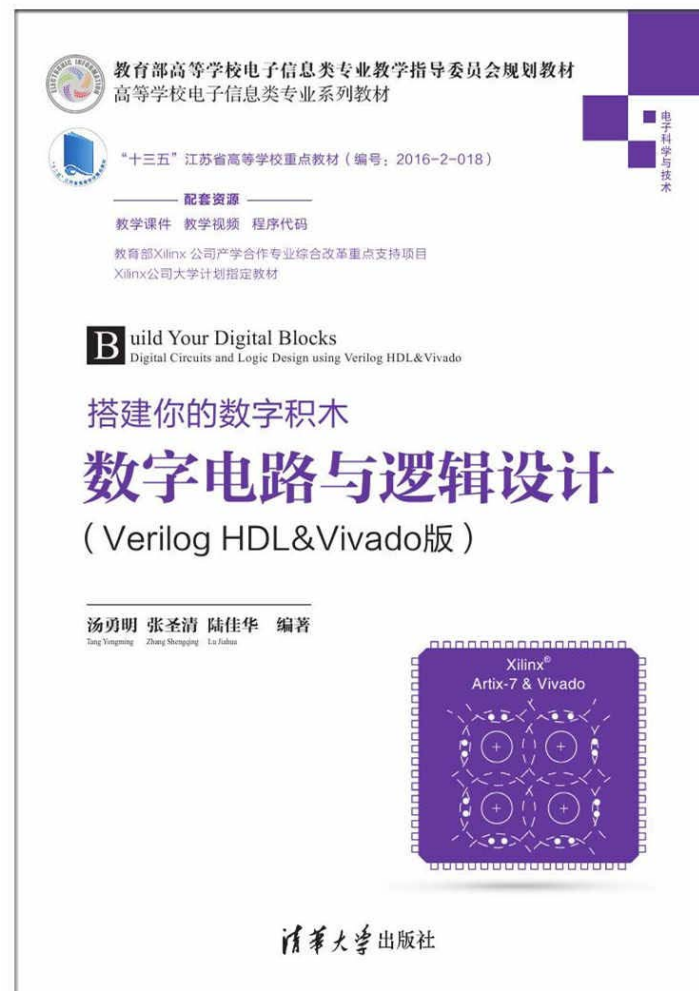


搭建你的数字积木 ——数字电路与逻辑设计 (Verilog HDL&Vivado版) —— 参考课件PPT

东南大学 & Xilinx大学计划部



東南大學
SOUTHEAST UNIVERSITY



Chap.2. 布尔代数和Verilog HDL基础

本章学习导言

- 上一章已经介绍了逻辑信号及数字系统的基本概念，也介绍了Xilinx公司最新的综合开发环境Vivado的基本使用操作，在本章和后续各章学习中都可以借助该综合开发环境。
- 本章的教学目的是给出逻辑电路设计的基本理论——**布尔代数**，包括公理、常用定律和标准表达式等，然后学习可以用来表达布尔代数相关数据类型和逻辑表达式的Verilog HDL语言基本要素，如端口变量申明、运算符等。
- 数制和码制作为可选部分，其中码制可以在布尔代数学习前介绍，也可以在组合逻辑电路设计学习后作为电路设计优化的案例学习。

主要内容

- 数制与码制（可选）
- 布尔代数和布尔定律
- 布尔代数化简
- Verilog HDL语言基础
- 习题与练习

数制和码制

➤ 计算机中的数制

- 十进制与R进制
- 二进制
- 八进制、十六进制
- 数制间的转换

➤ 计算机中的码制

- 数值数据的表示方法
- 非数值数据的表示方法

十进制的定义

(1) **十进制(Decimal):** 以十为基数的记数体制

用十个数码表示:

1、2、3、4、5、6、7、8、9、0

遵循**逢十进一、借一当十**的计数规律

$$157 = 1 \times 100 + 5 \times 10 + 7 \times 1$$

$$= 1 \times 10^2 + 5 \times 10^1 + 7 \times 10^0$$

数码

权重

十进制的定义（续）

一个n位整数和m位小数的十进制数V

$$\begin{array}{ccccccc} d_{n-1} & d_{n-2} & \dots & d_0 & . & d_{-1} & \dots & d_{-m} \\ \downarrow & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 10^{n-1} & 10^{n-2} & & 10^0 & & 10^{-1} & & 10^{-m} \end{array}$$

按权值展开：

$$\begin{aligned} V = & d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + \dots + d_1 \times 10^1 + d_0 \times 10^0 \\ & + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \dots + d_{-m} \times 10^{-m} \end{aligned}$$

$$= \sum_{i=-m}^{n-1} d_i \times 10^i$$

取值 权重

若用电子电路进行十进制数运算，必须要有**十个电路状态**与十个数码相对应。这样将在技术上带来许多困难，电路复杂，运算速度慢，而且很不经济。早期的模拟计算机就是如此。

R进制的定义

(2) **R进制** 对一n位整数和m位小数的R进制数为

$$\begin{array}{ccccccc} r_{n-1} & r_{n-2} & \cdots & r_1 & r_0 & . & r_{-1} \cdots r_{-m} \\ \updownarrow & \updownarrow & & \updownarrow & \updownarrow & & \updownarrow \\ R^{n-1} & R^{n-2} & & R^1 & R^0 & R^{-1} & R^{-m} \end{array}$$

则其按权展开式为

$$V = \sum_{i=-m}^{n-1} r_i \times R^i$$

对R进制数做加、减运算时遵循“逢**R**进一，借一当**R**”的原则。

二进制的定义

(3) **二进制(Binary)**: 以二为基数的记数体制
用两个数码表示:

0、1

遵循**逢二进一，借一当二**的规律

$$\begin{array}{r} 1\ 0\ 1\ 1 \\ 1\ 0\ 1\ 1\ \textcolor{red}{11} \\ +\ 1\ 0\ 0\ 1\ \textcolor{red}{9} \\ \hline 1\ 0\ 1\ 0\ 0\ \textcolor{red}{20} \end{array}$$

$$\begin{array}{r} 1\ 0\ 1 \\ 1\ 0\ 1\ 0\ \textcolor{red}{10} \\ -\ 0\ 1\ 0\ 1\ \textcolor{red}{5} \\ \hline 0\ 1\ 0\ 1\ \textcolor{red}{5} \end{array}$$

对一n位整数和m位小数的二进制数为

$$\begin{array}{ccccccc} b_{n-1} & b_{n-2} & \cdots & b_1 & b_0 & . & b_{-1} \cdots b_{-m} \\ \updownarrow & \updownarrow & & \updownarrow & \updownarrow & & \updownarrow \\ 2^{n-1} & 2^{n-2} & & 2^1 & 2^0 & & 2^{-1} \cdots 2^{-m} \end{array}$$

则其按权展开式为

$$\begin{aligned} V &= b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_1 \times 2^1 + b_0 \times 2^0 \\ &\quad + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \cdots + b_{-m} \times 2^{-m} \\ &= \sum_{i=-m}^{n-1} b_i \times 2^i = \sum_{i=-m}^{n-1} b_i \times W^i \end{aligned}$$

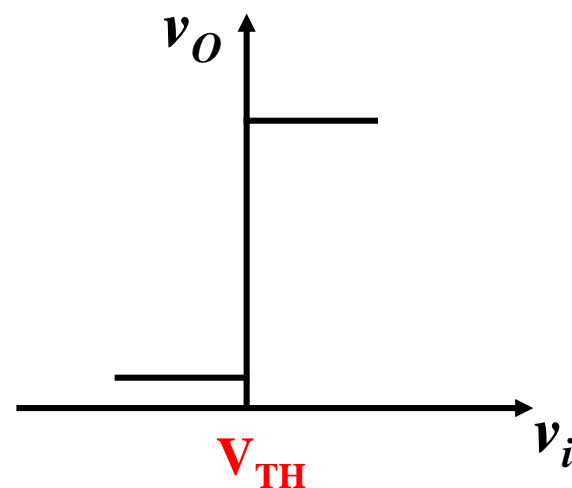
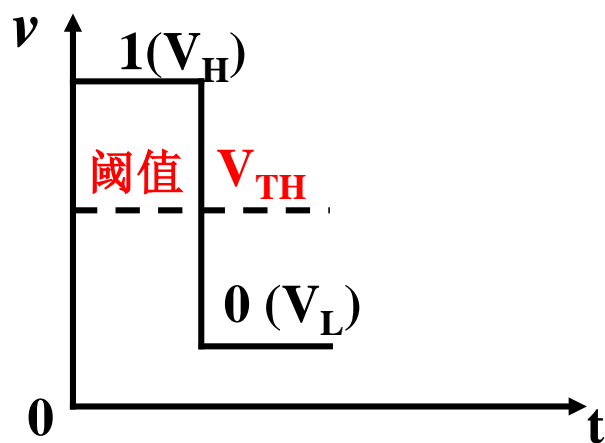
i 从-10到+10的权值表

i	权值	i	权值	i	权值	i	权值
-10	1/1024	-5	1/32	10	1024	5	32
-9	1/512	-4	1/16	9	512	4	16
-8	1/256	-3	1/8	8	256	3	8
-7	1/128	-2	1/4	7	128	2	4
-6	1/64	-1	1/2	6	64	1	2

二进制的优缺点



用电路的两个状态---高电平（1）和低电平（0）来表示二进制数，数码的产生，存储和传输简单、可靠。



二进制的优缺点（续）



需要的设备量少



运算规则简单



不符合人们的日常习惯，输入时将十进制转换成二进制，运算结果输出时再转换成十进制数。

二进制~十进制转换

(1) 二进制转换成十进制

例: 1011010.11

$$\begin{aligned}V &= 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 \\&\quad + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\&= 64 + 16 + 8 + 2 + 0.5 + 0.25 \\&= 90.75\end{aligned}$$

二进制数中所有数码1对应的权值相加

例：将二进制数11001.101转换成十进制

二进制中各个数码1对应的权值自左至右分别为16、8、1、0.5和0.125,将它们相加得到：

$$(11001.101)_b = (25.625)_d$$

所以转换的结果为 $16+8+1+0.5+0.125=25.625$

二进制~十进制转换（续）

(2) 十进制转换成二进制

* 原则：

整数：除基取余
小数：乘基取整

例1：将十进制整数27转换为二进制数。

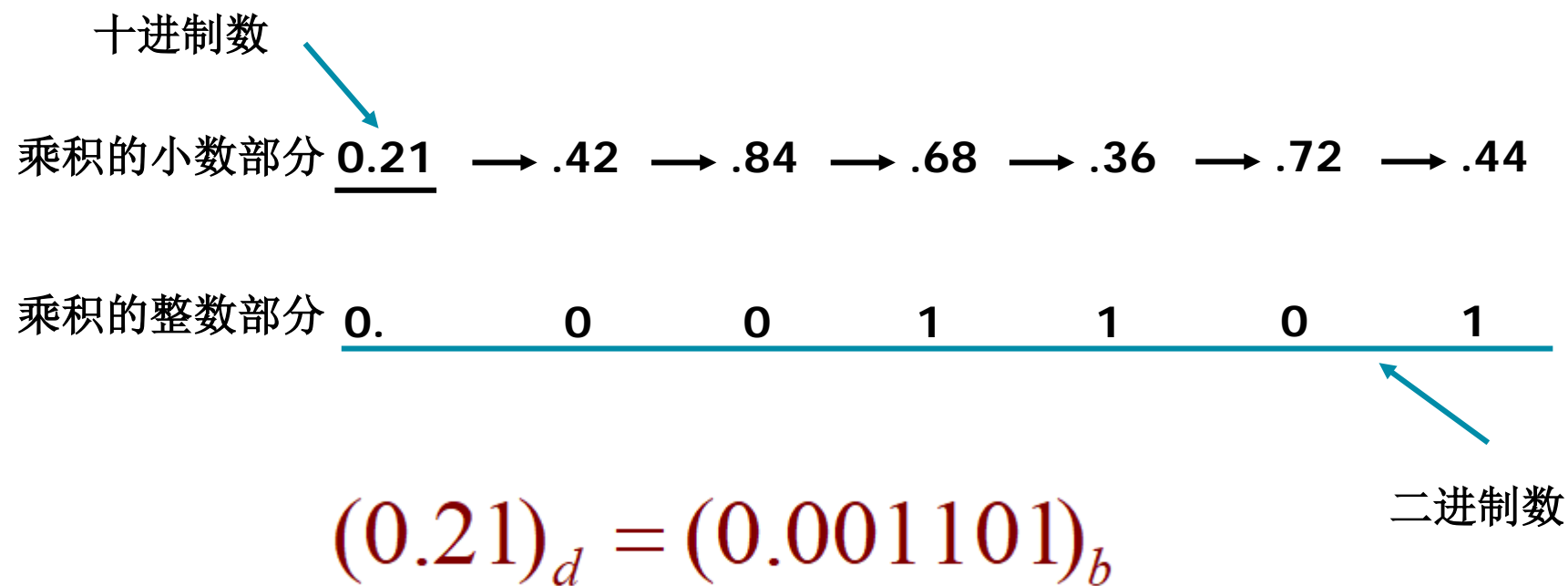
商	0	1	3	6	13	27
	←	←	←	←	←	
余数	1	1	0	1	1	

$$(27)_d = (11011)_b$$

十进制数

二进制数

例2 十进制小数0.21转换为二进制数，要求转换误差小于 2^{-6} 。



课堂习题

将下列二进制整数化成十进制数

(1) **1011010**

(2) **1000111**

将下列二进制小数化成十进制数

(1) **.1011**

(2) **.0101**

将下列二进制数化成十进制数

11011.111

课堂习题

将十进制整数化成二进制数

(1) 51 (2) 95

将十进制小数化成二进制数

(1) .5625 (2) .47

将十进制数化成二进制数

77.54

八进制与十六进制

二进制的缺点

二进制的缺点就是表示同一数字所需的位数多，
因此二进制对于书写和计算机输入是不方便的。

八进制(Octal): 逢八进一的进位制

$$(37.4)_o = 3 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (31.5)_d$$

$$\cdots b_{10} b_9 b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

$$2^9 \quad 2^6 \quad 2^3 \quad 2^0$$

$$8^3 \quad 8^2 \quad 8^1 \quad 8^0$$

八进制数码表

二进制数	八进制数码
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

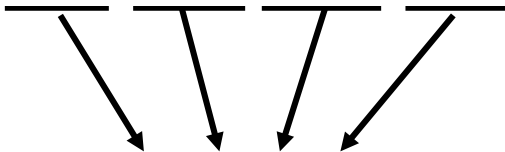
转换法则

二进制数转换为八进制数时以小数点为界，分别往高、往低每3位为一组，最后不足3位时用0补充然后写出每组对应的八进制字符，即为对应八进制数。

转换实例

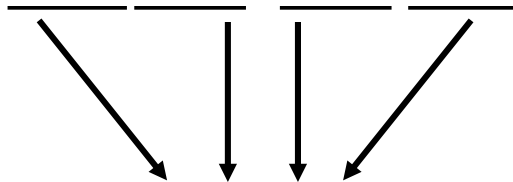
* 二进制数转换成八进制数

例1 $(100,110,010,001)_b$



$(4\ 6\ 2\ 1)_o$

例2 $(0.101,001,100,111)_b$



$(0.\ 5\ 1\ 4\ 7)_o$

转换实例

* 八进制数转换成二进制数

由八进制数转换为二进制数更为简单，只要将八进制的各位数码分别用对应的二进制数代入即可。

$$(351.25)_o = (011\ 101001.010\ 101)_b$$

十六进制

十六进制(Hexidecimal)定义

将二进制数从小数点向左、向右每4位作为一个单元，则每相邻两单元之间的关系为“逢十六进一”，如将每个单元用一个数符代表，这些数符便构成了一个十六进制数。

十六进制数码表

二进制数	十六进制数		二进制数	十六进制数
0000	0		1000	8
0001	1		1001	9
0010	2		1010	A
0011	3		1011	B
0100	4		1100	C
0101	5		1101	D
0110	6		1110	E
0111	7		1111	F

二进制~十六进制转换

* 二进制数转换成十六进制数

二进制数转换成十六进制数的是从小数点开始，分别向左和向右每四位作为一个单元，并分别用相应的十六进制数码代替。

例：将二进制数11000101.10101转换成十六进制数

$(1100, 0101.1010, 1000)_b$

$(C5.A8)_h$

(注意：小数点后面只有5位，必须加3个0凑成两个单元)

二进制~十六进制转换

* 十六进制数转换成二进制数

十六进制数转换成二进制数,即将十六进制的各位数码分别用对应的二进制数代入即可。

例：

$$(D3.1F)_h = (11010011.00011111)_b$$

课堂习题

例： 将二进制数化成八进制数

(1) 1100011

(2) .101101

(3) 1101.0011

将八进制数化成二进制数

(1) 271

(2) .32

(3) 35.26

将二进制数化成十六进制数

(1) 110010

(2) .001101

(3) 11010.11101

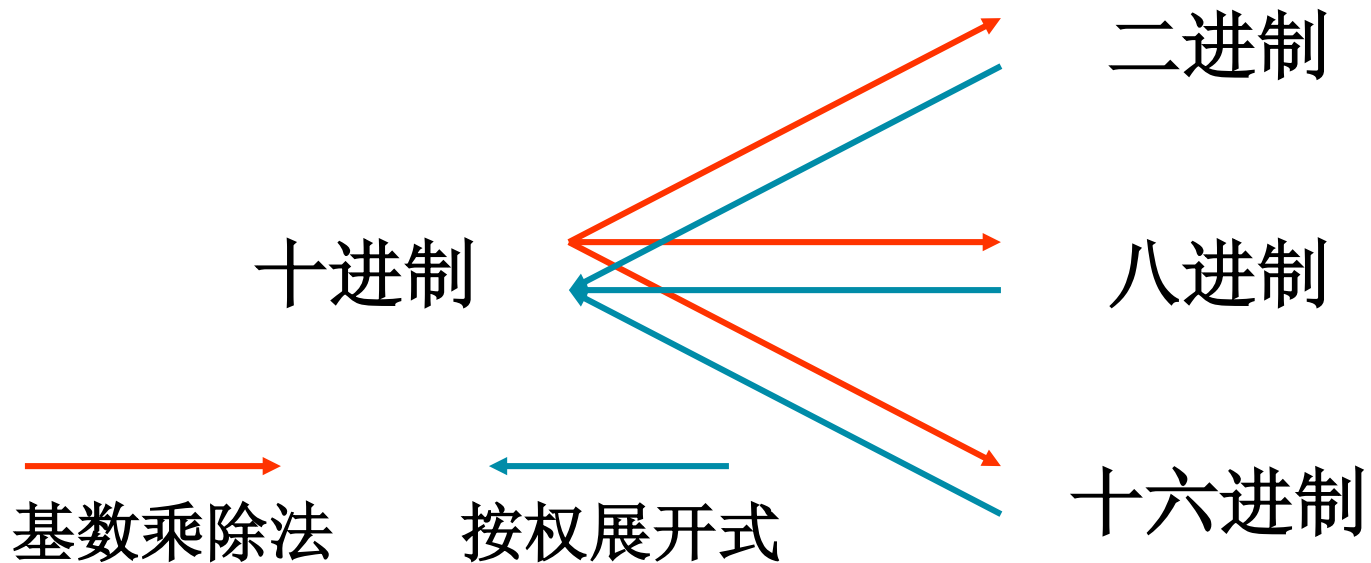
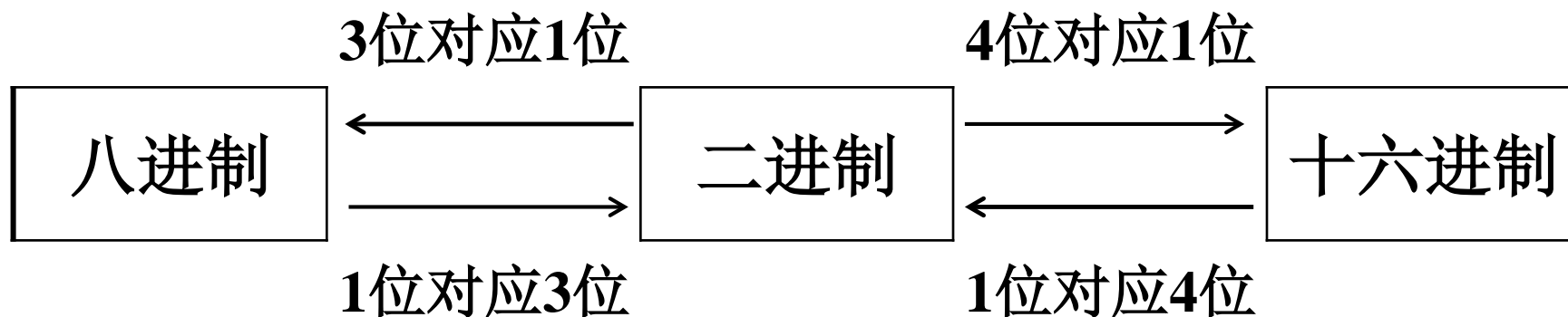
将十六进制数化成二进制数

(1) AE

(2) .E8

(3) 32.A6

数制转换小结



计算机中数的表示方法与格式

➤ 基本概念

- 字
- 字长
- 字节
- 码
- 码字
- 码元

实数在计算机中的表示

真值：真值是指在数值前面用“+”号表示正数,“-”号表示负数的带符号二进制数。

机器数：机器数是指在数字系统中用“0”表示符号为“+”,用“1”表示符号为“-”,即把符号“数值化”后的带符号二进制数。它有三种代码类型即原码、反码和补码。

实数的原码表示法



- * S是符号位(Sign)，通常用0表示符号+，用1表示符号-；
- * m是有效数字,也称尾数(Mantissa)。

例如

00001101 表示数+13；

10001101 表示数-13。

数字在计算机中的这种表示方法称为原码表示法。

原码的优缺点

- 原码的优点是容易理解。它和代数中的正负数的表示方法很接近。
- 原码的加法规则：
 - 先判断被加数和加数的符号是同号还是异号：
 - 同号时，做加法，结果的符号就是被加数的符号。
 - 异号时，先比较被加数和加数的数值(绝对值)的大小，然后由大值减去小值，结果的符号取大值的符号。
- 用上述规则设计加法器较复杂，这是原码的缺点。
- 为了简化加法器的设计，必须寻找其他表示负数的方法。这就是以下所讲的补码和反码。

实数的补码表示法

1. 补数(Complementary)的概念

如果A和A'两个数之和等于某个固定的数M（称为模），则称数A'是数A的关于模M的补数，或简称M数，即 $A = M - A'$
反之，亦然。

0~9关于10和9的补数

N	10的补数	9的补数
0	0	9
1	9	8
2	8	7
3	7	6
4	6	5
5	5	4
6	4	3
7	3	2
8	2	1
9	1	0

注：0关于模10的补数应为10，但10在十进制数中是2位数，此处只取1位，所以0关于模10的补数为0。

同余

同余的概念

$$A = A + n \times M \pmod{M}$$

补数的一种重要应用法则：

$$\begin{aligned} A - B &= A - B + M \pmod{M} && \text{(同余的性质)} \\ &= A + (M - B) \pmod{M} \\ &= A + B' \pmod{M} \end{aligned}$$

可见利用补数可以将加法和减法统一起来，使用同一套处理电路。

在使用n位二进制的计算机中，如不考虑符号位，则系统的模为 2^{n-1} ，所以一个数的补码表示为

S	$m' = 2^{n-1} + m$
---	--------------------

例如：模为10000000，
-0001101的补码为11110011；
+0001101的补码为00001101。

如果符号位与尾数一样参加运算，一个数N的补码的换算规则为

$$N' = 2^n + N$$

由于这种补码的模是2（对小数）或2的n次方，故称为2的补码或简称补码。它是一种以基数（的n次方）为模的补码，是一种真补码。

补码运算

例：在2的补码系统中完成二进制数+72与-13的加法运算。

解：

+72的补码为01001000

-13的补码为11110011

相加得：

01001000

+11110011

100111011



舍去 (+59)

二进制的R-1补码

补码运算虽然方便，求取补码却比较困难，因此计算机采用另一种称为基数减1补码（**R-1补码**），对二进制而言，**R=2**，故称**R-1**的补码为**1**的补码。

1的补码的格式与**2**的补码相同，只是模为 **2^n-1** 。当**n**为8时，模为**11111111**，即**255**。数**-0001101**的**1**的补码为**11110010**，比较原码与补码的尾数部分，我们可以发现，它们的每一位数码都是相反的，俗称“反码”。

与**2**的补码一样，正数的**1**的补码也与其原码相同。

反码运算

例1：在1的补码系统中完成二进制数+72与-13的加法运算。

解：+72和-13的1的补码分别为01001000和11110010， 将两数相加：

$$\begin{array}{r} 01001000 \\ +11110010 \\ \hline 100111010 \end{array} \quad (58)$$

若舍去溢出的1，结果为00111010（58），与本题的解不符，其原因是舍去的1代表 2^8 ，而不是反码系统的模 2^8-1 ，因此，如遇到这样的情况，还要将运算结果加上1，即

$$\begin{array}{r} 01001000 \\ +11110010 \\ \hline 100111010 \\ + \quad \quad \quad 1 \\ \hline 00111011 \end{array} \quad (59)$$

反码在补码转换运算中的作用

一个负数的2的补码恰好等于其1的补码加1，因此求某数的2的补码时，可以先求其1的补码，然后再加1。

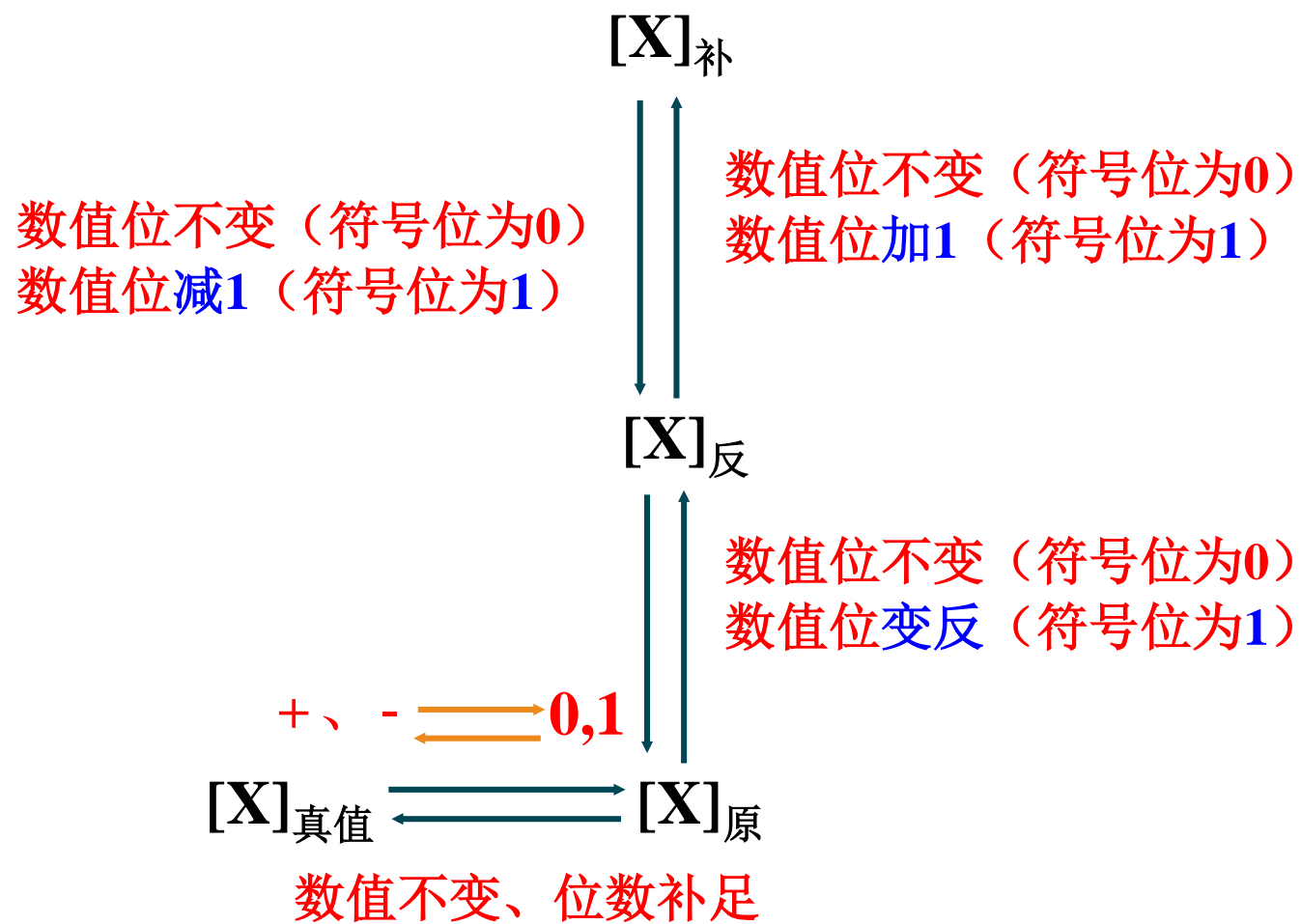
例2：求-31的2的补码

解：-31的原码为 $(10011111)_p$ ，将其尾数各

位变反得到其1的补码为 $(11100000)_{1's}$ ，再将

加上1，得到其2的补码为 $(11100001)_{2's}$ 。

转换法则小结



二进制数加减法的规则总结

数制	加法规则	变负规则	减法规则
无符号数	操作数相加；如果MSB产生进位，则结果超出范围。	不适用	被减数减去减数；如果MSB产生借位，则结果超出范围。
符号-数值 (原码)	(同号) 绝对值相加；符号与操作数相同；如果MSB产生进位，则溢出。 (异号) 较大绝对值减去较小绝对值，结果符号同绝对值较大数；不可能溢出。	改变操作数符号位	改变减数的符号后，同加法一样进行计算。
二进制补码	做加法；忽略MSB的进位；如果向符号位的进位输入与从符号位的进位输出不同，则产生溢出。	逐位取反末位加1	减数逐位取反，再与被减数相加，令初始进位为1。
二进制反码	做加法；如果MSB有进位，结果加1；如果向符号位的进位输入与从符号位的进位输出不同，则产生溢出。	逐位取反	减数逐位取反，同加法一样进行计算。

课堂练习

将下列二进制数用8位原码表示

(1) 11011 (2) -111100

将下列二进制数用8位补码表示

(1) 10101 (2) -1110111

将下列二进制数用8位反码表示

(1) 10011 (2) -1110111

课堂练习

写出与下列二进制原码对应的补码和反码

(1) 00110110 (2) 10101001

写出与下列二进制补码对应的反码和原码

(1) 01001110 (2) 11001111

写出与下列二进制反码对应的补码和原码

(1) 00011101 (2) 11101000

定点数

通常可将小数点的位置约定在数码的最右边或在其尾数的最左边。

若

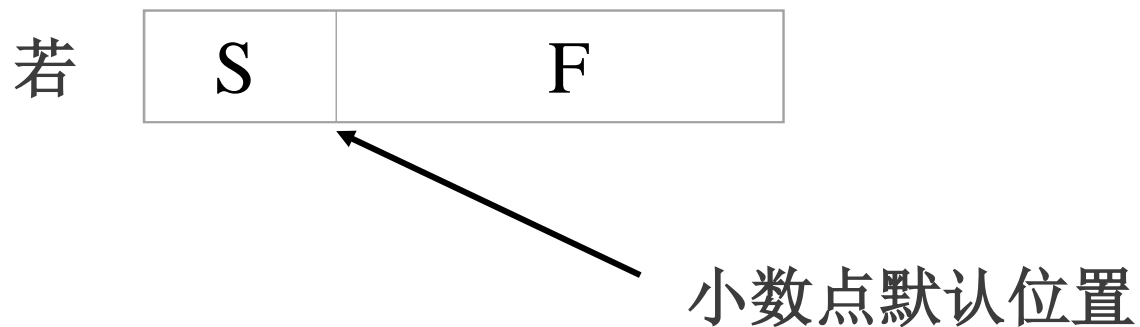


小数点默认位置

其值为 $(-1)^S \times I$ 显然，这种格式只能表示整数。
可以表示的范围为 $\left[-(2^{n-1}-1) \sim +(2^{n-1}-1) \right]$

可以达到的精度为 $1/2^{n-1}$

定点数（续）



其值为 $(-1)^S \times 0.F$

显然，这种格式只能表示小数。

可以表示的范围为 $\left[-\left(1 - 2^{-(n-1)}\right) \sim +\left(1 - 2^{-(n-1)}\right) \right]$

可以达到的精度为 $1/2^{n-1}$

浮点数

浮点数对应于数字的科学表示法，其通用格式为：



小数点默认位置

注：S是实数的符号，I是尾数，C则称为阶码，SC是阶码的符号。

此格式表示的数为 $V = (-1)^S \times I \times 2^{(-1)^{SC} \times C}$

设一个8位浮点数01011000（原码），其中

SC=0,C=2,S=1,I=8,则表示的数为 $-8 \times 2^{+2} = -32$ 。

浮点数（续）

浮点数也可以将小数点放在尾数的最左边，如下所示：



小数点默认位置

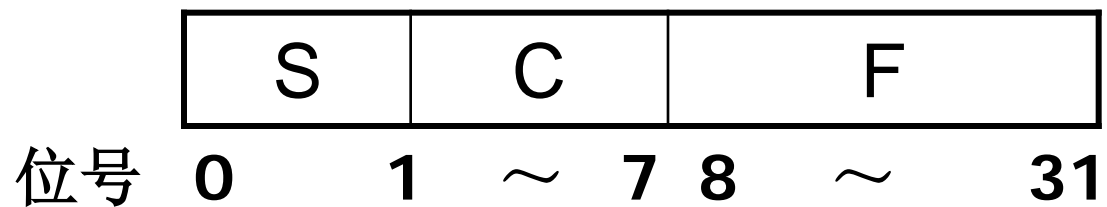
其值为 $V = (-1)^S \times 0.F \times 2^{(-1)^{SC} \times C}$

同样的数01011000（原码）在此格式下表示数字
 $-0.5 \times 2^{+2} = -2$

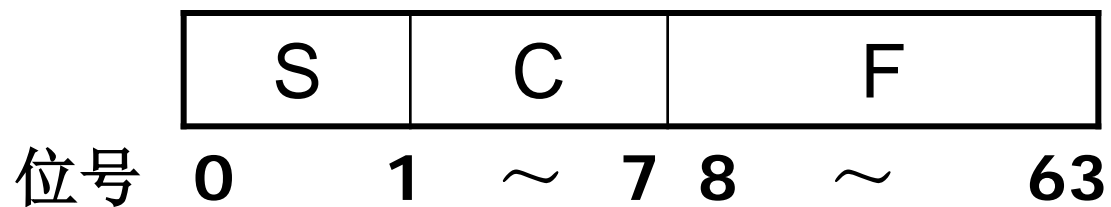
一般来说，浮点数所能表示的数的范围比定点数大的多；在同样字长且二者有效位都被充分利用的情况下，精度要低一些。

IBM浮点数格式

32位字长



64位字长



$$V = (-1)^S \times 0.F \times 16^{C-64}$$

例：求32位IBM格式表示的数据(41E00000)_h和(C1E00000)_h的数值

先将十六进制数(41E00000)₁₆转换成二进制数

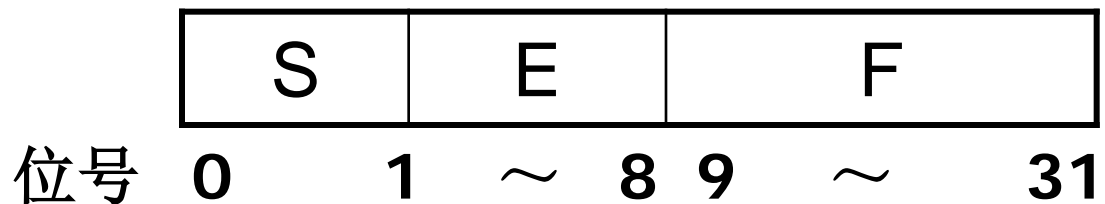
0 100,0001,1110,0000,0000,0000,0000,0000

S(0) C(65) F(0.875)

$$\begin{aligned} V &= (-1)^S \times 0.F \times 16^{C-64} \\ &= (-1)^0 \times 0.875 \times 16^{65-64} \\ &= 0.875 \times 16 \\ &= 14 \end{aligned}$$

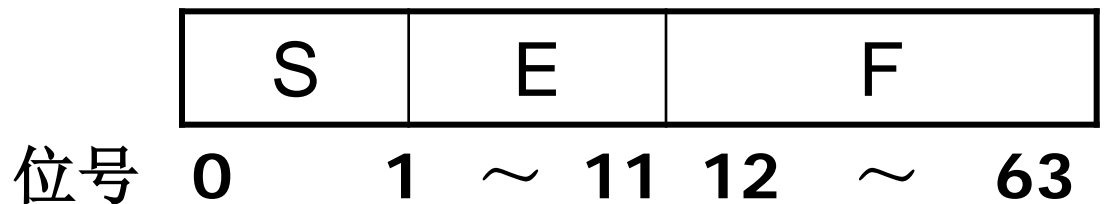
IEEE浮点数格式

32位字长



$$V = (-1)^S \times 1.F \times 2^{E-127}$$

64位字长



$$V = (-1)^S \times 1.F \times 2^{E-1023}$$

例：求32位IEEE格式表示的数据(3F800000)_h和64位IEEE格式表示的数据(C03E000000000000)_h的数值

先将十六进制数(3F800000)₁₆转换成二进制数

0 011,1111,1000,0000,0000,0000,0000,0000

S(0) E(127) F(0.0)

$$\begin{aligned} V &= (-1)^S \times 1.F \times 2^{E-127} \\ &= (-1)^0 \times 1.0 \times 2^{127-127} \\ &= 1.0 \times 2^0 \\ &= 1 \end{aligned}$$

将十六进制数(C03E000000000000)_h转换成二进制数

1 100,0000,0011,1110,0000,0000,0000,0000

↓ ↓ ↓

S(1) E(1027) F(0.875)

$$\begin{aligned} V &= (-1)^S \times 1.F \times 2^{E-1023} \\ &= (-1)^1 \times 1.875 \times 2^{1027-1023} \\ &= -1.875 \times 2^4 \\ &= -30 \end{aligned}$$

十进制数的表示方法

在数字系统的输入输出中，为了既满足系统中使用二进制数的要求，又适应人们使用十进制数的习惯，通常使用4位二进制代码对十进制数字符号进行编码，简称为二-十进制代码，或称BCD（**B**inary **C**oded **D**ecimal）码。

数字系统中常用的BCD码（BCD码不是二进制数，而是用二进制编码的十进制数）有8421码、2421码和余3码。

常用BCD码

十进制数 N	NBCD(8421)码	余3码	2421码	循环码
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 0 0	0 0 0 1	0 0 0 1
2	0 0 1 0	0 1 0 1	0 0 1 0	0 0 1 1
3	0 0 1 1	0 1 1 0	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 1 0
5	0 1 0 1	1 0 0 0	1 0 1 1	0 1 1 1
6	0 1 1 0	1 0 0 1	1 1 0 0	0 1 0 1
7	0 1 1 1	1 0 1 0	1 1 0 1	0 1 0 0
8	1 0 0 0	1 0 1 1	1 1 1 0	1 1 0 0
9	1 0 0 1	1 1 0 0	1 1 1 1	1 1 0 1

四种BCD码都是用四位二进制代码表示一位十进制数字。
四种BCD码与十进制数之间的转换是以四位对应一位, 直接进行变换, 一个n位十进制数对应的BCD码一定为4n位。

有权码

有权码(weighted code): 例,8421码,2421码。其特点是,当知道权值和代码时,就可计算出它代表的十进制值。

8421码的0111, 它代表 $0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7$

$$(a_3 a_2 a_1 a_0)_{8421\text{码}} = (8a_3 + 4a_2 + 2a_1 + a_0)_{10}$$

2421码的1110, 它代表 $1 \times 2 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 8$

$$(a_3 a_2 a_1 a_0)_{2421\text{码}} = (2a_3 + 4a_2 + 2a_1 + a_0)_{10} ;$$

8421码的优缺点

😊 编码值与字符从0到9的ASCII码的低4位相同，有利于简化输入输出过程从字符→BCD和从BCD→字符的转换操作，是实现人机联系时比较好的中间表示

😊 需要译码时，译码电路比较简单

😞 实现加减运算的规则比较复杂，在某些情况下需要对运算结果进行修正

余3码和循环码的优缺点

余三码的优缺点：

- 😊 执行十进制数相加时可以正确的产生进位信号，而且还能给减法运算带来方便
- 😞 实现加减运算的规则比较复杂，在某些情况下需要对运算结果进行修正

循环码（Gray码）的优缺点：

- 😊 属于安全码
- 😞 需要码表转换

课堂练习

例：

用8421BCD码表示下列各数

(1) 248 (2) 359

将下列用NBCD码表示的数还原为十进制数

(1) 100101010011 (2) 01111000

分别用2421码和循环码表示下列各十进制数

(1) 25 (2) 93

- 最常用的字符代码是ASCII码
 - ASCII码：每个字符是用7位二进制码表示。
是128个字符组成的字符集。其中32个控制字符,然后是空格,然后是数字,大写字母,小写字母。
 - ①数字0~9,高3位是011,低4位是0000~1001所以和二进制间进行转换很容易。
 - ②大小写字母之间进行转换也很容易,因为只是 b_5 的不同,例A的编码为1000001,而a的编码为1100001。

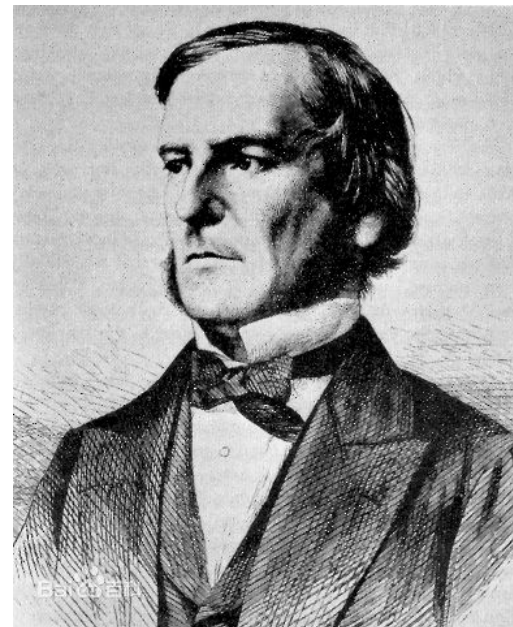
布尔及布尔代数

➤ 布尔 (George Boole)

- 1815.11~1864，英格兰人，19世纪最重要的数学家之一
- 1847年出版了《逻辑的数学分析》，第一次对符号逻辑贡献
- 1854年出版了《思维规律的研究》，全面介绍了逻辑代数

➤ 布尔代数 (逻辑代数)

- 把逻辑简化成极为容易和简单的一种代数
- 对适当材料的“推理”，转化为公式表示的初等运算
- 如今布尔发明的逻辑代数已经成为纯数学的一个主要分支



逻辑代数系统

逻辑代数是一个由逻辑变量集 K ，常量0和1以及“与”、“或”、“非”3种基本运算构成的一个封闭的代数系统，记为 $L=\{K, +, \cdot, -, 0, 1\}$ 。它是一个二值代数系统。常量0和1表示真和假，无大小之分。

逻辑代数基本运算

➤ 基本运算

- 非逻辑运算
- 与逻辑运算
- 或逻辑运算

➤ 常用组合运算

- 与非、或非
- 异或、同或

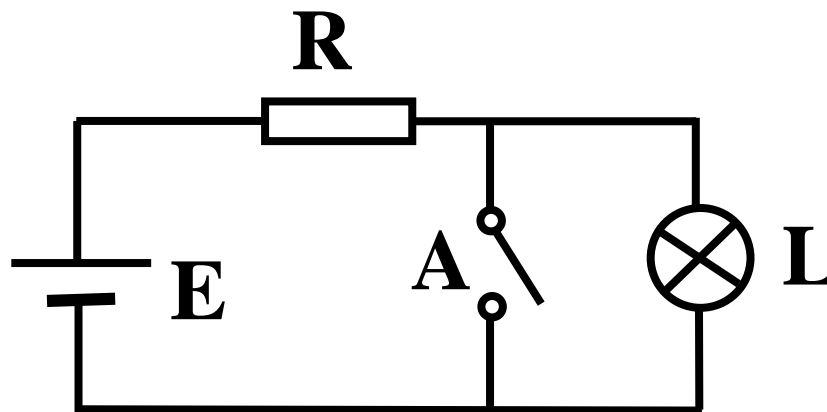
非逻辑的定义

非逻辑：决定事件发生的条件只有一个，条件不具备时事件发生（成立），条件具备时事件不发生。

非逻辑真值表

A	$L = A$
0	1
1	0

特点：1则0，0则1

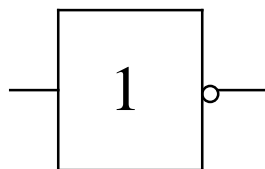


其函数表达式为：

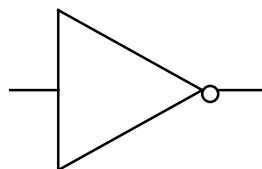
$$L = f(A) = \overline{A}$$

逻辑符号：

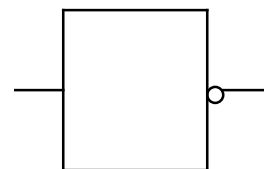
(a) 国标GB4728.12-85符号



(b) MIL符号



(c) 原部标SJ1223-77符号

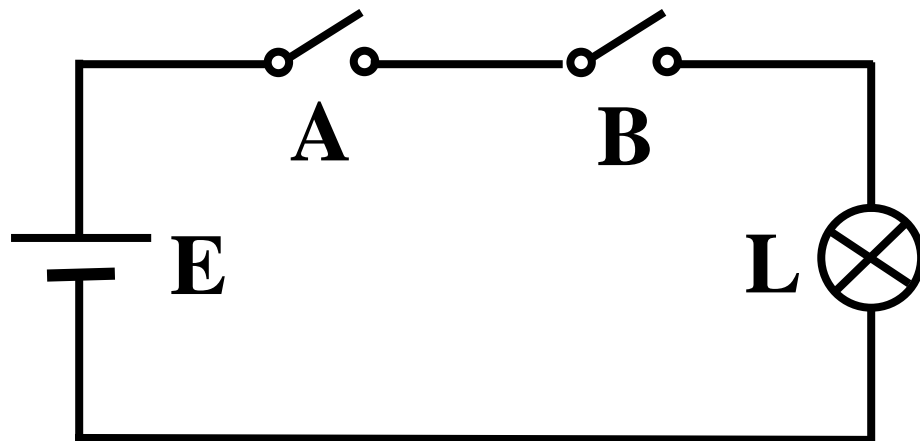


与逻辑的定义

与逻辑：决定事件发生的各条件中，所有条件都具备，事件才会发生（成立）。

与逻辑真值表

A	B	$L=A \times B$
0	0	0
0	1	0
1	0	0
1	1	1



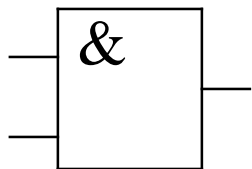
特点： 任0 则0, 全1则1

逻辑表达式:

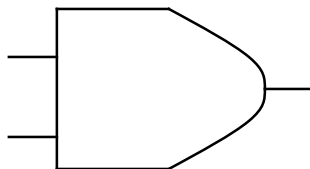
$$L = A \cdot B \quad \text{或} \quad L = A B$$

逻辑符号:

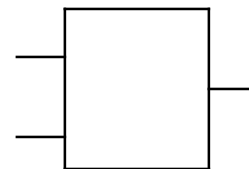
(a) 国标GB4728.12-85符号



(b) MIL符号



(c) 原部标SJ1223-77符号

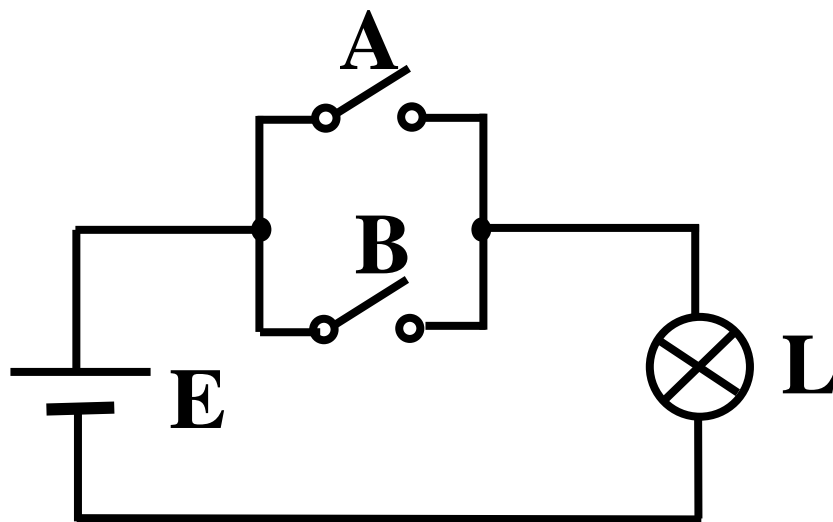


或逻辑的定义

或逻辑：决定事件发生的各条件中，有一个或一个以上的条件具备，事件就会发生（成立）。

真值表

A	B	$L=A+B$
0	0	0
0	1	1
1	0	1
1	1	1



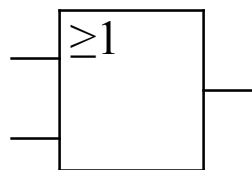
特点:任1 则1, 全0则0

逻辑表达式

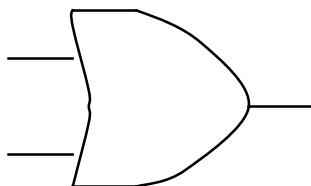
$$L = A + B$$

逻辑符号：

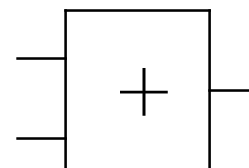
(a) 国标GB4728.12-85符号



(b) MIL符号



(c) 原部标SJ1223-77符号



逻辑运算优先级

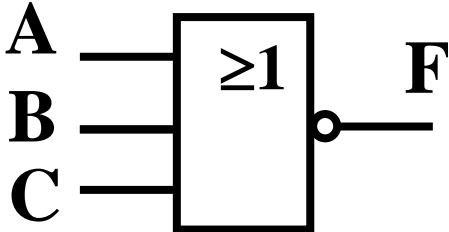
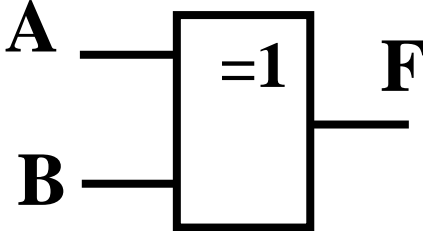
以上三种基本逻辑运算如在逻辑运算式中同时出现时，其优先顺序为：**非**>**与**>**或**，必要时还可用括号加以提前。

几种常用的逻辑关系

“与”、“或”、“非”是三种基本的逻辑关系，任何其它的逻辑关系都可以以它们为基础表示。

<p>与非： 条件A、B、C都具备，则F不发生。</p>	$F = \overline{A \bullet B \bullet C}$ <p>任0则1，全1则0</p>	
-------------------------------------	---	---

几种常用的逻辑关系（续）

<p>或非： 条件 A、B、C任一 具备，则F不 发生。</p>	$F = \overline{A + B + C}$ <p>任1则0，全0则1</p>	
<p>异或： 条件 A、B有一个具 备，另一个不 具备则F 发生 。</p>	$F = A\overline{B} + \overline{A}B$ $= A \oplus B$	

逻辑代数的基本定律

逻辑非、逻辑乘、逻辑加的基本运算规则

公理1 如 $A \neq 1$, 则 $A=0$

如 $A \neq 0$, 则 $A=1$

公理2 $\overline{0} = 1$

$\overline{1} = 0$

公理3 $0 \cdot 0 = 0$

$1 + 1 = 1$

公理4 $0 \cdot 1 = 1 \cdot 0 = 0$

$$1+0=0+1=1$$

公理5 $1 \cdot 1 = 1$

$$0+0=0$$

交换律 $A+B=B+A$

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

结合律 $A+(B+C)=(A+B)+C$

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

分配律

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

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

普通代数不适用!

如何证明?

控制律

$$A \cdot 0=0$$

$$A+1=1$$

自等律

$$A \cdot 1=A$$

$$A+0=A$$

重叠律

$$A \cdot A = A$$

$$A + A = A$$

吸收律

$$A + AB = A$$

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

互补律

$$A \cdot \overline{A} = 0$$

$$A + \overline{A} = 1$$

反演律

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

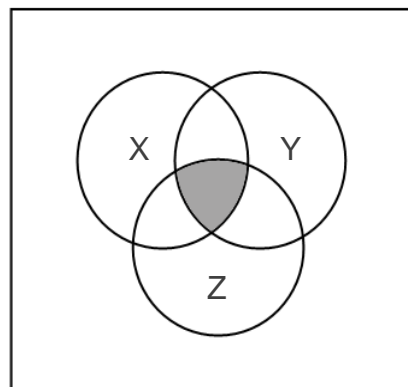
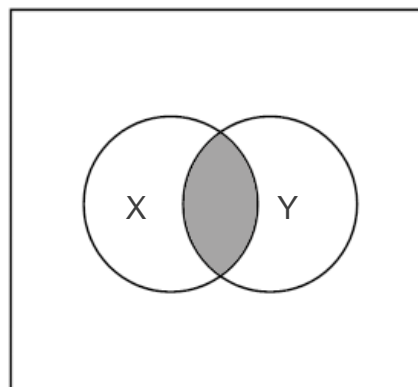
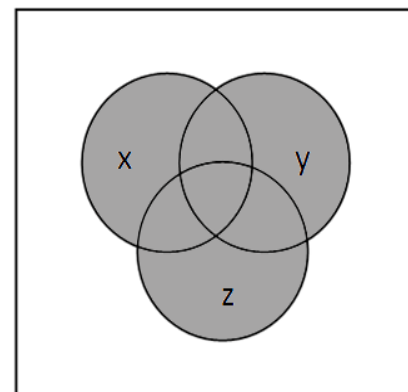
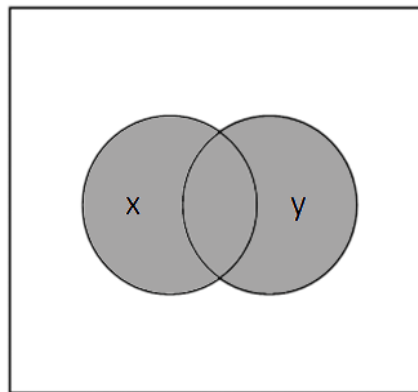
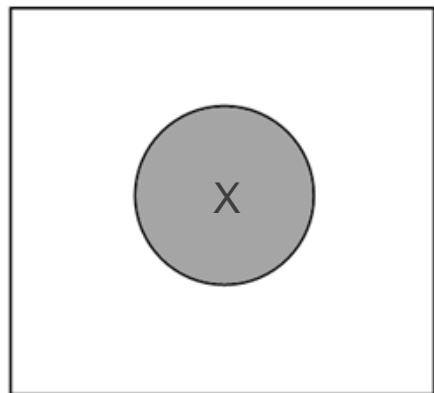
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

摩根定律，在函数求反
及与或变换时很有用

双重否定律

$$\overline{\overline{A}} = A$$

文氏图表示法（可用于验证布尔定律）



课堂练习

例：用逻辑代数的公理或定律证明下列等式

$$1 \quad AB + \overline{A}C + \overline{B}C = AB + C$$

$$2 \quad \overline{A\overline{B} + B\overline{C} + C\overline{A}} = ABC + \overline{A} \cdot \overline{B} \cdot \overline{C}$$

逻辑代数的基本规则（一）

（1）置换(Replacement)规则

置换规则表明，对于逻辑等式中的任一变量**X**，若将所有出现它的地方都用逻辑函数**G**置换，等式仍然成立。

例如 表达式 $A + ABC\overline{C}(D + E)$ 中的 $B\overline{C}(D + E)$ 一个字母**B** 所置换，可利用吸收律 $A + AB = A$ 将原式简化为A，即

$$A + ABC\overline{C}(D + E) \rightarrow A + AB = A$$

必须对等式两边所有的变量施行

逻辑代数的基本规则（二）

（2）对偶(Dual)规则

所有逻辑常量和逻辑符号分别作1与0、+与·的对换

注意：

- * 变换必须对所有的逻辑变量、逻辑符号施行，不能遗漏。
- * 必须保持原函数变量之间的运算顺序不变

逻辑代数的基本规则（三）

（3）反演(Invert)规则

所谓反演就是求一个函数 F 的反函数 \bar{F} ,所以反演规则也称求反规则或求补规则。

逻辑常量、逻辑变量和逻辑符号分别作0与1、+与 \cdot 、 X_i 与 \bar{X}_i 之间的变换

例：求逻辑函数 $F=(X_1\bar{X}_2+X_3)X_4$ 的反函数 \bar{F} 。

注意事项

在使用反演规则时应注意：

- (1) 上述变换必须对所有的逻辑常量、逻辑符号和逻辑变量施行，不能遗漏；
- (2) 必须保持原函数变量之间的运算顺序不变，必要时可添加括号；
- (3) x_i 与 $\overline{x_i}$ 之间的互换只对逻辑变量有效
- (4) 不属于单个变量上的反号应保留不变

注意事项

$F = \overline{AB} + \overline{C}(D + A)$ 中 \overline{AB} 属于非运算；不是反变量，因而在反演时，非号须保留，该函数的反函数是

$$\overline{F} = \overline{\overline{A} + \overline{B}}(C + \overline{D} \cdot \overline{A}) \quad \text{而不是} \quad \overline{F} = (\overline{A} + \overline{B})(C + \overline{D} \cdot \overline{A})$$

注意： $F + \overline{F} = 1$

而F与F'是两个相互独立的函数，只是形式上的对偶。

课堂练习

例：求下列逻辑函数的反函数 \overline{F} 和对偶函数 F'

$$1 \quad F = \overline{A} \cdot \overline{B} + \overline{C} \overline{D}$$

$$2 \quad F = \overline{\overline{A} + \overline{B} + C}$$

逻辑代数的常用公式

(1) 并项公式

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

证明:

$$AB + \overline{A}B = (A + \overline{A})B$$

$$= 1 \cdot B$$

$$= B$$

(分配律)

(互补律)

(自等律)

(2) 消冗余因子公式

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

(3) 消冗余项公式

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

推论:

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

(4) 消冗余因子公式

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

$$\overline{A} \cdot \overline{AB} = \overline{A}$$

逻辑函数及其描述方法

在逻辑代数中，任何对 n 个逻辑变量 x_1, x_2, \dots, x_n 进行有限次逻辑运算的逻辑表达式，称为 n 变量的逻辑函数或简称函数，记作： $F=f(x_1, x_2, \dots, x_n)$ 。

五种表示方法

- 逻辑表达式
- 逻辑图
- 真值表
- 卡诺图
- 标准表达式

逻辑函数描述示例

例：举重比赛有三名裁判，当运动员将杠铃举起后，须有两名或两名以上裁判认可，方可判定试举成功，若用字母A、B、C分别代表三名裁判的意见，同意为1，否定为0；F为裁判结果，试举成功时 $F=1$ ，试举失败时 $F=0$ 。则F与A、B、C之间的关系可以用以下几种方式表示。

逻辑函数表达式的定义

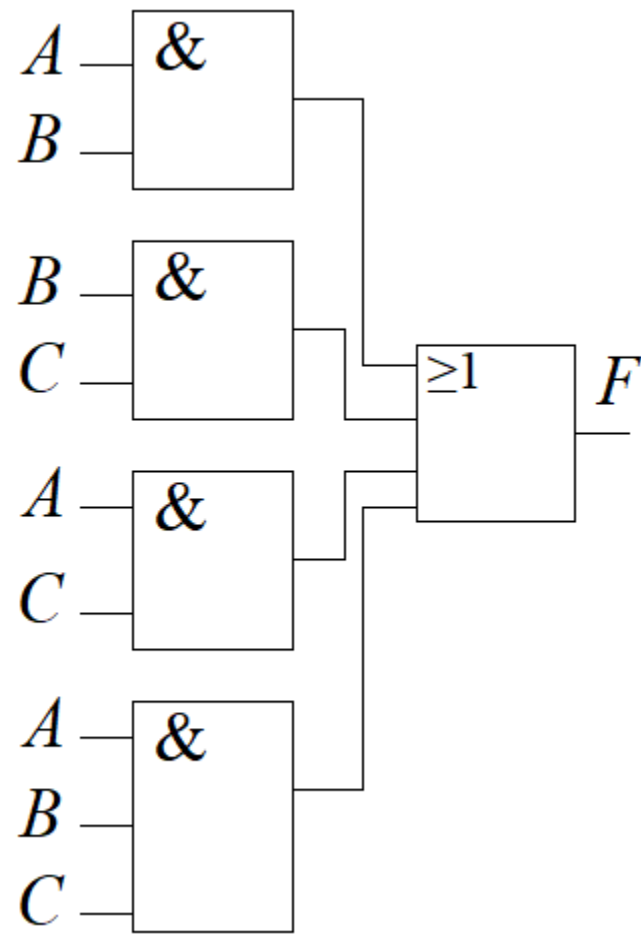
把逻辑函数的输入、输出关系写成与、或、非等逻辑运算的组合式，即逻辑代数式，又称为逻辑函数式。

逻辑函数表达式例题描述

在上例中，可用**AB**代表“裁判A、B都同意”，同理用**BC**和**AC**表示另两种有两名裁判同意的情况，**ABC**表示三名裁判都同意，所以

$$\mathbf{F=f(A,B,C)=AB+BC+AC+ABC}$$

逻辑图例题描述



真值表例题描述

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

唯一性

N个输入变量 \longrightarrow 2^n 种组合。

A	F
0	1
1	0

一输入变量，二种组合

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

二输入变量，四种组合

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

三输入变量，八种组合

卡诺图例题描述

在卡诺图中变量组合按照循环邻接的原则进行排列

输入变量

A \ BC				
	00	01	11	10
0			1	
1		1	1	1

输出变量 F 的值

AB \ C		
	0	1
00	0	0
01	0	1
11	1	1
10	0	1

卡诺图说明

卡诺图的每一个方块代表一种输入组合
对应的输入组合注明在阵列图的上方和左方。

A \ BC				
	00	01	11	10
0			1	
1		1	1	1

逻辑相邻：相邻单元输入变量的取值只能有一位不同

唯一性

AB \ CD				
	00	01	11	10
00	1	1		1
01	1	1		1
11	1		1	1
10	1			1

卡诺图说明（续）

用二进制对应的十进制数表示单元编号

		BC			
		00	01	11	10
A	0	m ₀	m ₁	m ₃	m ₂
	1	m ₄	m ₅	m ₇	m ₆

		CD			
		00	01	11	10
AB	00	m ₀	m ₁	m ₃	m ₂
	01	m ₄	m ₅	m ₇	m ₆
	11	m ₁₂	m ₁₃	m ₁₅	m ₁₄
	10	m ₈	m ₉	m ₁₁	m ₁₀

真值表例题描述

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

唯一性

$AB\bar{C}=1, F=1$

$A=1 \ B=1 \ \bar{C}=0$

$F=1 \cdot AB\bar{C}$

标准与-或表达式例题描述

$$\begin{aligned} F &= 0 \cdot \overline{A}\overline{B}\overline{C} + 0 \cdot \overline{A}\overline{B}C + 0 \cdot \overline{A}B\overline{C} + 1 \cdot \overline{A}BC + 0 \cdot A\overline{B}\overline{C} + 1 \cdot A\overline{B}C + \\ &\quad 1 \cdot AB\overline{C} + 1 \cdot ABC \\ &= \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \end{aligned}$$

注：由真值表直接推出，因而也具有唯一性，故而称为逻辑函数的**标准与-或表达式**。通常将这种由真值表推出的各个与项称为函数的**最小项(Minterm)**，并用符号 m_i 表示，其下标 i 就是真值表中对应行的坐标或者说卡诺图对应块的坐标。

三变量函数的最小项

变量坐标	最小项	最小项符号	函数F
0 0 0	$\overline{A}\overline{B}\overline{C}$	m0	f0
0 0 1	$\overline{A}\overline{B}C$	m1	f1
0 1 0	$\overline{A}B\overline{C}$	m2	f2
0 1 1	$\overline{A}BC$	m3	f3
1 0 0	$A\overline{B}\overline{C}$	m4	f4
1 0 1	$A\overline{B}C$	m5	f5
1 1 0	$AB\overline{C}$	m6	f6
1 1 1	ABC	m7	f7

最小项的重要性质

- 在输入变量的任何取值下必有一个最小项，而且只有一个最小项的值为1
- 全体最小项之和为1
- 任意两个最小项的乘积为0
- 具有相邻性的两个最小项之和可以合并成一项并消去一对因子

$$F = 0 \cdot \overline{A}\overline{B}\overline{C} + 0 \cdot \overline{A}\overline{B}C + 0 \cdot \overline{A}B\overline{C} + 1 \cdot \overline{A}BC + 0 \cdot A\overline{B}\overline{C} + 1 \cdot A\overline{B}C + 1 \cdot AB\overline{C} + 1 \cdot ABC$$

$$\begin{aligned} F &= 0 \cdot m_0 + 0 \cdot m_1 + 0 \cdot m_2 + 1 \cdot m_3 + 0 \cdot m_4 + 1 \cdot m_5 + 1 \cdot m_6 + 1 \cdot m_7 \\ &= m_3 + m_5 + m_6 + m_7 \\ &= \sum m(3, 5, 6, 7) \end{aligned}$$

推广：一个n变量函数有 2^n 个最小项，每个最小项是n个变量构成的积项，每个变量在此与项中只且出现一次。函数F的标准表达式就是这些最小项与对应的函数值乘积的总和，即

$$F = \sum_{i=0}^{2^n-1} f_i \cdot m_i$$

最大项与“或-与”表达式

$$F = \sum m_i$$

则 $\sum m_i$ 以外的那些最小项之和必为 \overline{F}

$$\overline{F} = \sum_{k \neq i} m_k$$

$$F = \overline{\sum_{k \neq i} m_k}$$

反演



$$F = \prod_{k \neq i} \overline{m_k} = \prod_{k \neq i} M_k$$

一个逻辑函数可以表示为若干最大项的乘积，与最小项之和表达式一样，最大项之积表达式也是唯一的，所以也称为标准或-与式。

三变量函数的最大项

变量坐标	最大项	最大项符号	函数F
0 0 0	A+B+C	M₀	f₀
0 0 1	A+B+\bar{C}	M₁	f₁
0 1 0	A+\bar{B}+C	M₂	f₂
0 1 1	A+\bar{B}+\bar{C}	M₃	f₃
1 0 0	\bar{A}+B+C	M₄	f₄
1 0 1	\bar{A}+B+\bar{C}	M₅	f₅
1 1 0	\bar{A}+\bar{B}+C	M₆	f₆
1 1 1	\bar{A}+\bar{B}+\bar{C}	M₇	f₇

$$M_i = \overline{m_i}$$

$$F = \sum_i m_i \quad (i = 3, 5, 6, 7)$$

$$F = \prod_{k \neq i} \overline{m_k} = \prod_{k \neq i} M_k = M_0 M_1 M_2 M_4$$

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

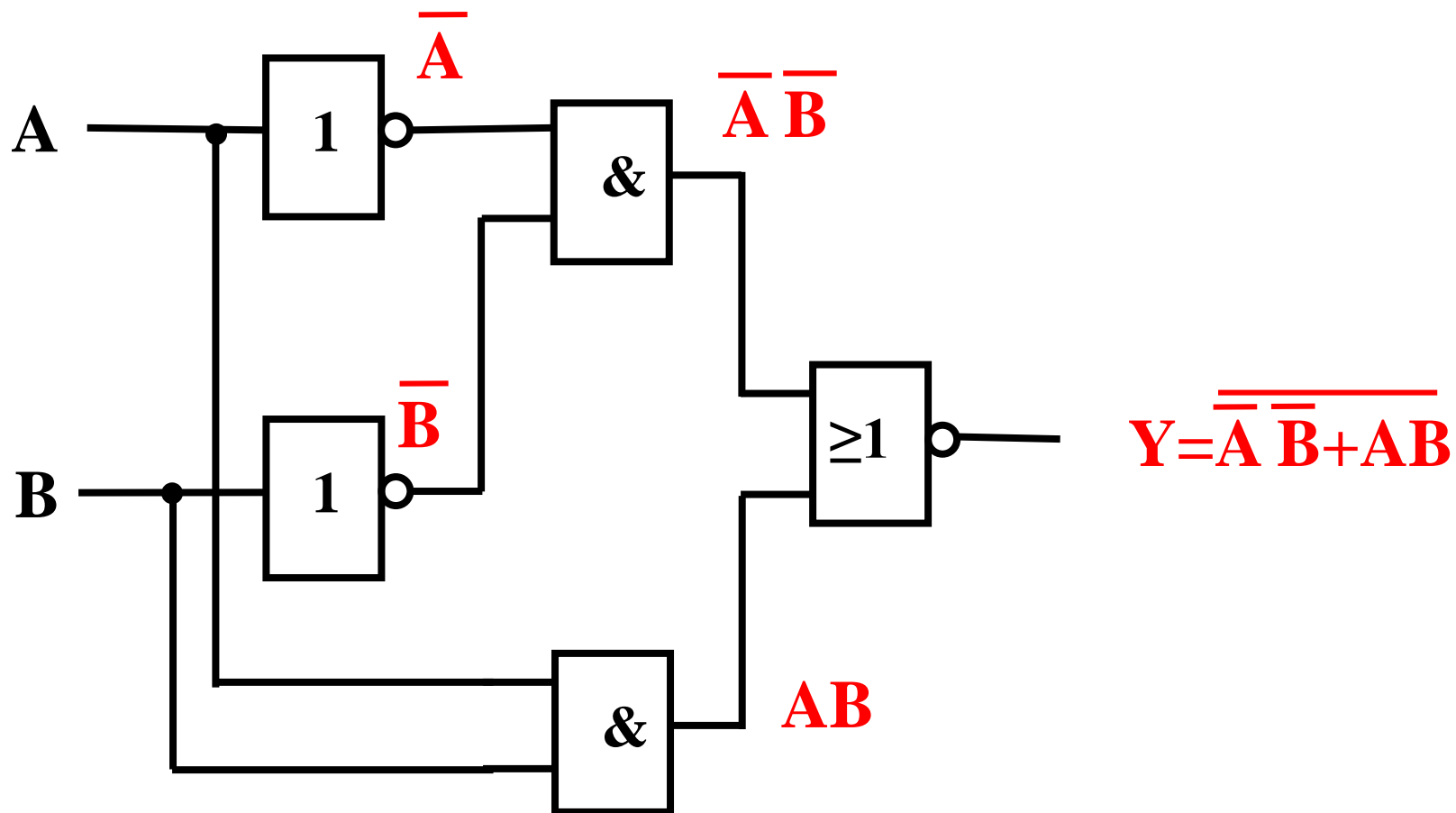
$$\begin{aligned}
 F &= (0 + M_0)(0 + M_1)(0 + M_2)(1 + M_3)(0 + M_4)(1 + M_5)(1 + M_6)(1 + M_7) \\
 &= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \\
 &= (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)
 \end{aligned}$$

从函数的真值表或卡诺图求其标准**或—与**式的方法有：

- (1) 求得反函数的标准**与—或**表达式，对其求反。
- (2) 找出函数真值表中所有等于0的行，根据每行的坐标求得对应的最大项（方法是，坐标变量为1的取反变量，坐标变量为0的取原变量，然后相加），将这些最大项相**与**。

逻辑函数表示方法间的相互转化

1、逻辑图↔逻辑表达式



例：已知逻辑函数为 $F = \overline{A + \overline{BC}} + \overline{A}B\overline{C} + C$ ，

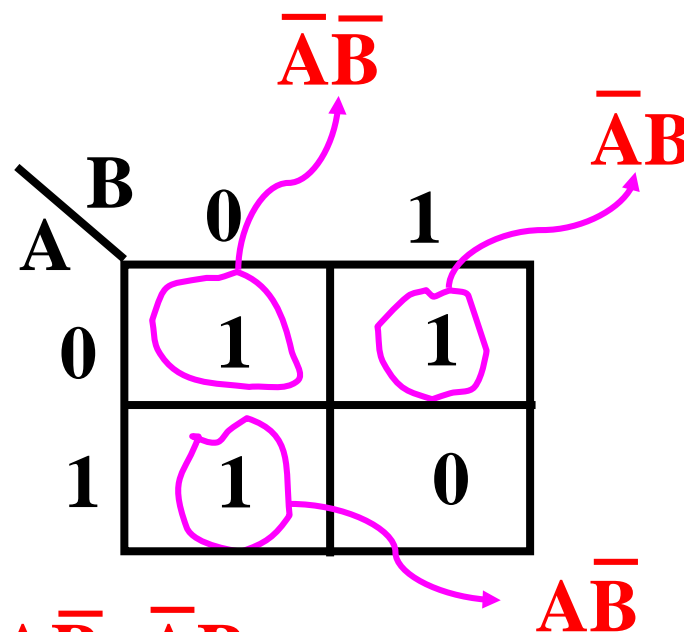
画出对应的逻辑图。

3、真值表、卡诺图 \leftrightarrow 逻辑代数式

方法: 将真值表或卡诺图中为1的项相加, 写成“与或式”。

真值表

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



$$Y = \bar{A}\bar{B} + A\bar{B} + \bar{A}B$$

此逻辑代数式并非是最简单的形式, 实际上此真值表是与非门的真值表, 其逻辑代数式为 $Y = \overline{AB}$ 因此, 有一个化简问题。

逻辑函数的简化

逻辑简化的意义和标准

逻辑简化的意义：

- (1) 逻辑简化对降低成本有直接的意义
- (2) 整个电路的功耗也相应减少
- (3) 提高了整个电路的工作速度

逻辑简化的目的：

消去多余的乘积项和每个乘积项中多余的因子，以得到逻辑函数式的最简形式。

逻辑简化的标准：

较常用的一种是要求简化的结果与一或式中与项最少且每项的变量最少，该标准对应与于电路就是所用的门最少，且每个门的输入端最少。

公式法简化

并项法 $AB + A\bar{B} = A$

试用并项法化简下列逻辑函数：

$$Y_1 = \overline{A\bar{B}CD} + \overline{A\bar{B}CD}$$

$$Y_2 = \overline{A\bar{B}} + \overline{ACD} + \overline{A\bar{B}} + \overline{ACD}$$

$$Y_3 = \overline{A\bar{B}C} + \overline{AC} + \overline{BC}$$

$$Y_4 = \overline{BCD} + \overline{BCD} + \overline{BCD} + \overline{BCD}$$

吸收法 $A + AB = A$

试用吸收法化简下列逻辑函数：

$$Y_1 = (\overline{\overline{AB}} + C)ABD + AD$$

$$Y_2 = AB + AB\overline{C} + ABD + AB(\overline{C} + \overline{D})$$

$$Y_3 = A + \overline{\overline{A} \cdot \overline{BC}}(\overline{A} + \overline{\overline{BC}} + D) + BC$$

消因子法 $A + \overline{A}B = A + B$

试用消因子法化简下列逻辑函数：

$$Y_1 = \overline{B} + ABC$$

$$Y_2 = A\overline{B} + B + \overline{A}B$$

$$Y_3 = AC + \overline{A}D + \overline{C}D$$

消项法

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

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

试用消项法化简下列逻辑函数：

$$Y_1 = AC + A\overline{B} + \overline{B} + C$$

$$Y_2 = A\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{E} + \overline{A}C\overline{D}E$$

$$Y_3 = \overline{A}\overline{B}C + ABC + \overline{A}B\overline{D} + A\overline{B}\overline{D} + \overline{A}BC\overline{D} + BC\overline{D}E$$

配项法

$$A + A = A \quad \text{重叠律}$$

$$A + \overline{A} = 1 \quad \text{互补律}$$

试用配项法化简下列逻辑函数：

$$Y_1 = \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC$$

$$Y_2 = A\overline{B} + \overline{A}B + B\overline{C} + \overline{B}C$$

Verilog HDL基本要素

module *module_name* (*port_list*);

port declarations

data type declarations

circuit functionality

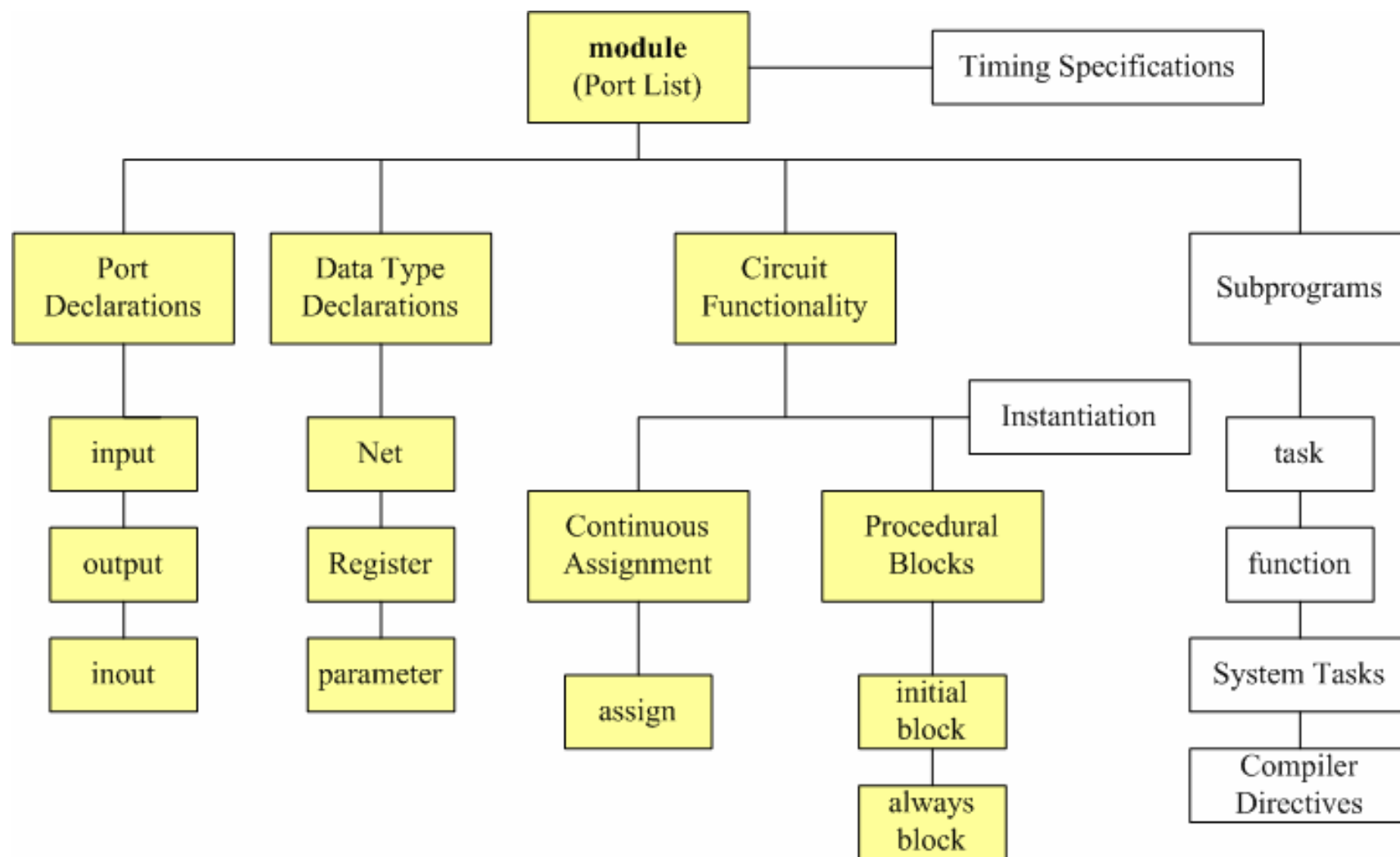
timing specifications

endmodule

注意点:

- 大小写敏感
- 所有关键词须小写
- 空格用于增加可读性
- 分号是语句终结符
- 单行注释: //
- 多行注释: /* */
- 时间规范用于仿真

Verilog HDL基本要素图



模块（module）和 端口（port）

➤ 模块声明

```
module <模块名>（模块端口列表）;  
    <模块内容>  
endmodule
```

➤ 端口列表

- 列出所有端口名称

➤ 端口声明

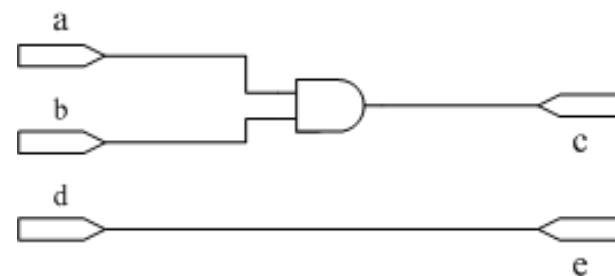
```
<端口类型> <端口名>;
```

➤ 端口类型

- input → 输入端口
- output → 输出端口
- inout → 双向端口

例：

```
module hello_world(a,b,c,d,e);  
input a, b, d;  
output c, e;  
    assign c = a & b;  
    assign e = d;  
endmodule
```



数据类型(Data Types)

➤常量

- 参数 (parameter)

➤变量

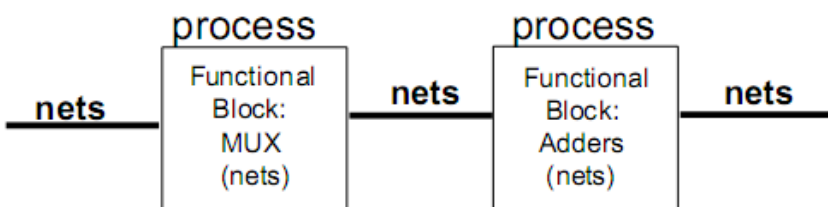
- 线网型 (nets type)
 - wire型最常用
- 寄存器型 (register type)
 - 标量
 - 向量
 - 数组

变量

➤线网型

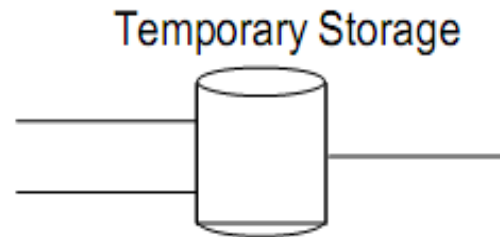
- 用关键词**wire**等声明
- 相当于硬件电路里的物理连接，特点是输出值紧跟输入值变化

例：**wire**[7:0] in, out;
assign out=in;



➤寄存器型

- 用**reg**或**integer**申明
- 具有保持作用的元件
- 注：不表示必将综合成物理（硬件）寄存器！**
- 在过程语句(**always**, **initial**)中赋值
- **integer**是含符号整数型变量



*存储器

➤二维寄存器数组

➤不能是线网型（Net）

— 例如：

```
reg[31:0] mem [0:1023]; // 1k × 32
```

```
reg[31:0] instr;
```

```
instr = mem[2];
```

➤注意：

— 不允许对存储器进行双索引操作

```
instr = mem[2][7:0] // 非法！
```

常量

➤ **parameter** 用来定义常量

例: **parameter** size=8;
reg [size-1:0] a, b;

*常量的合理正确使用在高级编程和大型程序设计中很重要!

数字 (Numbers)

➤ sized, unsized: <位宽>'<进制><数字>

- sized: 3'b010 // 3位二进制数字, 值为010
- unsized:
 - <进制>默认为十进制;
 - <位宽>默认为32-bit.

➤ 进制

- 十进制 ('d 或 'D)
- 十六进制 ('h 或 'H)
- 二进制 ('b 或 'B)
- 八进制 ('o 或 'O)

➤ 负数——在<位宽>前加负号

- 例: -8'd3

➤ 特殊数符

- '_' (下划线): 增加可读性
- 'x'或'X' (未知数)
- 'z' 或'Z' (高阻)

➤ 若定义的位宽比实际位数长

- 如果高位是0, x, z, 高位分别补0, x, z;
- 如果高位是1, 左边补0.

运算符 (Operators)

➤1. 算术运算符 (Arithmetic)

+	加
-	减
*	乘
	除
%	取模

➤2. 逻辑运算符 (Logical)

&&	逻辑与
	逻辑或
!	逻辑非

运算符 (Operators)

➤3. 位运算符 (Bitwise)

~	按位取反
&	按位与
	按位或
^	按位异或
^~, ~^	按位同或

➤4. 关系运算 (Relational)

<	小于
<=	小于或等于
>	大于
>=	大于或等于

➤注: <=也用于表示一种赋值操作

运算符 (Operators)

➤5.等价运算(Equality)

==	等于
!=	不等于
===	全等
!==	不全等

➤6.缩位运算(Reduction)

&	与
~&	与非
	或
~	或非
^	异或
^~, ~^	同或

运算符 (Operators)

➤7.移位(Shift)

>> 左移

<< 右移

➤8. 条件(Conditional)

?:

➤9. 位拼接 (Concatenation)

{ }

运算符优先级

➤ 缺省操作符优先级

+, -, !, ~ (单目操作符)

*, /, %

+, - (双目操作符)

<<, >>

<, <=, >, >=

==, !=, ===, !==

&, ~&

^, ^~

|

&&

||

?:

➤ () 可用于调整优先级

高优先级

低优先级



习题与训练

➤ 对之前举重比赛裁判评分逻辑进行Verilog HDL编程实现

例：举重比赛有三名裁判，当运动员将杠铃举起后，须有两名或两名以上裁判认可，方可判定试举成功，若用A、B、C分别代表三名裁判的意见输入，同意为1，否定为0；F为裁判结果输出，试举成功时F=1，试举失败时F=0。

➤ 其中，输入为 A、B、C，输出结果为F。

