4	
2	1	…… 0 …… d_5	
	0	…… 1 …… d_6	MSB (Most Significant Bit)

(2) 小数部分的转换（乘基取整法）

$$(D)_{\mathrm{m}} = d_{-1} \times 2^{-1} + d_{-2} \times 2^{-2} + \cdots + d_{-m} \times 2^{-m}$$

将上式两边同时乘以2， 便得到

$$2(D)_{\mathrm{m}} = d_{-1} + (d_{-2} \times 2^{-1} + \cdots + d_{-m} \times 2^{-m+1})$$

上式的整数部分即为d_{-1}。

将上式的小数部分继续乘以2，得到的整数部分即为d_{-2}.

重复上述过程，便可求出二进制小数的各位数码。

十进制的纯小数M转换成*R*进制数的步骤：

a. 将*M*乘以*R*，记下整数部分。

b. 将上一步乘积中的小数部分再乘以*R*，记下整数部分。

c. 重复做b，直到小数部分为0或者满足精度要求为止。

d. 将各步求得的整数转换成R进制的数码，并按照和运算过程相同的顺序排列起来，即为所求的*R*进制数。

[例1] 将$(0.64)_D$转换为二进制数，要求误差小于2^{-10}。

[解]

	0.64	0.28	0.56	0.12	0.24	0.48	0.96	0.92	0.84	0.68
(乘基)	×2	×2	×2	×2	×2	×2	×2	×2	×2	×2
	1.28	0.56	1.12	0.24	0.48	0.96	1.92	1.84	1.68	1.36
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
(取整)	1	0	1	0	0	0	1	1	1	1
	d_{-1}	d_{-2}	d_{-3}	d_{-4}	d_{-5}	d_{-6}	d_{-7}	d_{-8}	d_{-9}	d_{-10}

即$(0.64)_D = (0.1010001111)_B$

[例2] $(11.375)_D=(\ ?\ )_B$

[解]

$(11)_D=(1011)_B$

$(0.375)_D=(0.011)_B$

故

$(11.375)_D=(1011.011)_B$

3. 基数为2^i 的进制间的转换

(1) 二进制数转换成八或十六进制数：

八进制数和十六进制数的基数分别为$8=2^3$，$16=2^4$， 所以三位二进制数恰好相当一位八进制数，四位二进制数相当一位十六进制数。

方法：

a.以二进制数的小数点为起点，分别向左、向右，每三位(或四位)分一组。小数部分不足位时， 在有效位右边补0，使其足位；整数部分，最高位一组不足位时，可在有效位的左边补0。

b.把每一组二进制数转换成八或十六进制数，保持原排序。

[例1] 将$(1101101111000110.1011000101)_B$转换成十六进制数。

[解] 以小数点为界，每4位划分为1组，如下：

0110	1101	1100	0110.1011	0001	0100
6	D	C	6 . B	1	4

即$(1101101111000110.1011000101)_B = (6DC6.B14)_H$

(2) **八或十六进制数转换为二进制数**：按相反的步骤，即只要按原来顺序将每一位八进制数(或十六进制数)用相应的三位(或四位)二进制数代替即可。

[例] 分别求出$(375.46)_O$、$(678.A5)_H$的等值二进制数。

[解] 与八进制数 3 7 5 . 4 6 等值二进制数

011 111 101 . 100 110

与十六进制 6 7 8 . A 5 等值二进制数

0110 0111 1000 . 1010 0101

作业

自练题：

2.1 （4）

2.2 （5）

2.4 （4）

作业题：

2.3 （3）

除了数量大小，0、1组合还可以表示其它信息，如，2位二进制数的4种组合00、01、10、11分别表示前、后、左、右。没有数量概念，成为代表不同事物的代码。
键盘上每个数字、字母和功能键都有一个相应的代码表示。

2.2 码制

代码(编码)：用一定位数的一组二进制数码按一定规则排列起来表示特定信息的数码。

码制：形成这种代码所遵循的规则。

常用的编码

1. 自然二进制码

按自然数顺序排列的二进制码。

常用四位自然二进制码，表示十进制数0~15，各位的权值依次为2^3、2^2、2^1、2^0。

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

2. 二—十进制编码(BCD码)

用四位二进制码的10 种组合表示一位十进制数0~9，简称BCD码(Binary Coded Decimal) 。

由于四位二进制码可以有16种组合，从16种组合中任选用10 种组合，有

$$A_{16}^{10}=\frac{16!}{(16-10)!}\approx 2.9\times 10^{10}$$

而目前使用的编码方式还不到十种。

几种常用的BCD码

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

若某种代码的每一位都有固定的“权值”，则称这种代码为有权码；否则，叫无权码。

(1) 有权码

a. 8421BCD（NBCD）码

16种组合中的前10种表示十进制数0~9，由高位到低位的权值为2^3、2^2、2^1、2^0，即为8、4、2、1， 8421BCD由此得名。

十进制数	8421码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

[例]$(276.8)_{10}=(\ ?\ )_{NBCD}$

2 7 6 . 8

↓ ↓ ↓ ↓

0010 0111 0110 1000

$(276.8)_{10}=(001001110110.1000)_{NBCD}$

b. 5421 BCD码

5421 BCD码，从高位到低位的权值分别为5、 4、 2、 1。

十进制数	5421码
0	0000
1	0001
2	0010
3	0011
4	0100
5	1000
6	1001
7	1010
8	1011
9	1100

注：这种有权BCD码中，有的十进制数码存在两种加权方法。

例如， 5421 BCD码中的数码5，既可以用1000表示，也可以用0101表示。

可见，5421 BCD码的编码方案都不是唯一的。

(2) 无权码

这种码的每位没有固定的权，各组代码与十进制数之间的对应关系是人为规定的。

余3码是一种较为常用的无权码。

余 3 码是8421 BCD码的每个码组加3 (0011)形成的。

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

余三码的主要优点是执行十位数相加时可以产生正确的进位信号。使用它进行加法运算的规则是当两个余三码数相加不产生进位时，则应该从结果中减去0011；产生进位时则一方面将进位信号送给高位余三码，另一方面本位还要执行加上0011的修正操作。

数	余3代码	进位	和 数	修正数	结 果
0+0	0011+0011	0	0000 0110	+0011 −0011	0011 0011
0+1	0011+0100	0	0000 0111	+0011 −0011	0011 0100
0+2	0011+0101	0	0000 1000	+0011 −0011	0011 0101
0+3	0011+0110	0	0000 1001	+0011 −0011	0011 0110
0+4	0011+0111	0	0000 1010	+0011 −0011	0011 0111
0+5	0011+1000	0	0000 1011	+0011 −0011	0011 1000
0+6	0011+1001	0	0000 1100	+0011 −0011	0011 1001
0+7	0011+1010	0	0000 1101	+0011 −0011	0011 1010
0+8	0011+1011	0	0000 1110	+0011 −0011	0011 1011
0+9	0011+1100	0	0000 1111	+0011 −0011	0011 1100
9+1	1100+0100	1	0000 0000	+0011 +0011	0100 0011
9+2	1100+0101	1	0000 0001	+0011 +0011	0100 0100
9+3	1100+0110	1	0000 0010	+0011 +0011	0100 0101
9+4	1100+0111	1	0000 0011	+0011 +0011	0100 0110
9+5	1100+1000	1	0000 0100	+0011 +0011	0100 0111
9+6	1100+1001	1	0000 0101	+0011 +0011	0100 1000
9+7	1100+1010	1	0000 0110	+0011 +0011	0100 1001
9+8	1100+1011	1	0000 0111	+0011 +0011	0100 1010
9+9	1100+1100	1	0000 1000	+0011 +0011	0100 1011

返回

8421 BCD算术运算是一种十进制运算，如果2个BCD码相加的和大于$(1001)_2$，则需要进行+6修正，并向高位进位。

例：4+9=13

```
    0   1   0   0
+   1   0   0   1
  ---------------
    1   1   0   1
+   0   1   1   0 （修正）
  ---------------
1   0   0   1   1
--  ---------------
1           3
```

加法运算时2个压缩型BCD码（一个字节存放二个BCD码，低四位和高四位各是一个BCD码）所得结果需要调整。调整后，得到的结果还是压缩型BCD码。

汇编语言：十进制数加调整指令DAA (Decimal Adjust After Addition)

```
MOV AL, 43H

MOV BL, 29H

ADD AL, BL ;AL=6CH，其不是压缩型的BCD码，因为低四位'B'不是BCD码

DAA ;调整后，AL=72H，这是压缩型的BCD码，也有：43+29=72
```

2.其它常用的代码

(1) 格雷(Gray)码

a. 任意两组相邻代码之间只有一位不同。

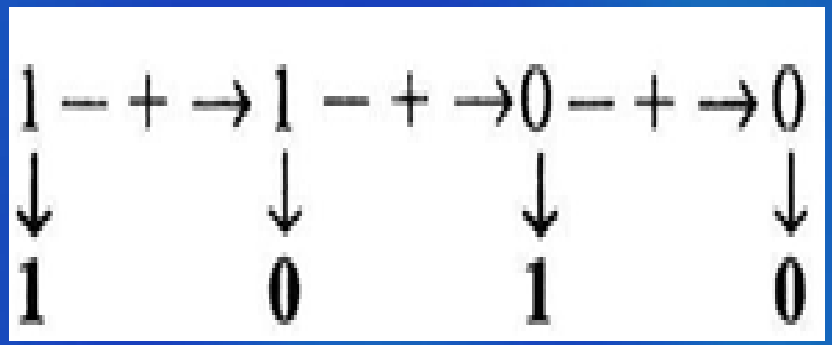
b. 首尾两个数码即最小数和最大数之间也符合此特点，故它可称为循环码。

十进制数	二进制码	Gray码
	$B_3B_2B_1B_0$	$G_3G_2G_1G_0$
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

运算规则（二进制码到格雷码）：

最高位保持不变，从最高位开始，相邻的两位相加，但不进位，其结果作为格雷码的次高位，依此类推，一直加到最低位。

Example Convert the binary number 1100 to Gray codes.

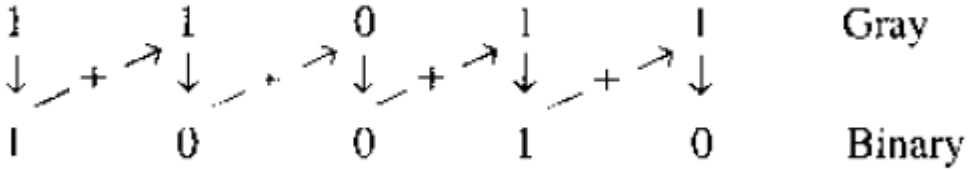


格雷码到二进制码：

Gray-to-Binary Conversion To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code number 11011 to binary is as follows:



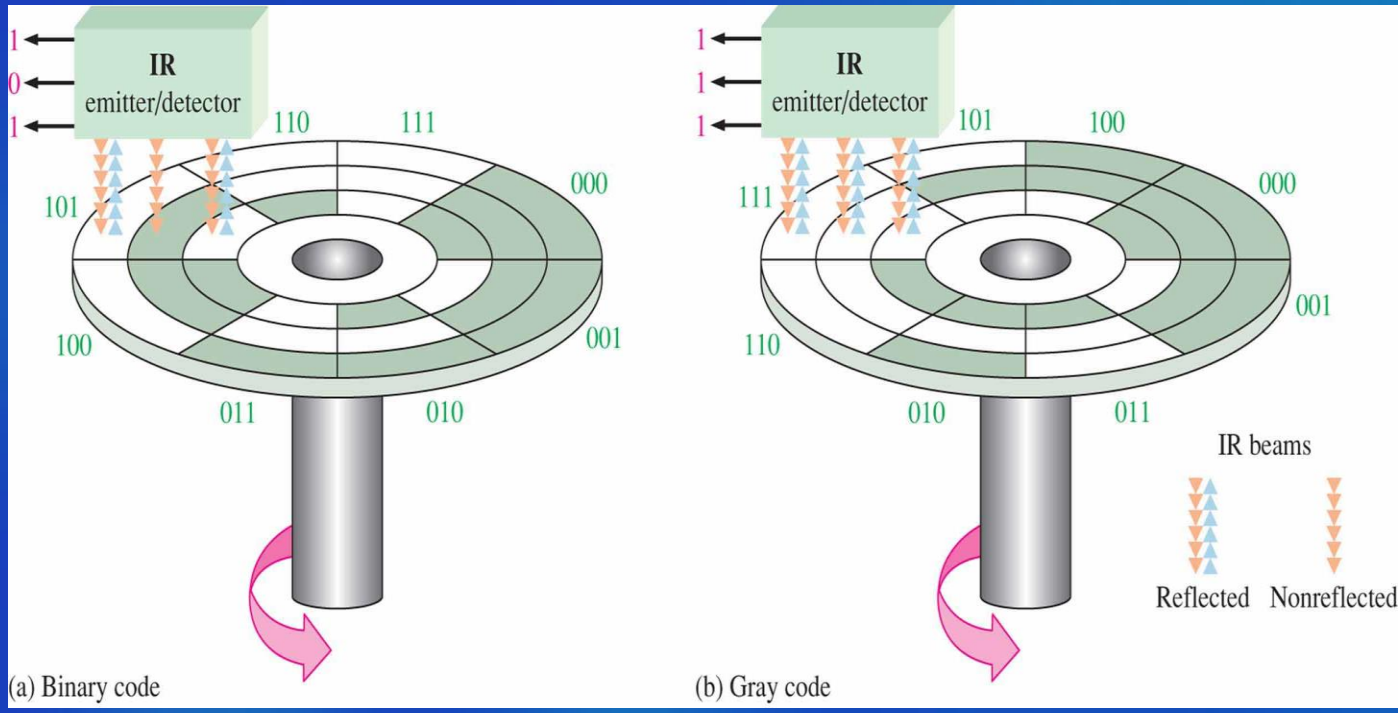
The binary number is 10010.

格雷码的单位距离特性可以降低错误的发生。

思考：一个绕轴转动的圆盘，如何实现自动定位?

Consider what happens when the beams are on the 111 sector and about to enter the 000 sector. If the MSB beam (the outer circle) is slightly ahead.

Then, the position would be incorrectly indicated by a transitional 011 instead of a 111 or a 000.



Gray code: Consider what happens when the beams are on the 111 sector and about to move into the next sector. The only two possible outputs are 111 and 101, no matter how the beams are aligned.

(2) 奇偶校验码

在信息存储与传送过程中，常由于某种随机干扰而发生错误。所以希望在传送代码时能进行某种校验以判断是否发生了错误，甚至能自动纠正错误。

奇偶校验码是一种可以检测一位错误的代码。

信息位 + 校验位

a. 使每一个码组中信息位和校验位的“1”的个数之和为奇数，称为奇校验。

b. 使每一个码组中信息位和校验位的“1”的个数之和为偶数，称为偶校验。

8421BCD奇偶校验码

十进制数	奇校验 8421BCD		偶校验 8421BCD	
	信息位	校验位	信息位	校验位
0	0000	1	0000	0
1	0001	0	0001	1
2	0010	0	0010	1
3	0011	1	0011	0
4	0100	0	0100	1
5	0101	1	0101	0
6	0110	1	0110	0
7	0111	0	0111	1
8	1000	0	1000	1
9	1001	1	1001	0

奇偶校验码常用于**串行通信纠错**，发送数据之前，收发双方约定好奇偶校验方式，接收端检查接收代码的奇偶性，若与发送端的奇偶性一致，则可认为接收到的代码正确，否则，接收到的一定是错误代码。

奇偶校验码只能检测一位错码，但不能测定哪一位出错，也不能自行纠正错误。若代码中同时出现多位错误，则奇偶校验码无法检测。但是，由于多位同时出错的概率要比一位出错的概率小得多，并且奇偶校验码容易实现，因而该码被广泛采用。

(3) 字符码

专门用来处理数字、字母及各种符号的二进制代码。

目前广泛使用字符码有博多码和ASCII码。

博多码是由五位二进制码组成，主要用于电传打字和数字通信中。

ASCII码(American Standard Code for Information Interchange，美国信息交换标准代码)由七位二进制码组成(128个字符），主要用于数字系统和计算机中。

高四位 $0b_6b_5b_4$ / 低四位 $b_3b_2b_1b_0$		ASCII非打印控制字符										ASCII 打印字符											
		0000					0001					0010		0011		0100		0101		0110		0111	
		0					1					2		3		4		5		6		7	
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符
0000	0	0	BLANK NULL	^@	NUL	空	16	►	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p
0001	1	1	☺	^A	SOH	头标开始	17	◄	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ

每个字符都是由代码的高3位$b_6b_5b_4$和低4位$b_3b_2b_1b_0$一起确定的。

3的ASCII码为33H，A的ASCII码为41H等。

2.3 算术运算和逻辑运算

在数字系统中，二进制数码0和1表示：{ 数量信息 / 逻辑信息

表示数量信息的运算称为算术运算。

表示逻辑信息的运算称为逻辑运算（与、或、非….）。

表示事物两种不同的逻辑状态，例如，用0、1分别表示电路的通、断，或者表示一件事情 是与非、真与假等。

2.3.1 算术运算

(1) 加法运算

0+0=0　　0+1=1　　1+0=1　　1+1=10　逢二进一

(2) 减法运算

0-0=0　　1-0=1　　1-1=0　　0-1=1 (向高位借1当2)

(3) 乘法运算

0×0=0　　1 × 0=0　　1 × 0=0　　1 × 1=1

(4) 除法运算

0÷1=0　　1 ÷ 1=1

数值在计算机中的表示

数字电路中，一位二进制是运算和信息处理的最小单位，各种信息都是用多位二进制编码进行识别和存储，以二进制数的形式进行运算处理的。

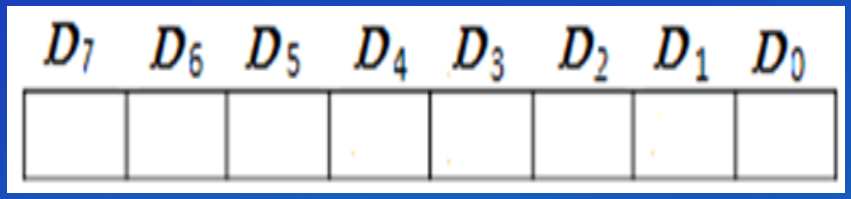
一个二进制**位**为一个**比特**（bit），8个二进制位为一个**字节**（Byte）。16位为一个**字**（Word），32位为**双字**（DWord）、64位为**四倍字**（QWord）。

计算机在同一时间内处理的一组二进制数称为一个计算机的“**字**”，而这组二进制数的位数就是“**字长**”。

通常称处理字长为8位数据的CPU叫8位CPU，32位CPU就是在同一时间内处理字长为32位的二进制数据。**这里的“字”并不是上述的双字节（Word）概念**，它决定着寄存器、加法器、传送数据的总线等设备的位数，直接影响着硬件的代价。

计算机数据线的位数和字长是相同的。这样从内存获取数据后，只需要一次就能把数据全部传送给CPU。

MSB **LSB**



每位的取值为0/1

一个字节的各位编号

一个字节所能表示的数值范围是什么？ 0~255 **无符号数**

在数字设备中，“+”、“-”号需要占用一位来表示。连同符号位在一起的数，叫**有符号数**。

unsigned int x

signed int y

带符号数的表示

一般将数的最高位设为符号位 { “0”表示 “+”；“1”表示 “-”

真值	机器数（原码）
+101	0 101
-101	1 101

Question:

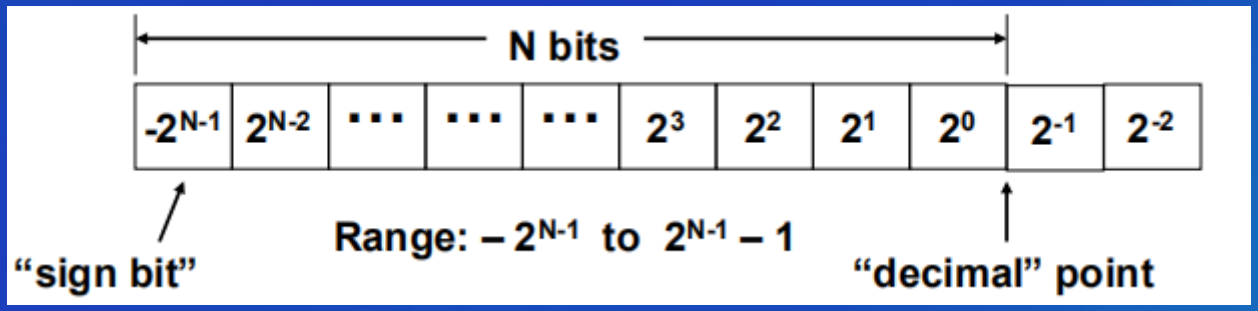
+5 + (-5) = 0

0101 + 1101 ≠ 0

+0: 0 000

-0: 1 000

2's complement (二进制补码)



编码规则：最高位的权值是一个带符号、同时带大小的权

N位补码的范围：-2^{N-1} ~ $2^{N-1}-1$

4位补码的范围：-8 ~ -1，0 ~ 7

+5 + (-5) = 0

0 101 + 1 011 = 0 000

基本上在所有的现代CPU体系结构中，带符号的二进制数都以补码形式表示。

思考：十进制数“-128”的补码是什么？

快速得到补码：

正数：$[X]_{补}$ = 原值（原码）

负数：按位取反（反码）+ 1，即$[X]_{补} = [X]_{反} + 1$

$X_1 = +4$ $[X_1]_{补} = 0\ 100$

$X_2 = -4$ $[X_2]_{补} = 1\ 011 + 1 = 1\ 100$

数位的扩展：如8位表示

$X_1 = +4$ $[X_1]_{补} = 00000100$

$X_2 = -4$ $[X_2]_{补} = 11111011+1 = 1111\ 1100$

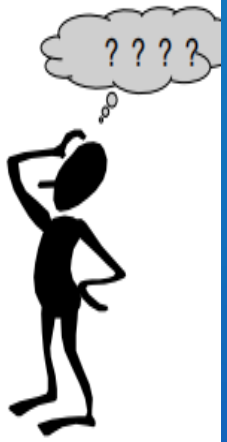
结论：直接将符号进行扩展

补码运算的好处：变减法运算为加法运算

13+10　　13-10　　-13+10　　-13-10

+13　0　01101	+13　0　01101	+13　0　1101
+10　0　01010	−10　1　10110	+10　0　1010
+23　0　10111	+3　0　00011	+23　1　0111
−13　1　10011	−13　1　10011	−13　1　0011
+10　0　01010	−10　1　10110	+10　0　1010
−3　1　11101	−23　1　01001	−3　1　1101

+13　0　1101
−10　1　0110
+3　0　0011
−13　1　0011
−10　1　0110
−23　0　1001



结论：只要位数选的足够，将2个加数的符号位和来自数值最高位的进位相加，结果就是和的符号位。

小数的补码

-2^{N-1}	2^{N-2}	.....	2^1	2^0	2^{-1}	2^{-2}

decimal point

8-bit 2's complement example:

11010110

$= -2^7 + 2^6 + 2^4 + 2^2 + 2^1$

$= -128 + 64 + 16 + 4 + 2$

$= -42$

By moving the implicit location of "decimal" point, we can represent fractions too:

1101.0110

$= -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3}$

$= -8 + 4 + 1 + 0.25 + 0.125$

$= -2.625$

问题：

对10进制的小数，在求补码、利用取反加1时，1应加在何处?

所有转换都在计算机最底层进行，而在我们使用的汇编、C等其他高级语言中使用的都是原码。

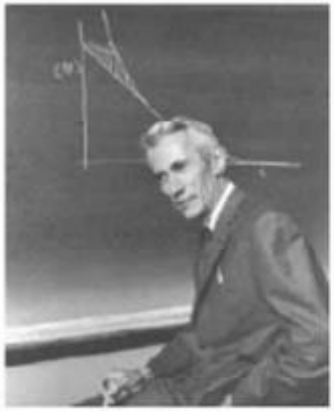
2.3.2 基本逻辑运算及符号

逻辑：事物之间的因果关系

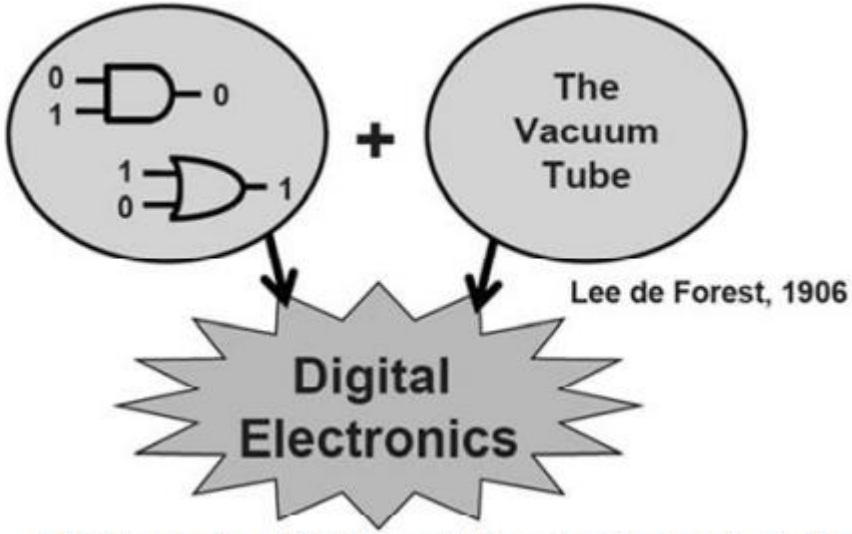
逻辑代数：逻辑运算的数学方法（布尔代数）

- 1854年，George Boole指出“逻辑不仅仅是哲学，也是数学”
- 数字电路中的逻辑代数：二值逻辑，逻辑变量的取值只有0和1两种情况

逻辑代数(Logic algebra)又叫布尔代数(Boolean algebra)。



Claude Shannon



- Claude Shannon发现布尔代数和电话交换电路之间存在相似性
- 1937年Shannon在他的MIT硕士论文“A Symbolic Analysis of Relay and Switching Circuits”中提出了二值电子元件，奠定了数字电路的理论基础

到20世纪60年代，数字技术的发展，布代数成为逻辑设计的基础，广泛地应用于数字电路的分析和设计中。但逻辑代数（数学模型）是比数字电路（物理模型）更高的1种抽象模型，逻辑代数描述的规则不会因实现的方式而变。

*逻辑变量*用字母来表示，取值0和1。常称为二值变量。

输入逻辑变量——表示事件条件的变量。

输出逻辑变量——表示事件结果的变量。

逻辑函数

设某一逻辑网络的输入变量为A_1、A_2、…、A_n，输出变量为L。当A_1、A_2、…、A_n的取值确定之后，L的值就惟一地被确定下来，则称L是A_1、A_2、…、A_n的逻辑函数：

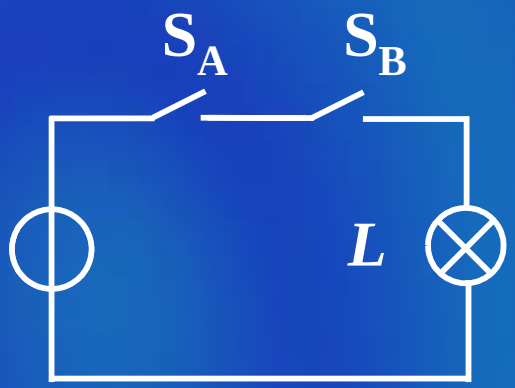
$$L=f(A_1、A_2、…、A_n)$$

基本逻辑运算

1. 与运算(逻辑乘)

当决定某事件的全部条件都具备时，事件才发生的因果关系，这种因果关系称之为逻辑与。

(1) 逻辑与电路

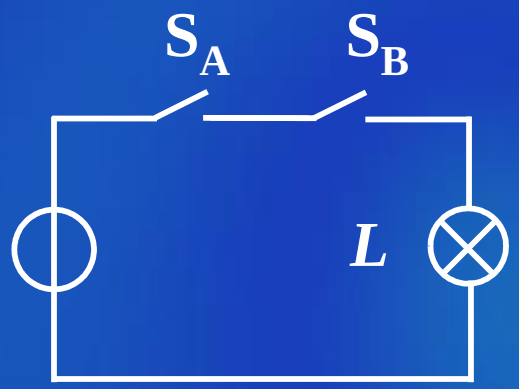


(2) 与逻辑真值表

真值表——由逻辑变量的所有可能取值组合及其对应的逻辑函数值所构成的表格。

用*A*、*B*和*L*分别代表开关S_A、S_B和灯的逻辑状态。

设定开关闭合和灯亮用1表示，开关断开和灯灭用0表示。



与逻辑真值表

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

(3) 逻辑函数式（与逻辑表达式）

$$L = A \cdot B$$

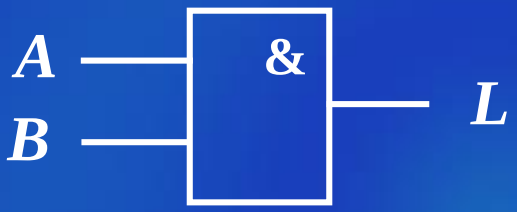
与逻辑真值表

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

(4) 与门符号

实现与运算的逻辑器件称为与门。

与门逻辑符号



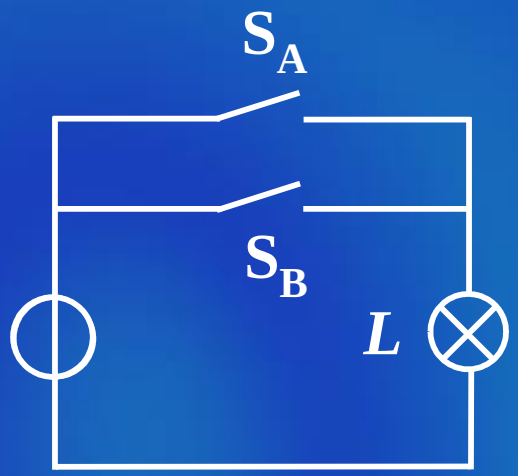
(a)长方形符号
(rectangular-shape symbol)

(b) 特殊形状符号
(distinctive-shape symbol，
更适合PLD结构描述)

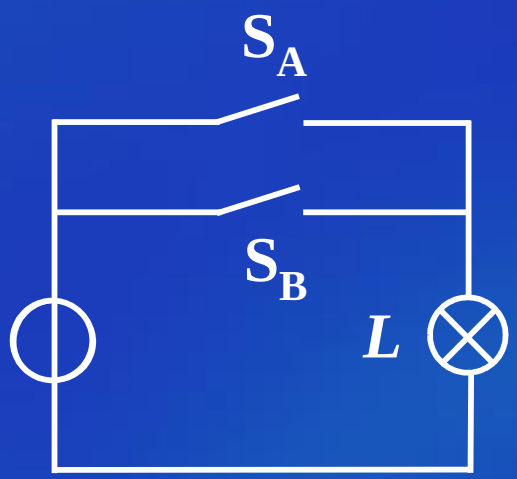
2. 或运算(逻辑加)

逻辑或——当决定某事件的全部条件中，任一条件具备，事件就发生的因果关系。

(1) 或逻辑电路



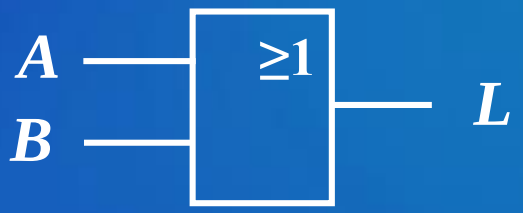
(2) 或逻辑真值表



或逻辑真值表

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

或门逻辑符号



(a)长方形符号

(b)特殊形状符号

(3) 逻辑函数式（或逻辑表达式）

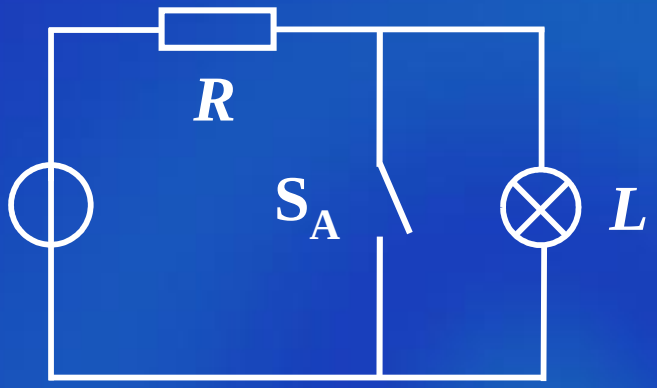
$$L = A + B$$

(4) 或门符号

3. 非运算(逻辑非)

逻辑非——当条件具备时，事件不发生，条件不具备时，事件就发生的因果关系。

(1) 逻辑非电路



(2) 非逻辑真值表

非逻辑真值表

A	L
0	1
1	0

(3) 逻辑函数式（或逻辑表达式）

$$L = \overline{A}$$

$\overline{A}$ 读作“A非”

式中，A称为原变量，$\overline{A}$ 称为反变量。

非运算也称为求反运算。

非逻辑真值表

A	L
0	1
1	0

(4) 非门符号

(a)长方形符号

(b)特殊形状符号

4. 逻辑常量间的基本运算

与	或	非
0·0=0	0+0=0	$\overline{0}=1$
0·1=0	0+1=1	
1·0=0	1+0=1	$\overline{1}=0$
1·1=1	1+1=1	

复合逻辑运算

1. 常用复合逻辑

(1) 与非逻辑

a. 逻辑表达式

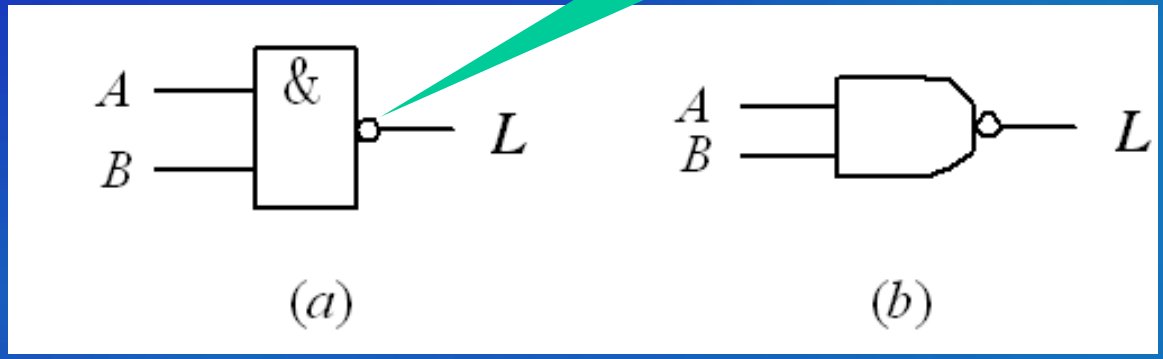
$$L = \overline{AB}$$

b. 真值表

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

c. 逻辑符号

表示非运算



(a)长方形符号　(b)特殊形状符号

(2) 或非逻辑

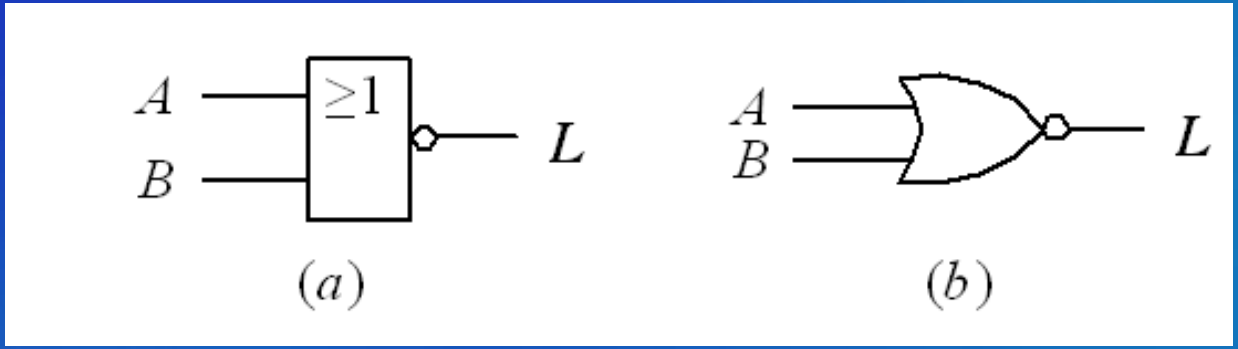
a. 逻辑表达式

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

b. 真值表

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

c. 逻辑符号



(a)长方形符号　　(b)特殊形状符号

(3) 与或非逻辑

a. 逻辑表达式

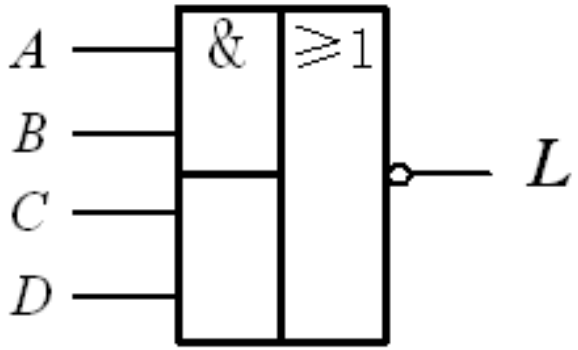
$$L = \overline{AB + CD}$$

b. 真值表

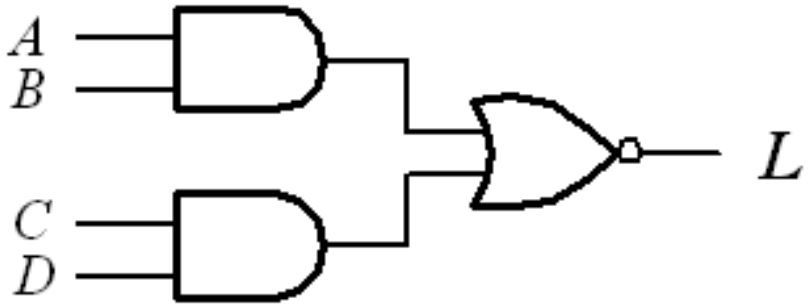
A	B	C	D	L
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

c. 逻辑符号

$$L=\overline{AB+CD}$$



(a)



(b)

(a)长方形符号

(b)特殊形状符号

(4) 异或逻辑

a. 逻辑表达式

$$L = \bar{A}B + A\bar{B} = A \oplus B$$

b. 真值表

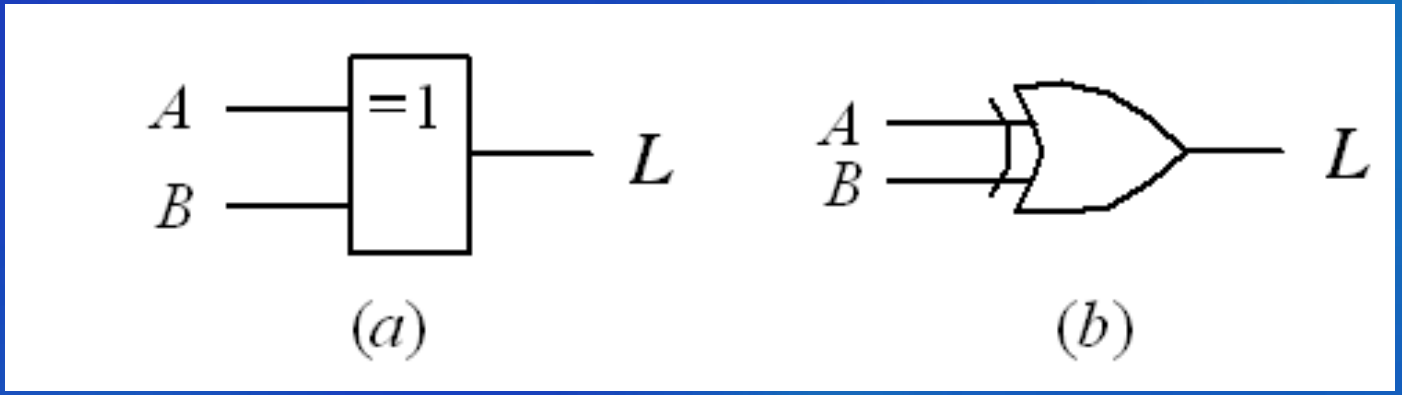
A	B	L
0	0	0
0	1	1
1	0	1
1	1	0

c. 逻辑功能

若两个输入变量A、B的取值相异，则输出变量L为1；若A、B的取值相同，则L为0。

d. 逻辑符号

$$L = \bar{A}B + A\bar{B} = A \oplus B$$



(a)长方形符号　　(b)特殊形状符号

(5) 同或逻辑

a. 逻辑表达式

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

b. 真值表

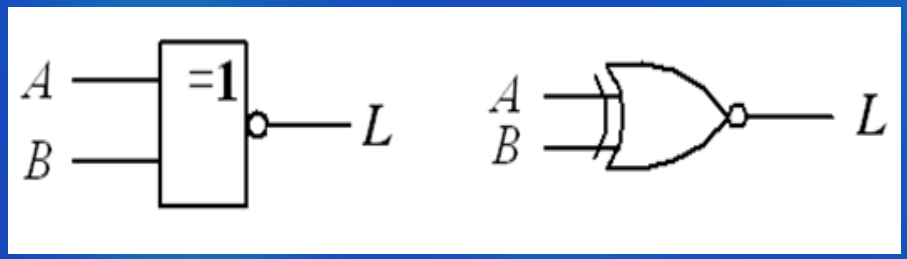
A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

c. 逻辑功能

若两个输入变量A、B的取值相异，则输出变量F为0；若A、B的取值相同，则F为1。

与异或逻辑功能相反

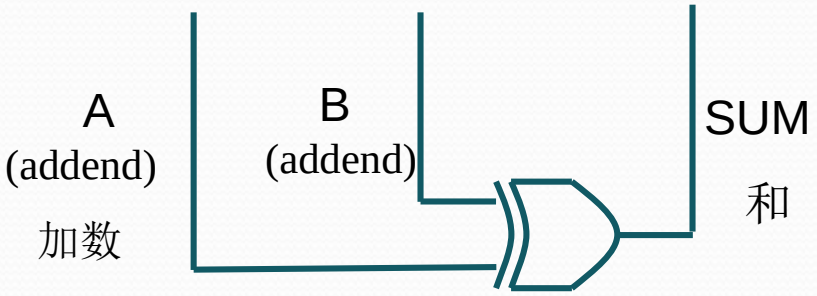
$$A \odot B = \overline{A \oplus B}$$



Applications of the XOR and XNOR gates

Example 1 The XOR gate is used to add two bits.

Input bits		Output
A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	0 (without 1 carry)

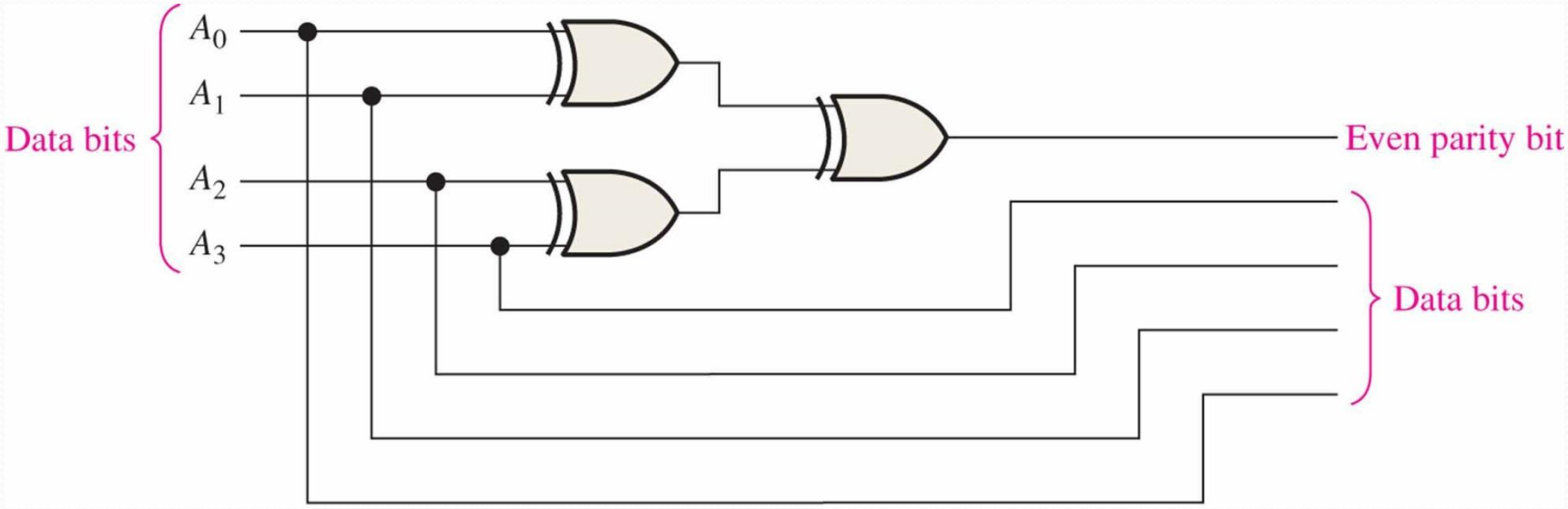


XOR gate can accomplish （完成） the arithmetic addition of 2 1-bit binary numbers, but no carry is produced.

Applications of the XOR and XNOR gates

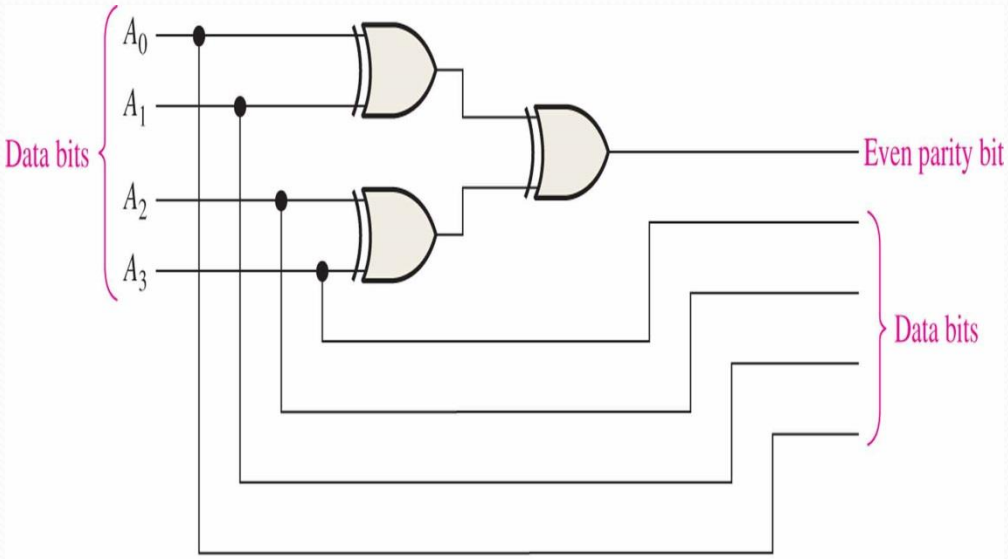
Example 2 Use exclusive-OR gates to implement an even-parity code generator for an original 4-bit code.

Solution



The circuit produces a 1 output when there is an odd number of 1s on the inputs in order to make the total number of 1s in the output code even. A 0 output is produced when there is an even number of 1s on the inputs.

Applications of the XOR and XNOR gates



The circuit produces a 1 output when there is an odd number of 1s on the inputs in order to make the total number of 1s in the output code even. A 0 output is produced when there is an even number of 1s on the inputs.

A3A2A1A0	Even parity bit
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

逻辑函数的基本定理及常用公式

基本公式 (可利用真值表证明)

(1) 0-1律： $A\cdot 0=0$ $A+1=1$

(2) 自等律： $A\cdot 1=A$ $A+0=A$

(3) 重叠律： $A\cdot A=A$ $A+A=A$

(4) 互补律： $A\cdot\overline{A}=0$ $A+\overline{A}=1$

(5) 交换律： $A+B=B+A$ $AB=BA$

(6) 结合律： $(A+B)+C=A+(B+C)$ $(AB)C=A(BC)$

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

(8)摩根(Morgan)定律(反演律) $\overline{AB}=\overline{A}+\overline{B}$ $\overline{A+B}=\overline{A}\,\overline{B}$

(9) 否否律 $\overline{\overline{A}}=A$

证明

$$(F_1) \qquad \overline{A+B} = \overline{A}\,\overline{B} \qquad (F_2)$$

A	B	F_1	F_2
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

重要规则

1. 代入规则

任何一个逻辑等式，如果将所有出现某一逻辑变量的位置都代之以一个逻辑函数，则等式仍成立。

[例1] 证明 $\overline{A+B+C}=\bar{A}\cdot\bar{B}\cdot\bar{C}$

[证明] 两变量的求反公式

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

将等式两边的B用B+C代入便得到

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

得到三变量的摩根定律。

同理，可将摩根定律推广到*n*变量

$$\overline{A_1+A_2+\cdots+A_n}=\overline{A}_1\cdot\overline{A}_2\cdot\ \cdots\ \cdot\overline{A}_n$$

$$\overline{A_1A_2\cdots A_n}=\overline{A}_1+\overline{A}_2+\cdots+\overline{A}_n$$

[例] 求函数 $F = A + \overline{B + \overline{C} + \overline{D + \overline{E}}}$ 的反函数。

[解1] 利用摩根定律

$$\overline{F} = \overline{A + \overline{B + \overline{C} + \overline{D + \overline{E}}}}$$

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

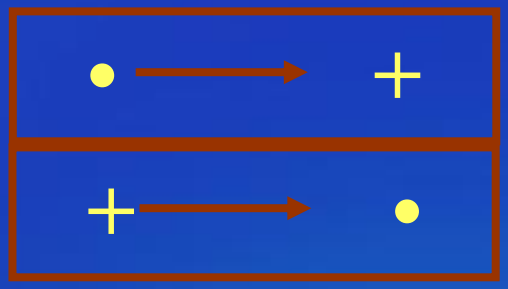
$$= \overline{A} \cdot (B + \overline{C} + \overline{D + \overline{E}})$$

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

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

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

2. 反演规则



变量与常数均取反

得到函数的非运算或反函数 $\overline{F}$

注意:

1. 原运算顺序保持不变。
2. 不是一个变量上的反号不动。

[解2] 利用反演规则

由 $F = A + \overline{B + \overline{C} + \overline{D + E}}$

得 $\overline{F} = \overline{A} \cdot \overline{B \cdot C \cdot \overline{D \cdot E}}$

[例] 求 $F = AB + \overline{A}C$ 的反函数。

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

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

作 业

自练题：

2.5 （4）

2.6（3） （4）

作业题：

2.7 （4）

2.8 （4）