ENGG 2020 数字电路

> Digital Logic and Systems

①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟㊱㊲㊳㊴㊵㊶㊷㊸㊹㊺㊻㊼㊽㊾㊿

bitwise　按位

protocol 协议

Lec 0 课程简介

Instructor: Dr. SUM Anthony

kwsum@cse.cuhk.edu.hk

Intended Learning Outcomes

- Understanding operations of logic gates, and flip-flops

> 理解逻辑门和触发器的运作

- Design and simplify digital logic circuits
    - combinational logic circuit（组合逻辑电路）
    - sequential logic circuits（时序逻辑电路）
- Build memory and storage systems

Topics

- Digital Systems and Binary Numbers (3.5 hrs)

> 数字系统与二进制数

- Boolean Algebra and Logic Gates (4 hrs)

> 布尔代数与逻辑门

- Gate Level Minimization (2 hrs)

> 门级简化

- Combinational Logic (5.5 hrs)

> 组合逻辑

- Synchronous Sequential Logic (5.5 hrs)

> 同步时序逻辑

- Registers and Counters (2 hrs)

> 寄存器与计数器

- Memory and Programmable Logic (2 hrs)

> 存储器与可编程逻辑

Tentative Schedule

Week	Date (Mon)	Lecture (2hrs)	Date (Tue)	Tutorial / Lab	Date (Wed)	Lecture (1hr)
1	Jan 6	Digital Systems and Binary Numbers	Jan 7	---	Jan 8	Digital Systems and Binary Numbers
2	Jan 13	Digital Systems and Binary Numbers	Jan 14	Lab1	Jan 15	Boolean Algebra and Logic Gates
3	Jan 20	---	Jan 21	T1	Jan 22	Boolean Algebra and Logic Gates
4	Jan 27	Boolean Algebra and Logic Gates	Jan 28	Lunar New Year	Jan 29	Lunar New Year
5	Feb 03	Lunar New Year	Feb 04	---	Feb 05	Boolean Algebra and Logic Gates
6	Feb 10	Gate Level Minimization	Feb 11	Lab2	Feb 12	Gate Level Minimization
7	Feb 17	---	Feb 18	T2	Feb 19	Combinational Logic
8	Feb 24	Combinational Logic	Feb 25	---	Feb 26	Combinational Logic
9	Mar 03	Reading Week	Mar 04	Reading Week	Mar 05	Reading Week
10	Mar 10	Combinational Logic	Mar 11	---	Mar 12	Synchronous Sequential Logic
11	Mar 17	Synchronous Sequential Logic	Mar 18	Lab3	Mar 19	Registers and Counters
12	Mar 24	---	Mar 25	T3	Mar 26	Registers and Counters
13	Mar 31	Registers and Counters	Apr 01	Homework	Apr 02	Memory and Programmable Logic
14	Apr 07	Memory and Programmable Logic	Apr 08	Lab4	Apr 09	Review
15	Apr 14	---	Apr 15	T4	Apr 16	---

Note: There is No Class with ---

Assessment

Assessment Items	Percentages
Lecture Attendance	5%
Laboratory × 4	40%
Homework × 1	15%
Final Examination	40%

> 每周二 Lab 是电脑模拟，不用实操.
>
> Tutorial 不计 Attendance.

Reference Book

Digital Design

- Authors: M. Morris Mano & Michael D. Ciletti
- Publisher: Pearson Education (US)
- Edition: Pearson International Edition, 4th Edition (or above, if any)

Lec 1 数字系统和二进制数

> Digital Systems and Binary Numbers

1.1 模拟信号与数字信号

> Analog vs. Digital

模拟（Analog）指通过连续变化的物理量（电压、电流等）表示信息. 模拟信号是连续的，其值可以在一个范围内取任意值. 如：

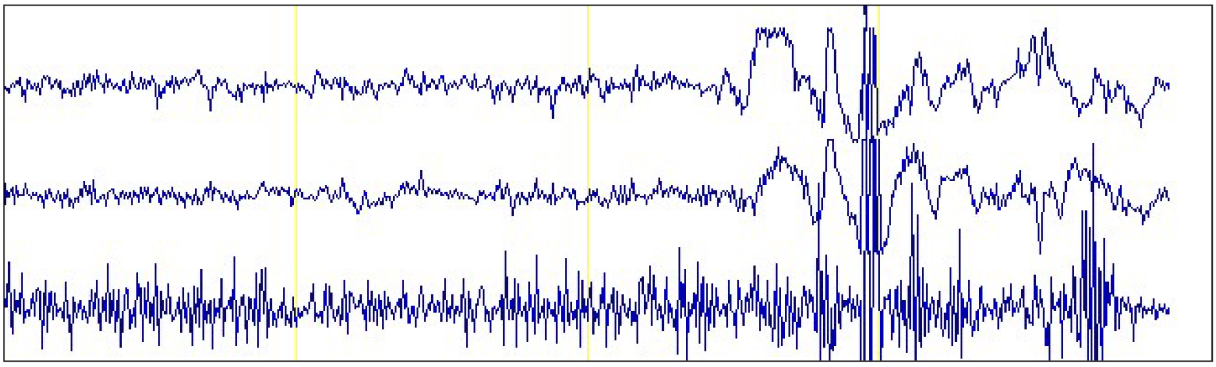
- 模拟音频：声音信号通过麦克风转换为连续的电压波形.
- 模拟视频：画面亮度和颜色信号使用连续电压或电流表示.

Nearly anything we can *feel*, *hear*, *touch*, and *sense* are analog signals

- We hear voice or *sound waves*
- We feel heat or *temperature*
- We see image or *light waves*

These signals are *analog* and are *continuous functions* of time

- They have value at any point in time
- For example, the electroencephalography (EEG，脑电图) signal produced by the brain



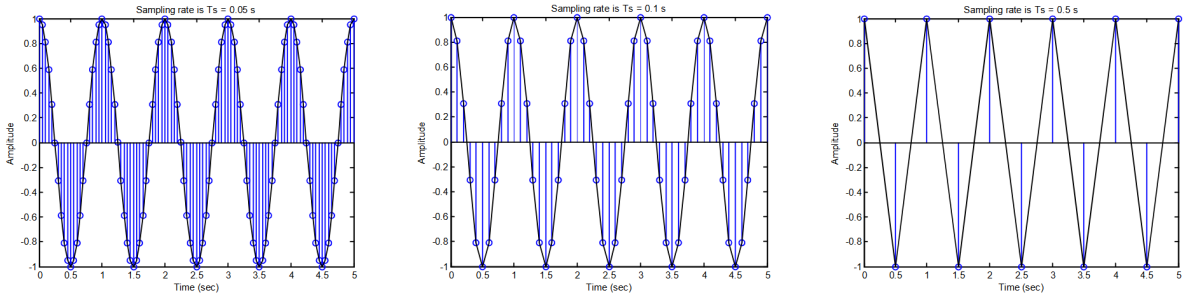
模拟的优点：

① 更自然

与之相对的**数字（Digital）**通过离散数值（通常是二进制的 0 和 1）表示信息. 数字信号是非连续的，只包含特定点的值.

Unlike analog signals, digital signals are actually *discrete samples*

- They are *not continuous*
- They are only available at a certain period of time, known as *sampling period*, Ts



数字的优点：

① 易于存储和传输，不易受噪声干扰.

② 可以准确复制，而模拟信号每次复制会损失质量.

③ 容易实现复杂处理，如压缩、加密等.

- Digital circuits are relatively *easy to design*
- Digital circuits have *higher accuracy*, and *programmability*
- Digital signals can be *stored easily*
- Digital circuit is comparatively more immune to error and noise, *error detection* and *error correction* can be implemented
- Digital signal *transmission* will not be degraded over a long distance
- Digital signal is either high or low, hence there is *less* chance of *confusion*

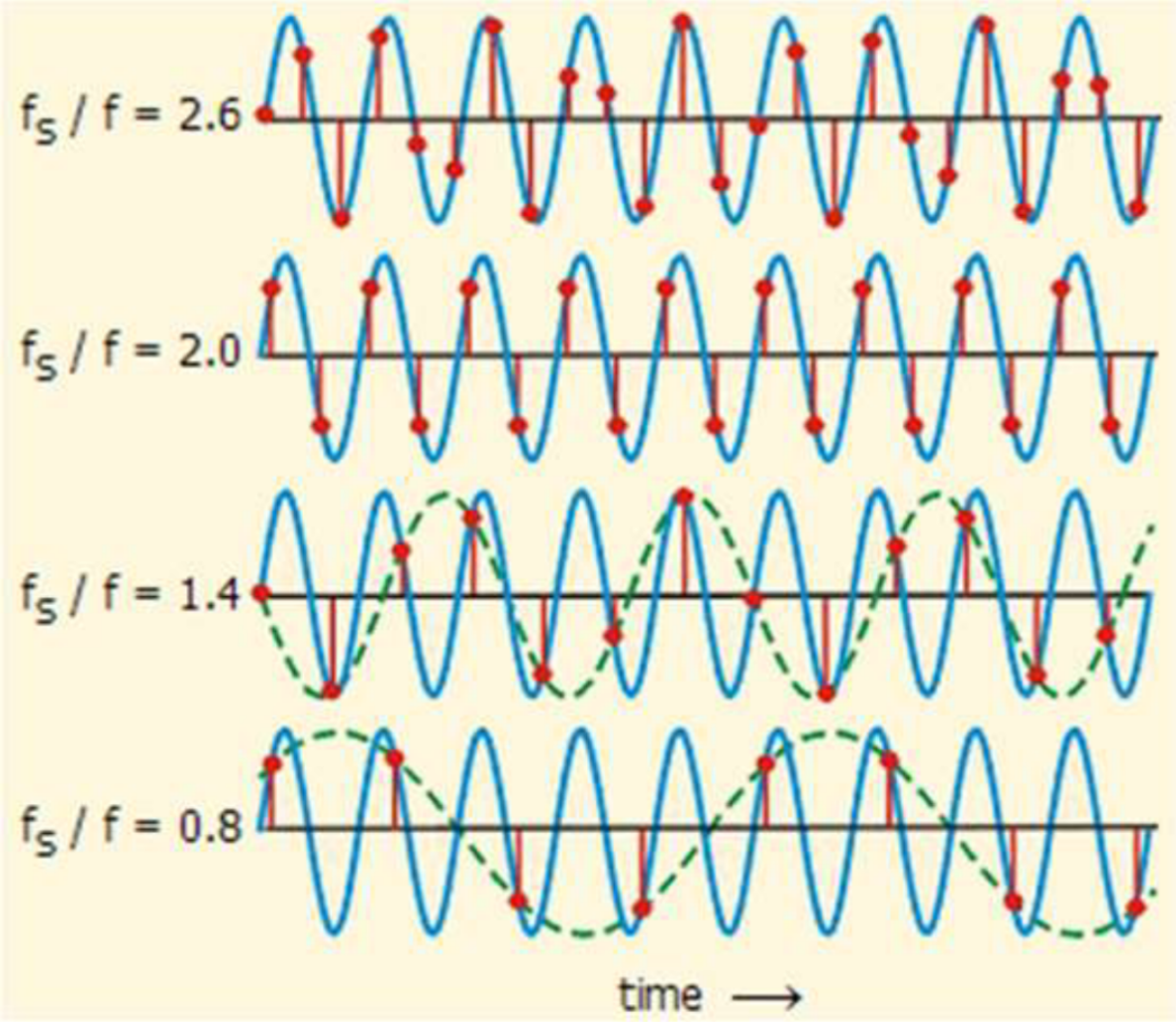
- Digital circuits have higher $flexibility$, we can change the functionality by software
- Digital circuits are more $reliable$, analog signal can be distorted by thermal noise

1.1.1 香农采样定理

> Shannon Sampling Theorem

如果一个连续信号的最高频率为 f_{max}，则只要采样频率 f_s 大于 $2f_{max}$，原始信号就可以从其采样值中完全重建.

If a function $x(t)$ contains no frequencies higher than f Hz, a sufficient sample-rate is therefore anything larger than $2f$ samples per second.



1.1.2 分辨率

> Resolution

采样数量：从原始信号中提取的离散点的数量.

> Number of Samples

原始信号可以通过插值在离散采样点之间重建. 采样点越多，信号还原得越好.

> The original signal can be reconstructed by $interpolation$ between consecutive discrete samples. The $more$ samples we have, the $better$ signal can be restored.

分辨率：离散值的精度，通常由位数（$number\ of\ bits$）决定.

- 4-bit：每个采样点可以用 $2^4 = 16$ 个不同离散值表示.
- 8-bit：每个采样点可以用 $2^8 = 256$ 个不同离散值表示.

分辨率越高，每个采样点的幅度值越接近真实信号幅度.

> The *higher* resolution, the *more accurate* value can be presented

采样数量和分辨率共同决定信号还原的质量. 采样频率不足会导致频率失真（如混叠），分辨率不足会导致幅度失真.

> 采样数量关注 x 轴，而分辨率关注 y 轴.

注意：分辨率的位数并不是指小数点后几位，而是信号的量化精度. 例如：

① 1-bit 分辨率

- 每个采样点只能取 2 个可能的值（0 和 1）.
- 非常粗糙，只能表示两种幅度.

② 8-bit 分辨率

- 每个采样点可以用 $2^8 = 256$ 个离散值表示.
- 可以更精确地接近原始信号的幅度.

③ 16-bit 分辨率

- 每个采样点可以用 $2^16 = 65536$ 个离散值表示.
- 更高的精度，常用于高质量音频或图像.

如果信号幅度范围是固定的（如 -1 到 $+1$），则更高的分辨率会使离散幅度值的间隔更小. 例如：

- 3-bit 分辨率：将幅度分成 $2^3 = 8$ 个离散值（如 $-1,\ -0.75,\ -0.5,\ \ldots,\ +1$）.
- 8-bit 分辨率：将幅度分成 $2^8 = 256$ 个离散值，幅度间隔更小.

1.1.3 信号保真度

> Fidelity

以医疗设备为例. 假设设备只能分辨 $\pm 0.2\ mV$ 的信号差异，但显示读数是 $0\ mV$.

真实信号可能在 $-0.2\ mV \sim +0.2\ mV$ 之间，而设备无法显示这些细微变化. 这样的误差在某些情况下没有影响，但在医疗场景中，可能有严重影响.

信号保真度（Fidelity）指的是信号处理系统保留真实信号细节的能力.

1.2 数制系统及转换

> Number Systems and their conversions
>
> 这节只讨论非负数.

1.2.1 常用进制

> Common Number Systems

Name	Base
Binary	2
Octal	8
Decimal	10
Hexadecimal	16

1.2.2 十进制

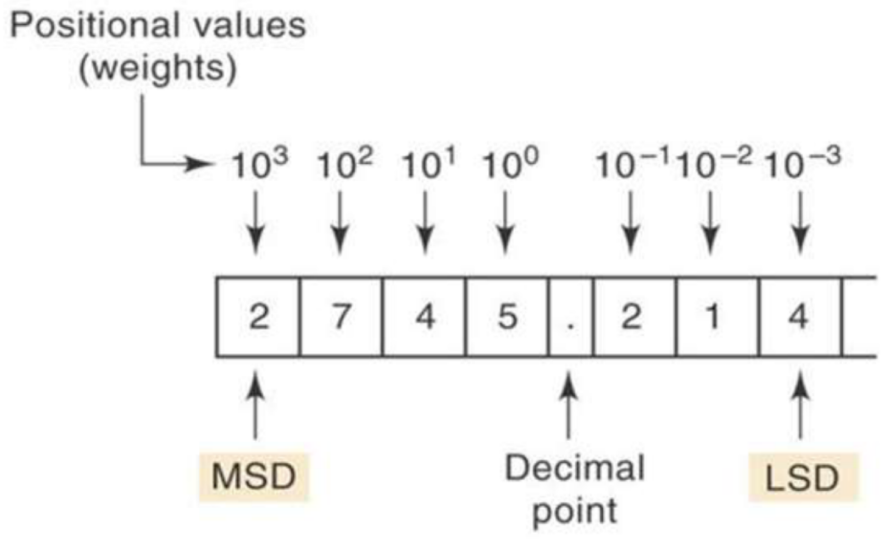
> Decimal

- Decimal digits: $0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9$
- Most Significant Digit (MSD)

> 一个数字的 Most Significant Digit 是该数字中位置权值最大的数字位（通常是最左边的数字）
>
> MSD 决定了一个数字的大致数量级.
>
> 十进制系统中，MSD 是最左边的非零数字. 0.045 的 MSD 是 4.

- Least Significant Digit (LSD)

> 一个数字的 Least Significant Digit 是该数字中位置权值最小的数字位（通常是最右边的数字）.
>
> LSD 决定了数字的最小变化单位.
>
> 0.045 的 LSD 是 5.



> 注意，这个概念可能会考到.

计算机科学中，MSD 和 LSD 的概念常用于排序算法（如基数排序）.

二进制系统中，最右边的比特是 Least Significant Bit (LSB)，最左边的比特是 Most Significant Bit (MSB).

1.2.3 十进制数

> Decimal Numbers

- Decimal digits: $0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9$

What does a decimal number 1234_{10} means?

- $1234_{10} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$

What is 56.78_{10}?

- $56.78_{10} = 5 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$

1.2.4 二进制数

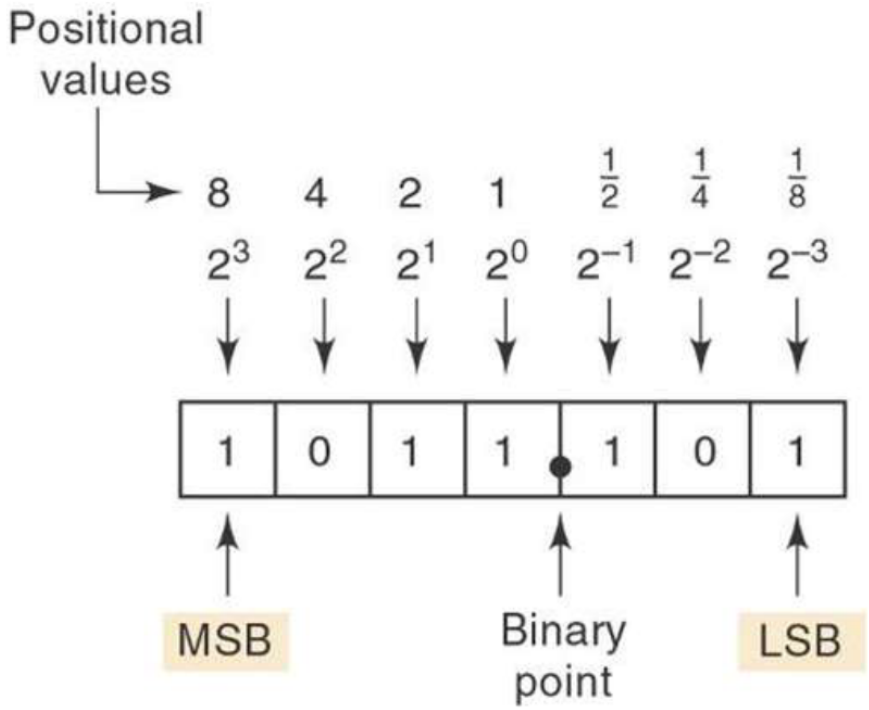
> 本节只讨论正数. 关于负数的二进制表示，见 `1.3.3 补码` .

- Binary digits: $0,\ 1$
- Most Significant Bit (MSB)
- Least Significant Bit (LSB)

1.2.5 二进制转十进制

> Binary to Decimal

- $1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- $1.01_2 = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$



1.2.6 十进制转二进制

> Decimal to Binary

By reversing the process

$$45_{10} = 32 + 8 + 4 + 1 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$76_{10} = 64 + 8 + 4 = 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

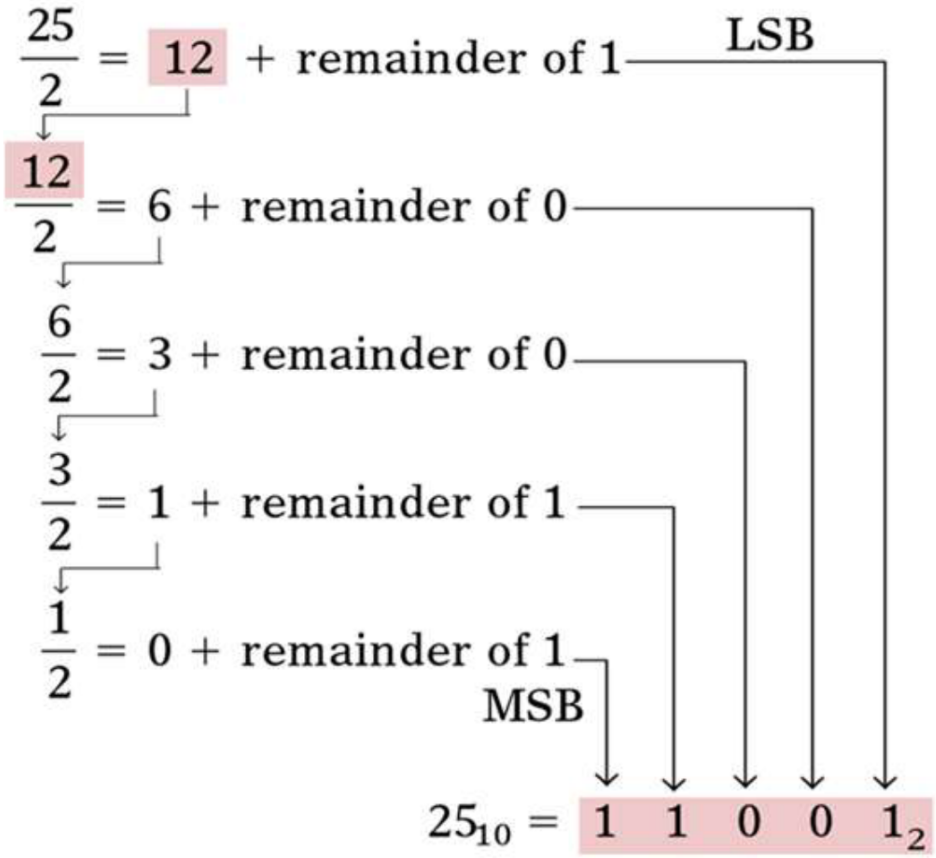
整数部分转换：除以 2 取余法

- 整数除以 2，记录余数.
- 商继续除以 2，重复此过程，直到商为 0.
- 最终的二进制数由所有余数按逆序排列.

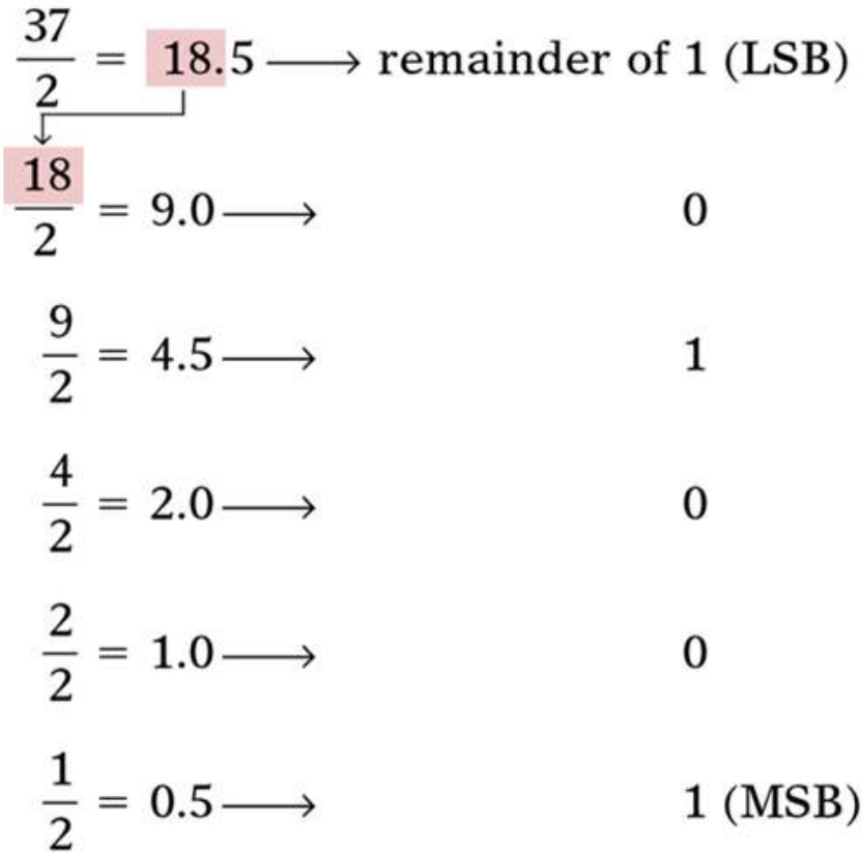
By repeated divisions

- Divide the decimal number by 2
- Write down the remainder after each division
- Until a quotient of zero is obtained
- The first remainder is the LSB
- The last remainder is the MSB

For example, $25_{10} = 11001_2$



For example, $37_{10} = 100101_2$



小数部分转换：乘以 2 取整法

- 小数部分乘以 2，记录整数部分（0 或 1）
- 取乘积的小数部分，继续乘以 2，重复此过程，直到小数部分为 0.
- 最终的二进制数小数部分由每次的记录按顺序排列.

> 注意，取余数法是逆序排列，取整法是顺序排列，二者不同.

For example, $0.25_{10} = 0.01_2$

- $0.25 \times 2 = 0.5 \to 0$
- $0.5 \times 2 = 1.0 \to 1$

For example, $0.3_{10} = 0.0\overline{0110011001}\ldots$

> 无限循环

- $0.3 \times 2 = 0.6 \to 0$
- $0.6 \times 2 = 1.2 \to 1$
- $0.2 \times 2 = 0.4 \to 0$
- $0.4 \times 2 = 0.8 \to 0$
- $0.8 \times 2 = 1.6 \to 1$
- $0.6 \times 2 = 1.2 \to 1$
- $\ldots$

1.2.7 十六进制

> Hexadecimal

- Hexadecimal digits: $0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A,\ B,\ C,\ D,\ E,\ F$
- Useful in handling *long binary strings*

> C 语言的地址就采用十六进制.

- By dividing them into 4-bit groups

- Useful in handling long binary strings
- By dividing them into 4-bit groups

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

4 bits

Additional to Decimal

1.2.8 十六转十进制

> Hexadecimal to Decimal

- $356_{16} = 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0 = 854_{10}$
- $2AF_{16} = 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 687_{10}$

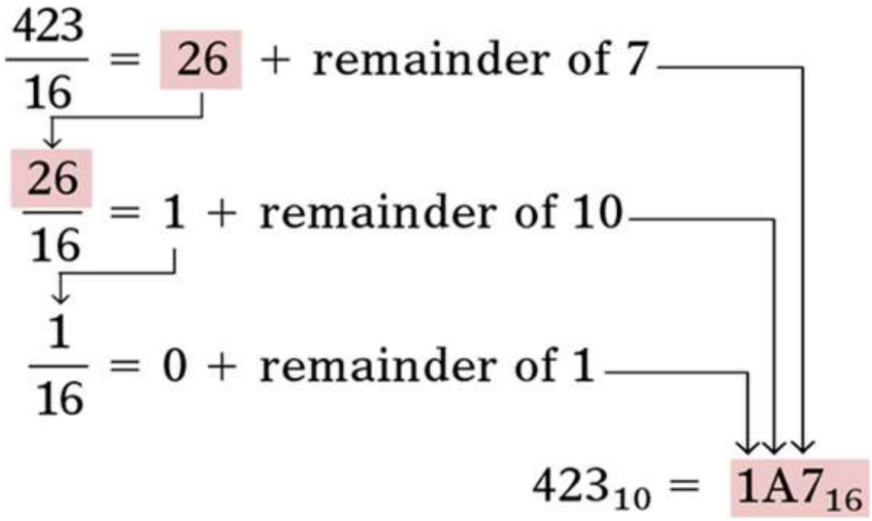
1.2.9 十转十六进制

> Decimal to Hexadecimal

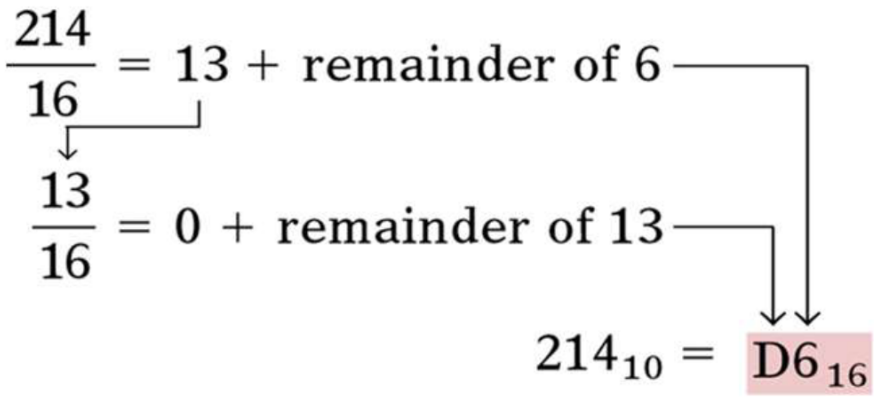
1.2.9.1 整数转换

> 类似十转二，同样可用取余数法十转十六.

For example, convert 423_{10} to Hexadecimal



For example, convert 214_{10} to Hexadecimal



1.2.9.2 浮点转换*

> 十进制浮点数转十六进制，补充内容，课内不考.

十进制浮点数转为十六进制有两种形式：

① 类似十转二，整数和小数部分分别转换，然后拼接.

整数部分：取余数法

小数部分：乘 16 取整法

> 注意这里取整要转为对应进制，即 16 进制.

Example: $10.625_{10} = A.A_{16}$

> 整数部分：$10_{10} = A_{16}$
>
> 小数部分：$0.625 \times 16 = 10.0$，取整 A_{16}，小数部分为 0，转换结束.

② 转换为 IEEE 754 标准的二进制浮点数，然后表示为十六进制.

步骤：确定符号位（正数为 0，负数为 1）、转为二进制科学计数法、根据 IEEE 754 标准分配指数和尾数（单精度/双精度）、拼接、二转十六进制（见 `1.2.B 二转十六进制`）.

> 具体 Example 见 `1.2.9.3 IEEE 754*`.

1.2.9.3 IEEE 754*

IEEE 754 是电气电子工程师协会（IEEE）制定的一个标准，用于表示浮点数. 是目前计算机中最常用的浮点数表示标准.

回顾：浮点数

浮点数（Floating Point Number）是一种用科学计数法表示的数字，常用于表示非常大或非常小的数值.

> 注意，浮点数涵盖范围远大于十进制小数，因此有自己的名称，而不是简单地称为小数.

它的形式为：

$$\pm M \times 2^E$$

> $\pm$：数字的符号（正或负，二选一）.
>
> M：尾数（Mantissa），也称有效数字，通常是一个小数.
>
> 2：基数（IEEE 754 标准中基数固定为 2）.
>
> E：指数（Exponent），表示尾数需要移动的小数点位置.

IEEE 754 使用 二进制 科学计数法来表示浮点数，并将其分为三部分：符号位、指数位 和 尾数位.

① 单精度（32 位）

单精度浮点数占用 4 字节（32 位）

符号位（S）	指数位（E）	尾数位（M）
1 位	8 位	23 位

> 符号位（S）：表示数字的正负，0 表示正数，1 表示负数.
>
> 指数位（E）：表示指数，用 偏移量（Bias）编码. 单精度偏移量为 127，即存储指数 $=$ 实际指数 $+ 127$（避免存储负指数，实际指数可正可负）.
>
> 尾数位（M）：存储尾数的小数部分（默认尾数的整数部分为 1，不用存储）.

② 双精度（64 位）

双精度浮点数占用 8 字节（64 位）

符号位（S）	指数位（E）	尾数位（M）
1 位	11 位	52 位

> 指数位偏移量为 1023.

数值表示公式

$$\text{值} = (-1)^S \times (1.M) \times (2^{E-Bias})$$

Example: 单精度浮点数表示

将十进制数 10.625_{10} 转换为 IEEE 754 单精度浮点数

① 符号位（S）

- 10.625 是正数，符号位 $S = 0$.

② 转为二进制科学计数法

整数部分：$10_{10} = 1010_2$

> 除以 2 取余数法.

小数部分：$0.625_{10} = 0.101_2$

> 乘 2 取整法.

$10.625_{10} = 1010.101_2$

转为科学计数法

> 后续步骤省略进制 2，理解即可.

$1010.101 = 1.010101 \times 2^3$

③ 指数位（E）

偏移量：127

> 题目要求单精度.

实际指数：3

存储指数 $=$ 实际指数 $+ 127 = 130$

$130_{10} = 10000010_2$

> 这里刚好满 8 位，若不满 8 位要在左侧用 0 补齐.

④ 尾数位（M）

第二步中，二进制科学计数法小数部分为 010101.

不足 23 位，右侧用 0 补齐：01010100000000000000000

⑤ 拼接

符号位：0

指数位：10000010

尾数位：01010100000000000000000

拼接结果：$0\ 10000010\ 01010100000000000000000$

> 拼接结果即计算机系统中浮点数的存储形式，但过于冗长，难以阅读，调试工具和技术文档多转为十六进制呈现，便于程序员快速核对数据是否正确.

⑥ 转为十六进制

> 见 `1.2.B 二转十六进制`.

二进制分段：$0100\ 0001\ 0010\ 1010\ 0000\ 0000\ 0000\ 0000$

转为十六进制：$0100\ 0001\ 0010\ 1010\ 0000\ 0000\ 0000\ 0000_2 = 412A0000_{16}$

特殊情况

IEEE 754 标准中还定义了几个特殊值：

① 零（0）

- 正零：符号位为 0，指数位和尾数位全为 0.
- 负零：符号位为 1，指数位和尾数位全为 0.

② 无穷大（Infinity）

- 正无穷：符号位为 0，指数位全为 1，尾数位全为 0.
- 负无穷：符号位为 1，指数位全为 1，尾数位全为 0.

③ 非数字（NaN）

- 指数位全为 1，尾数位非零.

④ 非规格化数（Denormalized Numbers）

当指数位全为 0 时，表示一个非常小的数，其值为：

$$(-1)^S \times (0.M) \times (2^{1-Bias})$$

> 指数部分是 $1-Bias$ 而不是 $0-Bias$，因为「确保非规格化数与规格化数两种表示之间的平滑过渡」，要比「拓展一点点指数范围」更重要.
>
> 最小的规格化数：1.0×2^{-126}，而非规格化数尾数部分无隐含 1，因此把指数固定为 $1-Bias$ 能实现平滑过渡.

1.2.A 十六转二进制

> Hexadecimal to Binary

四位四位转换.

- Each Hexadecimal digit gives 4 Binary bits

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

- $9F2_{16} = 1001\ 1111\ 0010_2$

1.2.B 二转十六进制

> Binary to Hexadecimal

先补齐左边的 0，然后四位四位转换.

- Group the binary number into 4-bit groups from LSB
- Leading 0 can be added to the left of the MSB

For example, convert 1110100110_2 to Hexadecimal

$1110100110_2 = 0011\ 1010\ 0110 = 3A6_{16}$

1.2.C 八进制

> Octal

- Octal digits: $0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7$

Octal Symbol	Binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

1.2.D 八 十 十六互转

$173_8 = 123_{10} = 7B_{16}$

1.2.E 八转二进制

> Octal to Binary

- Each Octal digit gives 3 Binary bits
- $173_8 = 001\ 111\ 011_2$

1.2.F 任意进制转换

> Convert from any base to any base

第一步：将源进制转换为十进制.

$decimal = \sum digit \times base^{digit\ number}$

> $digit$：某位上的数字.
>
> $digit\ number$：位数，从 0 开始.

第二步：将十进制的数字转换为目标进制.

> 取余数法.

1.3 二进制运算

> Binary Addition and Subtraction

1.3.1 二进制加法

> Binary Addition

从右往左按位加法.

> Bitwise addition from right to left.

Example: $0110_2 + 1001_2 = 01111_2$ with Carry bit $= 0$

	0	1	1	0
+	1	0	0	1
Carry	0	0	0	0
Sum	1	1	1	1

Example: $0110_2 + 1101_2 = 10011_2$ with Carry bit $= 1$

	0	1	1	0
+	1	1	0	1
Carry	1	1	0	0
Sum	0	0	1	1

If the system can only support 4-bit numbers, the result of the addition will be truncated to 4 bits.

> This means that any overflow beyond the 4 bits will be lost in the result. However, the system can detect and handle this overflow by using the **Carry flag** in the **System Status Register**.

1.3.2 二进制减法

> Binary Subtraction

从右往左按位减法.

> Bitwise substraction from right to left.

Example: $1110_2 - 0001_2 = 1101_2$

> $14 - 1 = 13$

Example: $0110_2 - 1001_2 = 1101$??

> $6 - 9 = -3$
>
> 错误

我们需要引入负数的二进制表示.

1.3.3 有符号二进制系统

> Signed Binary System

① 符号位表示法

> Sign-Magnitude Representation

这种方法中，最高（最左侧）的二进制位用作符号位：

- 0 表示正数.
- 1 表示负数.

其他位表示数值.

优点：直观.

缺点：有两个 0.

Decimal	Signed	1's	2's
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	---
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	---	---	1000

Unsigned Binary	Unsigned Decimal		Signed 1's Complement	Signed Decimal		Signed 2's Complement	Signed Decimal
000	0		111	-0		000	0
001	1		110	-1		111	-1
010	2		101	-2		110	-2
011	3		100	-3		101	-3
100	4		011	+3		100	-4
101	5		010	+2		011	+3
110	6		001	+1		010	+2
111	7		000	+0		001	+1

② 1's 补码

> 1's complement，一补数表示法.

对一个二进制数的每一位进行取反，所得到的结果就是该数的 1's complement.

> 通常用来表示负数.

对于正数，其 1's complement 与原码相同；对于负数，将该数原码中所有位进行取反.

> 原码指 Unsigned，即忽略负号时的二进制码.

Example: For a 4-bit 1's complement signed system, calculate $7_{10} + 2_{10}$ in binary numbers. What are the values of the sum, carry flag, and overflow flag after the calculation?

$7_{10} = 0111_2$

$2_{10} = 0010_2$

Sum: $7_{10} + 2_{10} = 0111_2 + 0010_2 = 1001_2 = -6_{10}$

Carry: 0

Overflow: 1

> It is overflow because adding two positive numbers yields a negative result (两个负数相加得到正数，也是 overflow)
>
> $7 + 2 = 9$ which is out of the numbering range of the system (i.e. -7 to $+7$)

缺点：存在「正零」与「负零」两种表示，可能使运算稍显复杂.

③ 2's 补码

> 2's complement，二补数表示法.

2's 补码是在 1's 补码的基础上再加 1. 它是目前计算机中最普遍使用的表示有符号整数的方法.

对正数，其 2's complement 与原码相同；对负数，先求出其 1's 补码，再在 1's 补码的基础上加 1.

Example: For a 4-bit 2's complement signed system, calculate $-5_{10} + 4_{10}$ in binary numbers. What are the values of the sum, carry flag, and overflow flag after the calculation?

$-5_{10} = 0101_c = 1011_2$

$4_{10} = 0100_2$

sum: $-5_{10} + 4_{10} = 1011_2 + 0100_2 = 1111_2 = 15_{10}$

> 错了，1111_2 在 signed system 里是负数
>
> $1111_2 = -1_{10}$

carry: 0

overflow: 0

注意，补码系统通常都是 signed 的，unsigned 的补码系统不存在.

1.3.4 溢出

> Overflow
>
> Carry 和 Overflow 是不同的概念.

Overflow indicates that the calculation result is $out\text{-}of\text{-}range$.

1.4 浮点数表示

> Floating Point Representation

在 `1.2.9.3 IEEE 754*` 亦有提及.

1.4.1 浮点转十进制

> Floating Point to Decimal

There are 3 elements in a 32-bit single precision floating point representation.

- Sign：符号位，表示数字的正负. 0 表示正数，1 表示负数.

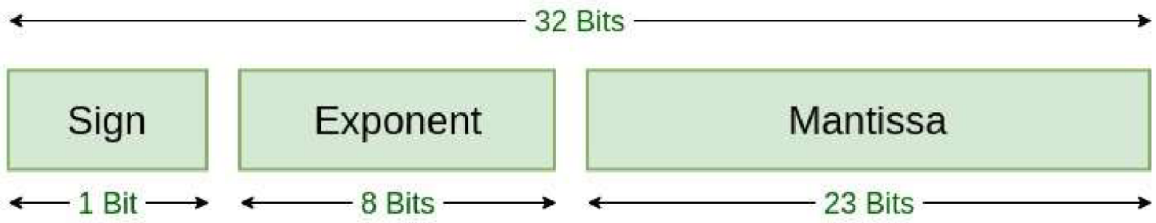
> 0 for positive number, 1 for negative number

- Exponent：指数位，表示指数，用 偏移量（Bias）编码. 单精度偏移量为 127，即存储指数 $=$ 实际指数 $+\ 127$（避免存储负指数，实际指数可正可负）.

> an exponent with a bias of 127
>
> For an exponent $= 3$, the biased exponent $= 127 + 3 = 130$
>
> For an exponent $= -2$, the biased exponent $127 - 2 = 125$

- Mantissa：尾数位，存储尾数的小数部分（默认尾数的整数部分为 1，不用存储）.

> the rest of digital after the decimal point of the normalized binary number
>
> For normalized number $= 1.101$, the mantissa is $1010\ 0000\ 0000\ 0000\ 0000\ 000$



Single Precision
IEEE 754 Floating-Point Standard

Example: Convert -0.078125_{10} to IEEE 754 Floating Point Standard format.

Sign: 1

convert decimal to binary number:

$0.078125_{10} = 0.000101_2 = 1.01 \times 2^{-4}$

exponent: $-4 + 127 = 123_{10} = 01111011_2$

mantissa: 01000000000000000000000

result: $1\ 01111011\ 01000000000000000000000$

(optional) convert to Hexadecimal: $1011\ 1110\ 1101\ 0000\ 0000\ 0000\ 0000\ 0000_2 = BED00000$

1.5 编码系统

> BCD and Gray Codes

1.5.1 BCD

> Binary Coded Decimal，二进制编码的十进制
>
> 用二进制表示十进制数.

Binary Coded Decimal (BCD) is a coding system that assigns a four-digit binary code to each decimal digit (i.e. 0-9)

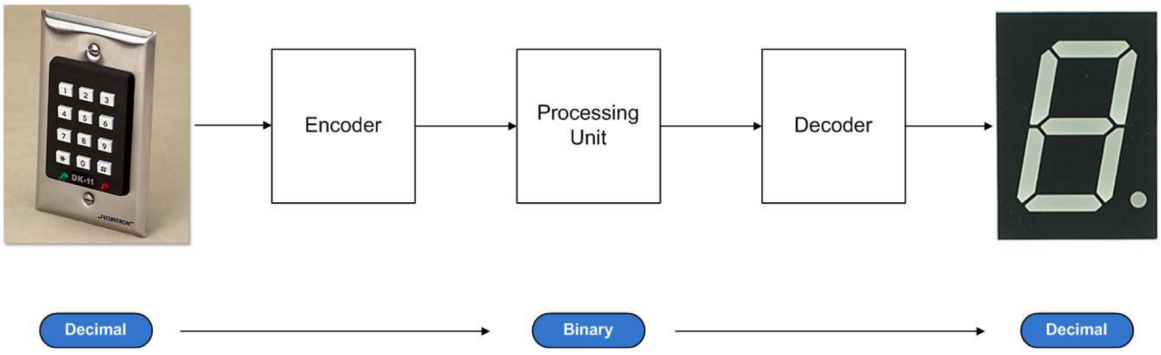
DECIMAL	BINARY	BINARY CODE DECIMAL 8 4 2 1
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

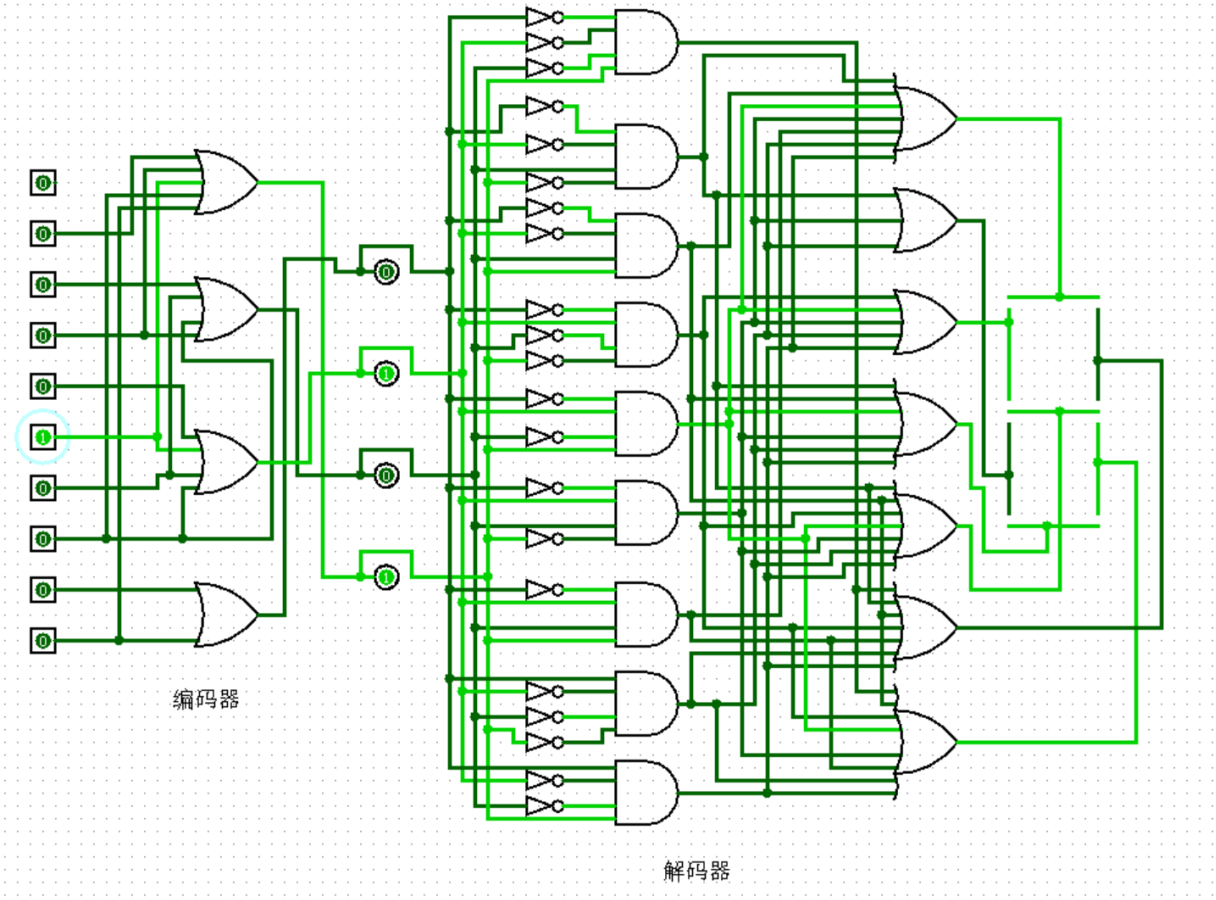
① 应用

> Application of BCD

BCD are commonly used in number display systems.

When you press a number on the keypad, the corresponding number is showed on a 7-segment LED display.



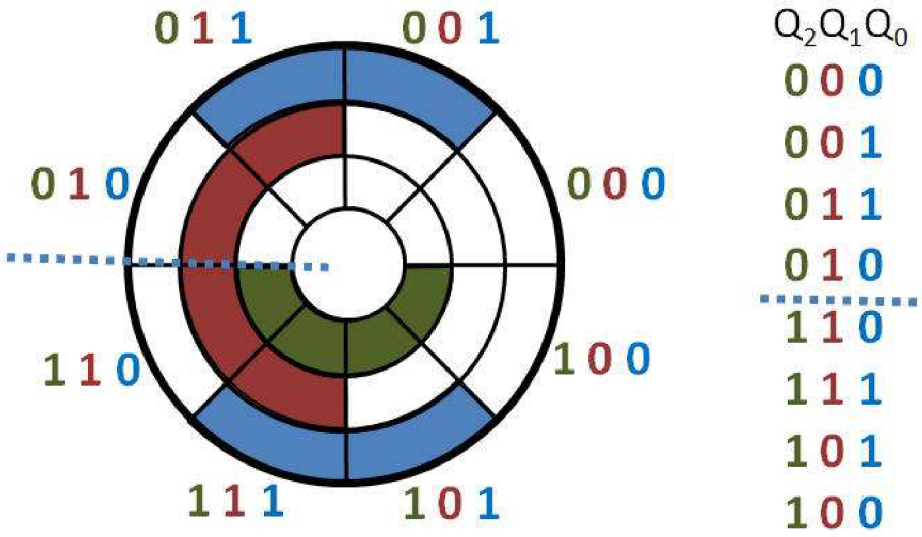


> 这里略去了中间的处理单元.
>
> 详见 `附录 1.七段显示屏` .

1.5.2 格雷码

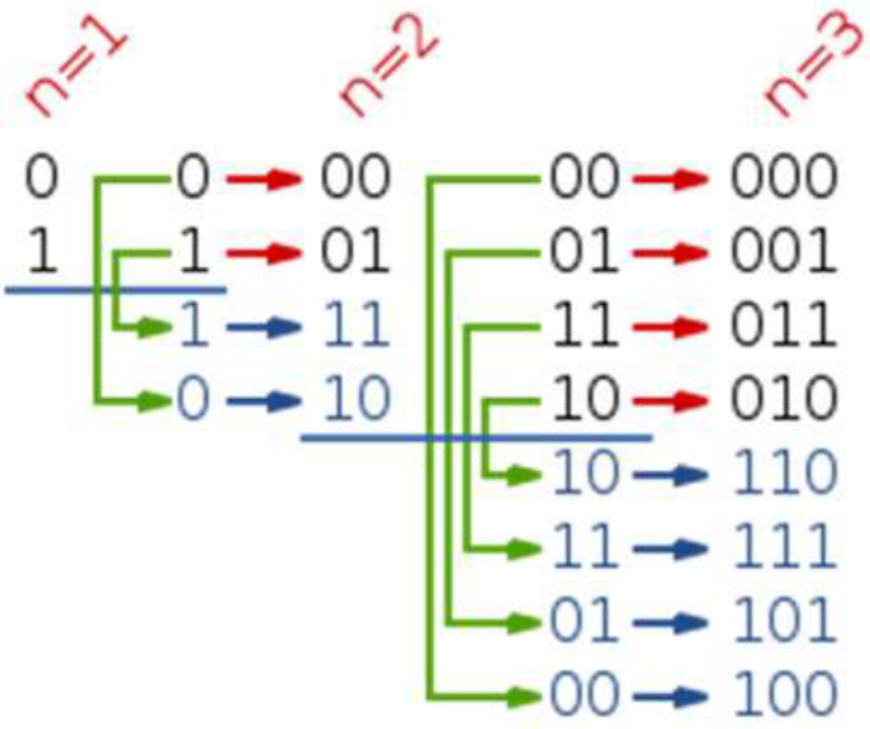
> Gray Code

旋转编码器



① 生成

格雷码生成方法：反射构造



- Gray code is also called reflected binary number

1.5.3 ASCII 码

> ASCII Code (American Standard Code for Information Interchange)

A 7-bit code is used to represent all the alphanumeric data used in computer

> 7 位二进制数表示，支持 128 种字符（0 到 127）.

Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code
0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	&	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	$	70	46	F	102	66	f
7	07	BEL	39	27	‘	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	140	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

① 标准 ASCII

> 7 位二进制编码，支持 128 个字符.

十进制	十六进制	二进制	字符/名称	描述
0	0x00	00000000	NUL (Null)	空字符
1	0x01	00000001	SOH (Start of Header)	标题开始
2	0x02	00000010	STX (Start of Text)	正文开始
3	0x03	00000011	ETX (End of Text)	正文结束
4	0x04	00000100	EOT (End of Transmission)	传输结束
5	0x05	00000101	ENQ (Enquiry)	请求
6	0x06	00000110	ACK (Acknowledge)	收到通知
7	0x07	00000111	BEL (Bell)	响铃
8	0x08	00001000	BS (Backspace)	退格

十进制	十六进制	二进制	字符/名称	描述
9	0x09	00001001	TAB (Horizontal Tab)	水平制表符
10	0x0A	00001010	LF (Line Feed)	换行
11	0x0B	00001011	VT (Vertical Tab)	垂直制表符
12	0x0C	00001100	FF (Form Feed)	换页
13	0x0D	00001101	CR (Carriage Return)	回车
14	0x0E	00001110	SO (Shift Out)	不用切换
15	0x0F	00001111	SI (Shift In)	启用切换
16	0x10	00010000	DLE (Data Link Escape)	数据链路转义
17	0x11	00010001	DC1 (Device Control 1)	设备控制1
18	0x12	00010010	DC2 (Device Control 2)	设备控制2
19	0x13	00010011	DC3 (Device Control 3)	设备控制3
20	0x14	00010100	DC4 (Device Control 4)	设备控制4
21	0x15	00010101	NAK (Negative Acknowledge)	拒绝接收
22	0x16	00010110	SYN (Synchronous Idle)	同步空闲
23	0x17	00010111	ETB (End of Trans. Block)	传输块结束
24	0x18	00011000	CAN (Cancel)	取消
25	0x19	00011001	EM (End of Medium)	媒介结束
26	0x1A	00011010	SUB (Substitute)	替补
27	0x1B	00011011	ESC (Escape)	转义
28	0x1C	00011100	FS (File Separator)	文件分隔符
29	0x1D	00011101	GS (Group Separator)	分组分隔符
30	0x1E	00011110	RS (Record Separator)	记录分隔符
31	0x1F	00011111	US (Unit Separator)	单元分隔符
32	0x20	00100000	(空格)	空格
33	0x21	00100001	!	感叹号
34	0x22	00100010	"	双引号
35	0x23	00100011	#	井号
36	0x24	00100100	$	美元符号
37	0x25	00100101	%	百分号
38	0x26	00100110	&	和号
39	0x27	00100111	'	单引号
40	0x28	00101000	(	左括号

十进制	十六进制	二进制	字符/名称	描述
41	0x29	00101001	)	右括号
42	0x2A	00101010	*	星号
43	0x2B	00101011	+	加号
44	0x2C	00101100	,	逗号
45	0x2D	00101101	-	减号
46	0x2E	00101110	.	句号
47	0x2F	00101111	/	斜杠
48	0x30	00110000	0	数字0
49	0x31	00110001	1	数字1
50	0x32	00110010	2	数字2
51	0x33	00110011	3	数字3
52	0x34	00110010	4	数字4
53	0x35	00110011	5	数字5
54	0x36	00110011	6	数字6
55	0x37	00110111	7	数字7
56	0x38	00111000	8	数字8
57	0x39	00111001	9	数字9
58	0x3A	00111010	:	冒号
59	0x3B	00111011	;	分号
60	0x3C	00111100	<	小于号
61	0x3D	00111101	=	等号
62	0x3E	00111110	>	大于号
63	0x3F	00111111	?	问号
64	0x40	01000000	@	电子邮件符号
65	0x41	01000001	A	大写字母A
66	0x42	01000010	B	大写字母B
67	0x43	01000011	C	大写字母C
68	0x44	01000100	D	大写字母D
69	0x45	01000101	E	大写字母E
70	0x46	01000110	F	大写字母F
71	0x47	01000111	G	大写字母G
72	0x48	01001000	H	大写字母H

十进制	十六进制	二进制	字符/名称	描述
73	0x49	01001001	I	大写字母I
74	0x4A	01001010	J	大写字母J
75	0x4B	01001011	K	大写字母K
76	0x4C	01001100	L	大写字母L
77	0x4D	01001101	M	大写字母M
78	0x4E	01001110	N	大写字母N
79	0x4F	01001111	O	大写字母O
80	0x50	01010000	P	大写字母P
81	0x51	01010001	Q	大写字母Q
82	0x52	01010010	R	大写字母R
83	0x53	01010011	S	大写字母S
84	0x54	01010100	T	大写字母T
85	0x55	01010101	U	大写字母U
86	0x56	01010110	V	大写字母V
87	0x57	01010111	W	大写字母W
88	0x58	01011000	X	大写字母X
89	0x59	01011001	Y	大写字母Y
90	0x5A	01011010	Z	大写字母Z
91	0x5B	01011011	[	左方括号
92	0x5C	01011100	\	反斜杠
93	0x5D	01011101	]	右方括号
94	0x5E	01011110	^	插入符号
95	0x5F	01011111	_	下划线
96	0x60	01100000	`	重音符
97	0x61	01100001	a	小写字母a
98	0x62	01100010	b	小写字母b
99	0x63	01100011	c	小写字母c
100	0x64	01100100	d	小写字母d
101	0x65	01100101	e	小写字母e
102	0x66	01100110	f	小写字母f
103	0x67	01100111	g	小写字母g
104	0x68	01101000	h	小写字母h

十进制	十六进制	二进制	字符/名称	描述
105	0x69	01101001	i	小写字母i
106	0x6A	01101010	j	小写字母j
107	0x6B	01101011	k	小写字母k
108	0x6C	01101100	l	小写字母l
109	0x6D	01101101	m	小写字母m
110	0x6E	01101110	n	小写字母n
111	0x6F	01101111	o	小写字母o
112	0x70	01110000	p	小写字母p
113	0x71	01110001	q	小写字母q
114	0x72	01110010	r	小写字母r
115	0x73	01110011	s	小写字母s
116	0x74	01110100	t	小写字母t
117	0x75	01110101	u	小写字母u
118	0x76	01110110	v	小写字母v
119	0x77	01110111	w	小写字母w
120	0x78	01111000	x	小写字母x
121	0x79	01111001	y	小写字母y
122	0x7A	01111010	z	小写字母z
123	0x7B	01111011	{	左大括号
124	0x7C	01111100		
125	0x7D	01111101	}	右大括号
126	0x7E	01111110	~	波浪号
127	0x7F	01111111	DEL	删除

> `0x` 是十六进制的前缀.

ASCII 码表可分为两部分：

控制字符

> 0 - 31 和 127
>
> 这些字符主要用于控制设备（如打印机、显示器）或通信.

十进制	十六进制	二进制	字符/名称	描述
0	0x00	00000000	NUL (Null)	空字符

十进制	十六进制	二进制	字符/名称	描述
1	0x01	00000001	SOH (Start of Header)	标题开始
2	0x02	00000010	STX (Start of Text)	正文开始
3	0x03	00000011	ETX (End of Text)	正文结束
4	0x04	00000100	EOT (End of Transmission)	传输结束
5	0x05	00000101	ENQ (Enquiry)	请求
6	0x06	00000110	ACK (Acknowledge)	收到通知
7	0x07	00000111	BEL (Bell)	响铃
8	0x08	00001000	BS (Backspace)	退格
9	0x09	00001001	TAB (Horizontal Tab)	水平制表符
10	0x0A	00001010	LF (Line Feed)	换行
11	0x0B	00001011	VT (Vertical Tab)	垂直制表符
12	0x0C	00001100	FF (Form Feed)	换页
13	0x0D	00001101	CR (Carriage Return)	回车
14	0x0E	00001110	SO (Shift Out)	不用切换
15	0x0F	00001111	SI (Shift In)	启用切换
16	0x10	00010000	DLE (Data Link Escape)	数据链路转义
17	0x11	00010001	DC1 (Device Control 1)	设备控制1
18	0x12	00010010	DC2 (Device Control 2)	设备控制2
19	0x13	00010011	DC3 (Device Control 3)	设备控制3
20	0x14	00010100	DC4 (Device Control 4)	设备控制4
21	0x15	00010101	NAK (Negative Acknowledge)	拒绝接收
22	0x16	00010110	SYN (Synchronous Idle)	同步空闲
23	0x17	00010111	ETB (End of Trans. Block)	传输块结束
24	0x18	00011000	CAN (Cancel)	取消
25	0x19	00011001	EM (End of Medium)	媒介结束
26	0x1A	00011010	SUB (Substitute)	替补
27	0x1B	00011011	ESC (Escape)	转义
28	0x1C	00011100	FS (File Separator)	文件分隔符
29	0x1D	00011101	GS (Group Separator)	分组分隔符
30	0x1E	00011110	RS (Record Separator)	记录分隔符
31	0x1F	00011111	US (Unit Separator)	单元分隔符
127	0x7F	01111111	DEL (Delete)	删除

可打印字符

> 32 - 126
>
> 这些字符是人类可见的字符，包括空格、数字、大小写字母、标点符号等.

十进制	十六进制	二进制	字符/名称	描述
32	0x20	00100000	(空格)	空格
33	0x21	00100001	!	感叹号
34	0x22	00100010	"	双引号
35	0x23	00100011	#	井号
36	0x24	00100100	$	美元符号
37	0x25	00100101	%	百分号
38	0x26	00100110	&	和号
39	0x27	00100111	'	单引号
40	0x28	00101000	(	左括号
41	0x29	00101001	)	右括号
42	0x2A	00101010	*	星号
43	0x2B	00101011	+	加号
44	0x2C	00101100	,	逗号
45	0x2D	00101101	-	减号
46	0x2E	00101110	.	句号
47	0x2F	00101111	/	斜杠
48	0x30	00110000	0	数字0
49	0x31	00110001	1	数字1
50	0x32	00110010	2	数字2
51	0x33	00110011	3	数字3
52	0x34	00110010	4	数字4
53	0x35	00110011	5	数字5
54	0x36	00110011	6	数字6
55	0x37	00110111	7	数字7
56	0x38	00111000	8	数字8
57	0x39	00111001	9	数字9
58	0x3A	00111010	:	冒号
59	0x3B	00111011	;	分号

十进制	十六进制	二进制	字符/名称	描述
60	0x3C	00111100	<	小于号
61	0x3D	00111101	=	等号
62	0x3E	00111110	>	大于号
63	0x3F	00111111	?	问号
64	0x40	01000000	@	电子邮件符号
65	0x41	01000001	A	大写字母A
66	0x42	01000010	B	大写字母B
67	0x43	01000011	C	大写字母C
68	0x44	01000100	D	大写字母D
69	0x45	01000101	E	大写字母E
70	0x46	01000110	F	大写字母F
71	0x47	01000111	G	大写字母G
72	0x48	01001000	H	大写字母H
73	0x49	01001001	I	大写字母I
74	0x4A	01001010	J	大写字母J
75	0x4B	01001011	K	大写字母K
76	0x4C	01001100	L	大写字母L
77	0x4D	01001101	M	大写字母M
78	0x4E	01001110	N	大写字母N
79	0x4F	01001111	O	大写字母O
80	0x50	01010000	P	大写字母P
81	0x51	01010001	Q	大写字母Q
82	0x52	01010010	R	大写字母R
83	0x53	01010011	S	大写字母S
84	0x54	01010100	T	大写字母T
85	0x55	01010101	U	大写字母U
86	0x56	01010110	V	大写字母V
87	0x57	01010111	W	大写字母W
88	0x58	01011000	X	大写字母X
89	0x59	01011001	Y	大写字母Y
90	0x5A	01011010	Z	大写字母Z
91	0x5B	01011011	[	左方括号

十进制	十六进制	二进制	字符/名称	描述
92	0x5C	01011100	\	反斜杠
93	0x5D	01011101	]	右方括号
94	0x5E	01011110	^	插入符号
95	0x5F	01011111	_	下划线
96	0x60	01100000	`	重音符
97	0x61	01100001	a	小写字母a
98	0x62	01100010	b	小写字母b
99	0x63	01100011	c	小写字母c
100	0x64	01100100	d	小写字母d
101	0x65	01100101	e	小写字母e
102	0x66	01100110	f	小写字母f
103	0x67	01100111	g	小写字母g
104	0x68	01101000	h	小写字母h
105	0x69	01101001	i	小写字母i
106	0x6A	01101010	j	小写字母j
107	0x6B	01101011	k	小写字母k
108	0x6C	01101100	l	小写字母l
109	0x6D	01101101	m	小写字母m
110	0x6E	01101110	n	小写字母n
111	0x6F	01101111	o	小写字母o
112	0x70	01110000	p	小写字母p
113	0x71	01110001	q	小写字母q
114	0x72	01110010	r	小写字母r
115	0x73	01110011	s	小写字母s
116	0x74	01110100	t	小写字母t
117	0x75	01110101	u	小写字母u
118	0x76	01110110	v	小写字母v
119	0x77	01110111	w	小写字母w
120	0x78	01111000	x	小写字母x
121	0x79	01111001	y	小写字母y
122	0x7A	01111010	z	小写字母z
123	0x7B	01111011	{	左大括号

十进制	十六进制	二进制	字符/名称	描述
124	0x7C	01111100		
125	0x7D	01111101	}	右大括号
126	0x7E	01111110	~	波浪号

② 扩展 ASCII*

> 如 ISO 8859-1，扩展到 256 个字符（8 位编码）.
>
> 不考.

1.5.4 Unicode

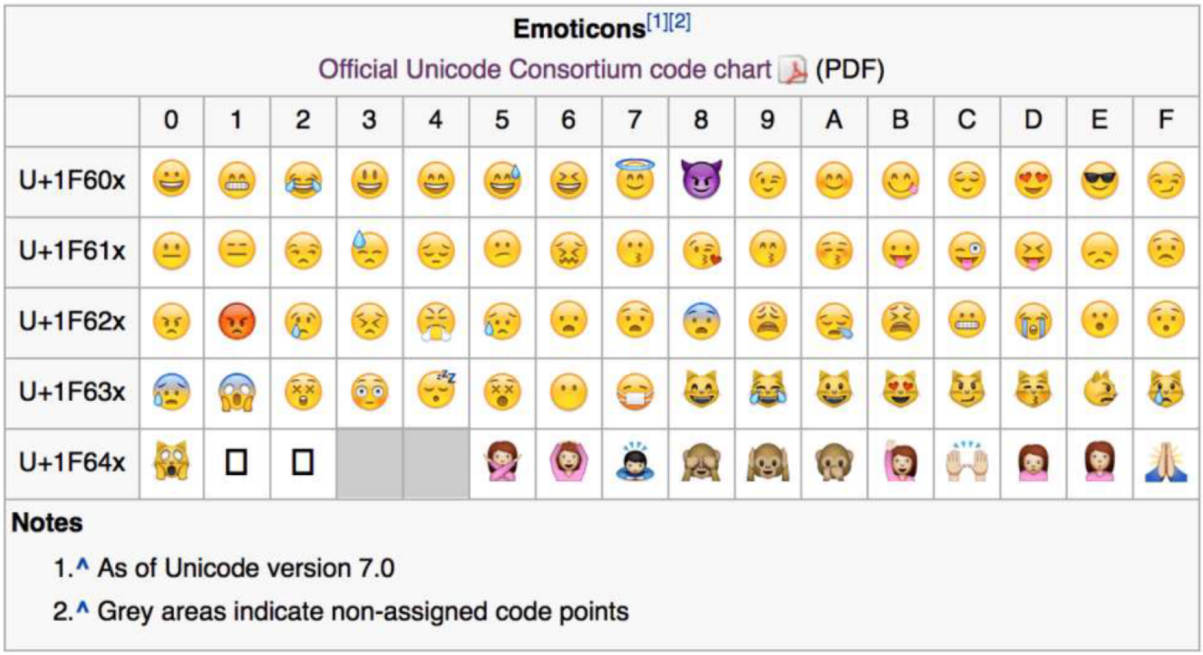
> 统一码，万国码

ASCII is not sufficient for global intercommunication

A character set including all languages is needed

At most 4 bytes are required for a character

UTF8 and UTF16 are different encodings of Unicode



Unicode（统一码、万国码）是一个国际标准，旨在为世界上所有语言的文字、符号、表情符号等提供唯一的数字编码. 它解决了传统字符编码（如 ASCII、GB2312、ISO 8859 等）中字符集有限、不兼容的问题，使不同语言和符号可以在计算机中被统一表示和处理.

① 特点

全球性：覆盖了几乎所有已知的书写系统，包括拉丁文、中文、阿拉伯文、日文、表情符号等.

唯一性：每个字符都有一个唯一的编码点（Code Point）.

兼容性：与许多传统编码（如 ASCII）兼容.

扩展性：Unicode设计是开放的，可以不断添加新的字符和符号.

② UTF-8

> 不考.

③ UTF-16

> 不考.

④ UTF-32

> 不考.

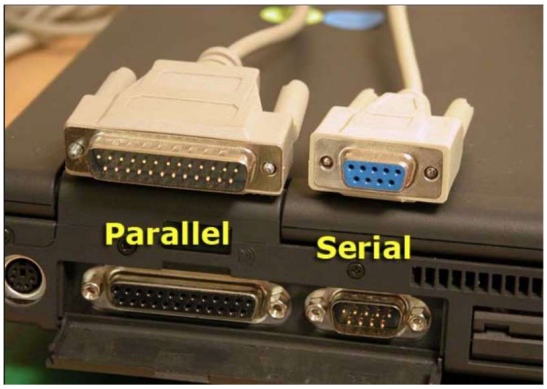
1.6 错误检测与纠正

> Error Detection and Correction

1.6.1 并行与串行

> Parallel vs Serial Transmission

Data transmission can be done in parallel or serial.



① 并行传播

> Parallel Propagation

并行传播是指同时传递多个信息或数据的传播方式. 它通常利用多个通道或资源以并行的方式进行信息的传递.

特点:

- 同时性 - 多个数据位或信息在同一时间内被传输.

适用场景:

- 短距离高效通信，例如计算机内部的总线（如并行接口）.
- 多核处理器之间的任务分配和通信.

优点:

- 速度快：可以同时传输多个数据位或数据流.
- 高吞吐量：适用于带宽需求高的场景.

缺点:

- 硬件复杂：需要更多的物理资源（如多个通道）.
- 易受干扰：并行信号容易受到电磁干扰，特别是传输距离较长时.
- 同步困难：需要解决不同信号之间的同步问题.

应用:

- 计算机的内存数据总线（早期的并行数据总线，如IDE接口）.
- 并行打印机接口（如旧的LPT接口）.
- 多线程或多进程并行执行任务.

② 串行传播

> Serial Propagation

串行传播是指将信息或数据按照顺序一个接一个地传递的传播方式. 它使用单一的通道或资源以线性方式传输数据.

特点:

- 逐位传输 - 数据按顺序一位一位或一个数据包一个数据包传输.

适用场景:

- 长距离传输（减少资源和干扰问题）.

- 通信设备和网络中的数据传输.

优点：

- 硬件简单：只需要单一的传输通道，降低了设计和实施的复杂性.
- 抗干扰能力强：由于只有一个信号通道，受电磁干扰的可能性较低.
- 适合长距离：串行传输在长距离通信中更稳定可靠.

缺点：

- 传输速度较慢：由于逐位传输，理论上速度可能低于并行传输（不过现代串行技术通常通过高频率弥补）.
- 需要额外的控制协议：例如启动和结束信号的标记.

应用：

- USB（通用串行总线）.
- 串口通信（如RS-232标准）.
- 网络通信中的数据帧传输（如以太网）.
- 串行硬盘接口（如SATA）.

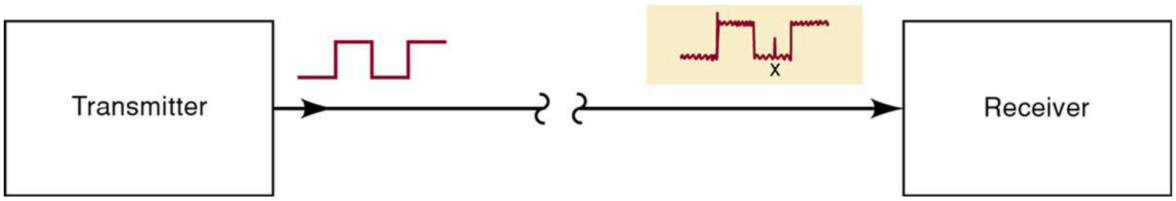
③ 并行 vs 串行

特性	并行传播	串行传播
传输路径	多通道（多条路径）	单通道（单条路径）
传输速度	理论上更快（同时传输多位）	较慢（逐位传输，但频率更高可弥补）
硬件复杂度	高（需要多个信号线）	低（只需要一条信号线）
抗干扰能力	较差（容易受干扰）	较强（单通道抗干扰能力强）
适用距离	短距离	长距离
同步问题	存在同步问题	无需同步问题
应用场景	内部高速处理（如计算机总线）	外部通信（如网络和串口传输）

1.6.2 错误检测

> Error Detection

- Electrical noise can cause errors during data transmission
- Fluctuations in voltage or current present in all electronic systems
- Error detection and correction is important



1.6.3 奇偶校验

> Parity Check

通过在原始数据后面添加一个校验位（Parity Bit）的方式，使数据的二进制位数满足某种奇偶性规则，从而实现错误检测.

- Parity Check is a simple error detection method which requires one extra bit
- The extra bit is called the parity bit
- There are two parity options, either even or odd
- For even/odd parity, the number of 1s in a data group must be an even/odd number respectively
- If the receiver find that the number of 1s is not matched the pre-set even/odd parity protocol（协议），an error is detected

> 注意，奇偶校验都是检查数据中 1 出现的次数.

- Assume we have an 8-bit original data 1011 1001
- For even parity, the transmitting data will become 1 1011 1001
- For odd parity, the transmitting data will become 0 1011 1001

分为奇校验和偶校验，取决于具体应用场景和系统约定. 如果你正在开发或对接一个系统，必须查看相关的文档或规范，按照对方的要求选择奇校验或偶校验.

> 接收端使用哪种校验方式是已知的.

校验过程

- 发送端：根据数据计算校验位，并将数据和校验位一起发送.
- 接收端：收到数据后，重新计算数据中 1 的个数（包括校验位在内），检查是否符合预期.

优点

- 实现简单，开销低.
- 能检测单个比特错误.

缺点

- 无法检测偶数个比特错误（例如两个比特被同时翻转）.
- 只能检测，不能纠正错误.

① 奇校验

发送端（保证数据中 1 的总个数为奇数）

- 如果数据中 1 的个数是奇数，则校验位为 0（保持奇数性）.
- 如果数据中 1 的个数是偶数，则校验位为 1（变为奇数）.

接收端（检查数据中 1 的总个数是否为奇数）

- 如果数据中 1 的个数是奇数，则没有单比特错误.
- 如果数据中 1 的个数是偶数，则有错误.

② 偶校验

发送端（保证数据中 1 的总个数为偶数）

- 如果数据中 1 的个数是偶数，则校验位为 0（保持偶数性）.
- 如果数据中 1 的个数是奇数，则校验位为 1（变为偶数）.

接收端（检查数据中 1 的总个数是否为偶数）

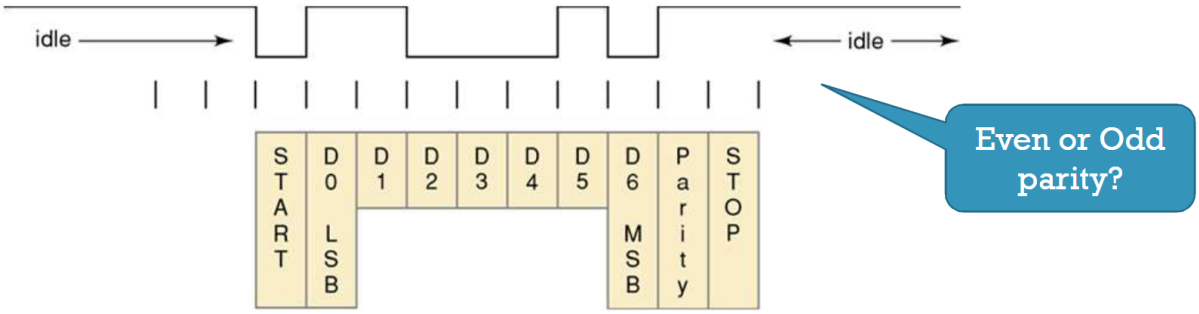
- 如果数据中 1 的个数是偶数，则没有单比特错误.
- 如果数据中 1 的个数是奇数，则有错误.

注意：奇偶校验无法检测出偶数个比特错误.

③ 应用：ASCII

> 已发邮件，待补充.
>
> 2025.1.20 已完成.

- An ASCII character must be framed, so that the receiver knows where the data begins and ends
- The first bit is a start bit which is logic 0
- Followed by 7-bit ASCII code D0 to D6
- Followed by a parity bit
- Ended with one/two stop bit(s) which is/are logic 1



- Start and stop bits are not included in the parity checking.

> 例如上图，D0-D6 有 3 个 1. 采用偶校验，校验位取 1，补齐 4 个 1.
>
> 不用计入 start 和 stop 的值.

1.6.4 汉明码

> Hamming Code
>
> 经典的纠错码，通过添加冗余位（校验位）检测和纠正单比特错误.
>
> 纠错相比检错，多了一步定位错误位置.

- An encoding scheme that can correct any single bit error
- Contains many parity bits that are placed in between the data bits strategically
- When the position number of the encoding bit stream is a power of 2, that bit will be used as check bits
- Each check bit has a corresponding check equation that covers a portion of the original data

- 8-bit original data: 0100 1011 (D8~D1)
- 8-bit original data with error: 0101 1011 (D8~D1)
- 4 parity bits: P4, P3, P2, P1 = 1, 0, 1, 0
- For even parity, we have the received data: 0101 1101 0110

12	11	10	9	8	7	6	5	4	3	2	1
D8	D7	D6	D5	P4	D4	D3	D2	P3	D1	P2	P1
0	1	0	1	1	1	0	1	0	1	1	0

① 特点

- 可以检测并纠正**单比特错误**；可以检测但不能纠正**双比特错误**.
- 校验位数量与数据位数量成对数关系.
- 主要应用于二进制数据流传输.

② 原理

Step 1：对于一个长度为 m 的二进制数据，需要添加 r 个校验位，使得

$$2^r \geq m + r + 1$$

> 待证明.

Step 2： 确定校验位位置

- 按 2 的幂次排列，1，2，4，8，$\ldots$
- 剩余位置用于数据位.

Step 3：奇偶校验

> 注意，使用奇校验还是偶校验也是提前规定好（已知）.

每个校验位检查对应位置：从第 n 位开始，检 n 位隔 n 位.

例如：P2 在第 2 位，则从第 2 位开始，检 2 位隔 2 位.

> $2, 3, 6, 7, \ldots$

Step 4：生成汉明码

将计算出的校验位插入编码中，生成最终汉明码.

> 可以用同样的奇偶校验规则检测，多个校验位可以实现纠错.
>
> 待证明.

Lec 2 布尔代数和逻辑门

> Boolean Algebra and Logic Gates

2.1 数字逻辑

> Digital Logic

Digital logic allows only two values, 0 and 1

> 1 在实际电路中不一定用高电位表示，取决于使用的标准.

- Logic 0 can be false, off, low, no, open switch
- Logic 1 can be true, on, high, yes, closed switch

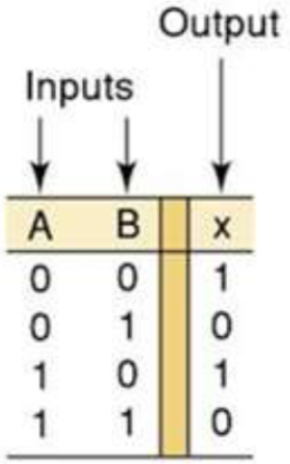
2.2 真值表

> Truth Table

2.2.1 定义

A truth table describes the relationship between the input and output of a logic circuit.

(a) A 2-input table has $2^2 = 4$ entries



A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

(b) A 3-input table has $2^3 = 8$ entries

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(c) A 4-input table has $2^4 = 16$ entries

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

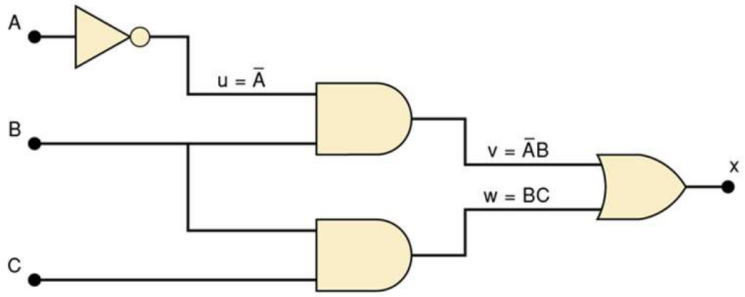
> 真值表从上到下要按升序排序，便于检索.

2.2.2 分析逻辑电路

One of the best ways to analyze a logic circuit to use a truth table.

> 列举了所有可能的 输入-输出 组合.

- Step 1: For a N-input logic circuit, create a 2^N-row truth table (excluding the header row), and list all combinations of the inputs **in order**
- Step 2: Create one column for each intermediate node



A	B	C	U= A'	V= A'B	W= BC	X
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

> 每个中间节点都开一列.

Step 3: Fill in the values of each column accordingly.

Step 4: Until the output values are determined.

A	B	C	U= A'	V= A'B	W= BC	X
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

2.3 逻辑门

> Logic Gates

2.3.1 与门

> AND Gate
>
> 编程里的 `&&`

当所有输入为 1 时，输出为 1；否则输出为 0.

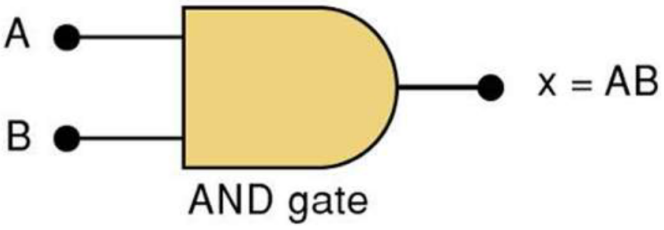
AND logical operation is similar to **multiplication**:

$$X = A \cdot B$$

AND

A	B	x = A · B
0	0	0
0	1	0
1	0	0
1	1	1

(a)

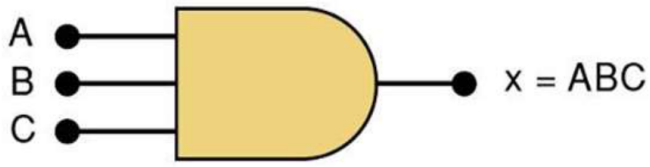


(b)

> (a) The truth table
>
> (b) the symbol of AND

与门支持多输入：

A	B	C	x = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



> Three inputs AND gate

2.3.2 或门

> OR Gate
>
> 编程里的 `||`

当任意输入为 1 时，输出为 1；否则输出为 0.

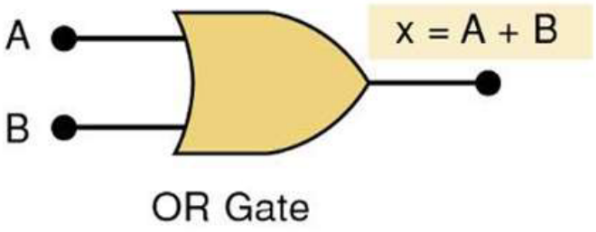
OR logical operation is similar to **summation**:

$$X = A + B$$

OR

A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

(a)

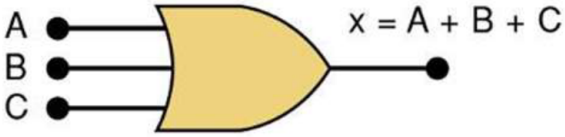


OR Gate

(b)

> (a) The truth table
>
> (b) the symbol of OR

与门支持多输入：



A	B	C	x = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

> Three inputs OR gate

2.3.3 非门

> NOT Gate
>
> 编程里的 `!`

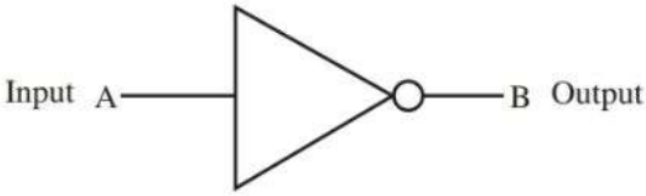
将输入信号取反. 输入为 0 时输出为 1，输入为 1 时输出为 0.

NOT logical operation is the **inverse/complement** of the input

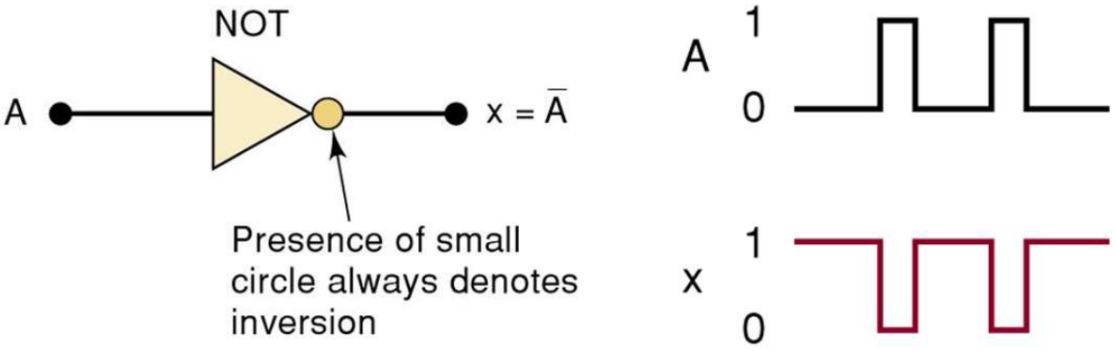
$$X = \overline{A} \quad or \quad X = A'$$

NOT

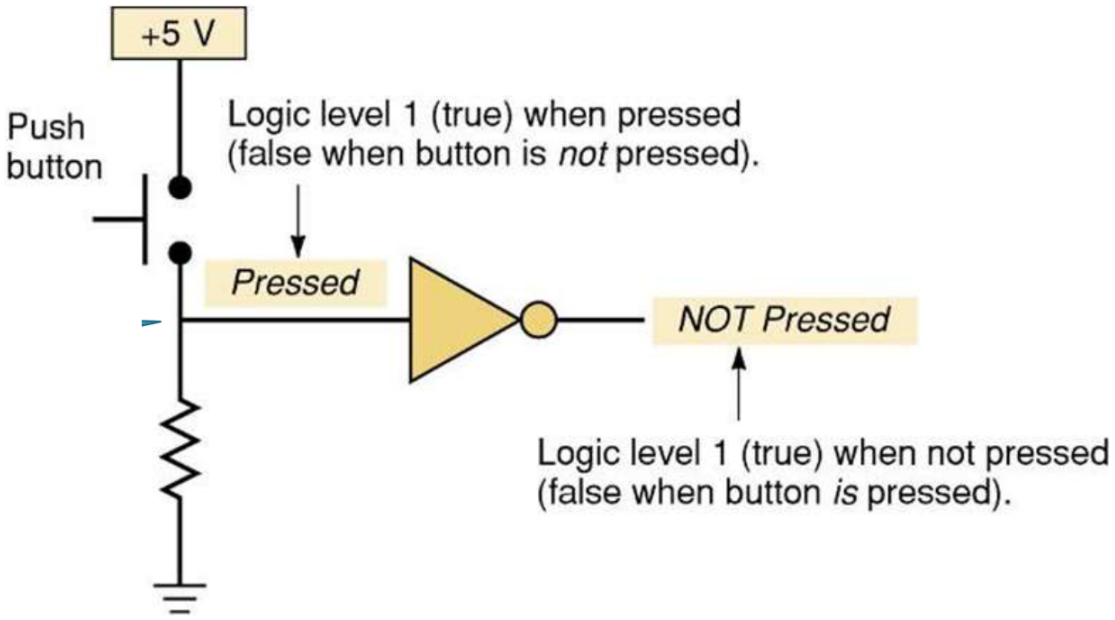
A	$x = \overline{A}$
0	1
1	0



The truth table and the symbol of NOT



A NOT gate is also called an inverter



Application of a NOT gate

注意：非门不支持多输入.

2.3.4 或非门

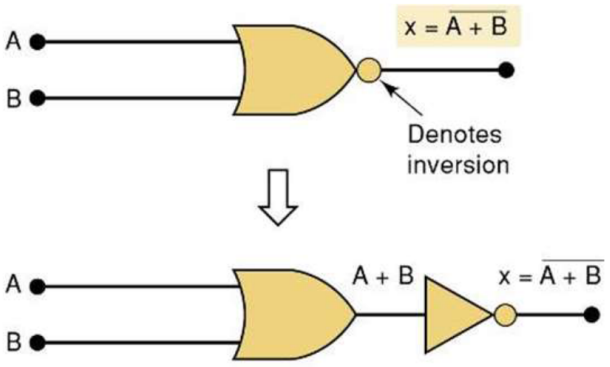
> NOR Gate
>
> 「或门」后接「非门」的组合.
>
> 支持多输入.

「或门」输出的取反，即：输入都为 0 时，输出为 1；否则输出为 0.

NOR gate is an inverted OR gate

An inversion "bubble" is placed at the output of the OR gate,

$$X = \overline{A + B} \quad or \quad X = (A + B)'$$



		OR	NOR
A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

2.3.5 与非门

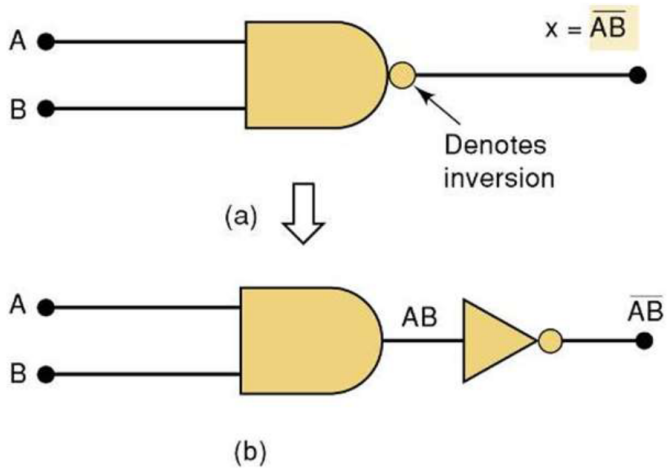
> NAND Gate
>
> 「与门」后接「非门」的组合.
>
> 支持多输入.

「与门」输出的取反，即：输入全为 1 时，输出 0；否则输出 1.

NAND gate is an inverted AND gate

$$X = \overline{AB} \quad or \quad X = (AB)'$$

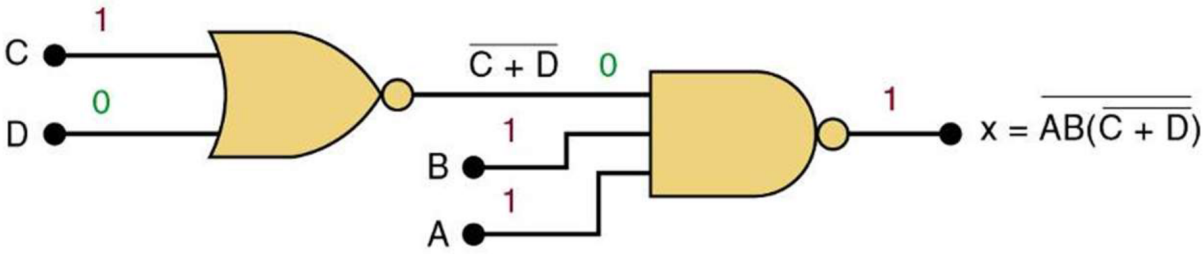
> 注意这里的括号不能省略.



		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

(c)

多次取反的输出，可以通过 NOR 和 NAND 的组合实现：



2.3.6 异或门

> XOR Gate
>
> X 指 Exclusive，排他
>
> 通常只支持两输入，如果要拓展到多输入，需要修改逻辑.

Similar to OR gate, except that when all inputs are 1s, the output will be 0

> 在「或」的基础上，保证「异」，输出才为 1.

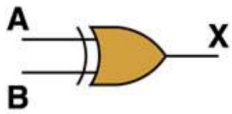
「两个输入不同」时为 1；「两个输入相同」时为 0.

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

Boolean Expression

$X = A \oplus B$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

拓展：多输入 XOR

修改逻辑. 当输入为 $A, B, C \cdots$ 时，输出 Y 满足

$$Y = A \oplus B \oplus C \cdots$$

解释：多输入 XOR 可以看作逐个计算 XOR 的结果. 例如，输入 A, B, C, D 时，

$$Y = ((A \oplus B) \oplus C) \oplus D$$

> 根据此逻辑，三输入时，当 A, B, C 均为 1，输出也为 1.

（2025.2.24）例：把 $Y = A \oplus B \oplus C$ 化为 SOP 形式.

$$\begin{aligned}
Y &= A \oplus B \oplus C \\
&= (A \oplus B) \oplus C \\
&= (A\overline{B} + \overline{A}B) \oplus C \\
&= A\overline{B} \oplus C + \overline{A}B \oplus C \qquad \text{XOR 满足分配律？} \\
&= \overline{A\overline{B}}C + A\overline{B}\,\overline{C} + \overline{\overline{A}B}C + \overline{A}B\overline{C} \\
&= (\overline{A} + B)C + A\overline{B}\,\overline{C} + (A + \overline{B})C + \overline{A}B\overline{C} \\
&= \overline{A}C + BC + A\overline{B}\,\overline{C} + AC + \overline{B}C + \overline{A}B\overline{C} \\
&= C + A\overline{B}\,\overline{C} + \overline{A}B\overline{C}
\end{aligned}$$

这个结果是**错误的**，因为不能默认 XOR 对 OR 运算满足分配律.

> 但是它满足交换律和结合律. 可以通过列真值表证明或代数证明.

正确做法：列真值表，直接从真值表中得到 SOP:

$$\begin{aligned}
Y &= A \oplus B \oplus C \\
&= A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,\overline{B}C + ABC
\end{aligned}$$

2.3.7 功能完备

> Functionally Complete

如果一个逻辑门集合是功能完备的，那么用该集合中的门可以构造出任意布尔函数，也就是实现任何所需要的逻辑运算或逻辑电路.

在布尔代数中，逻辑运算（如 AND、OR、NOT）是构成任意布尔表达式的基本元素. 当一组逻辑门能用来构造所有这些基本运算时，就称这组门具有功能完备性.

① 异或门

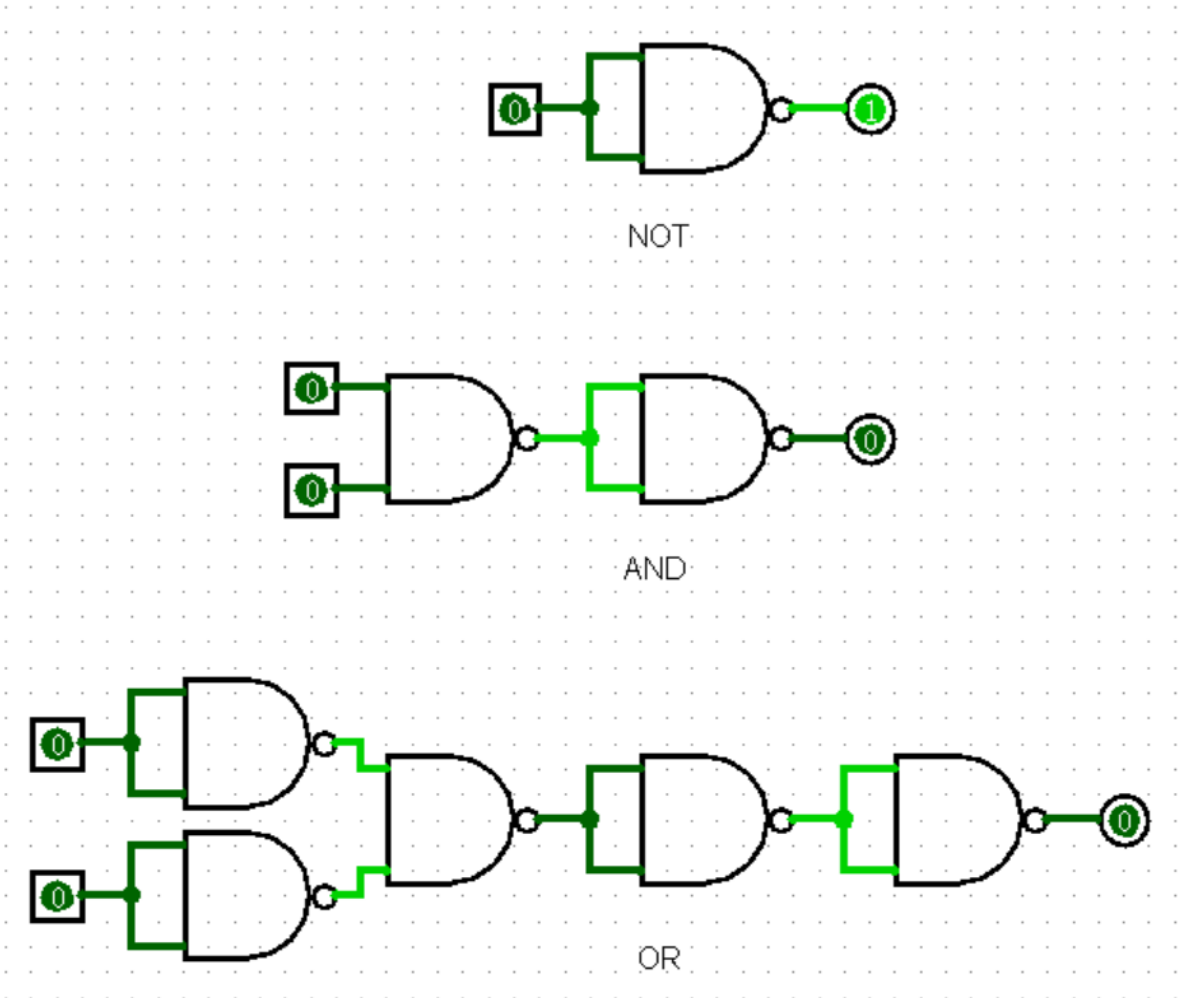
> XOR

单独使用异或门无法构造出所有可能的布尔函数，异或门本身**不具备**功能完备性.

> 待证明.

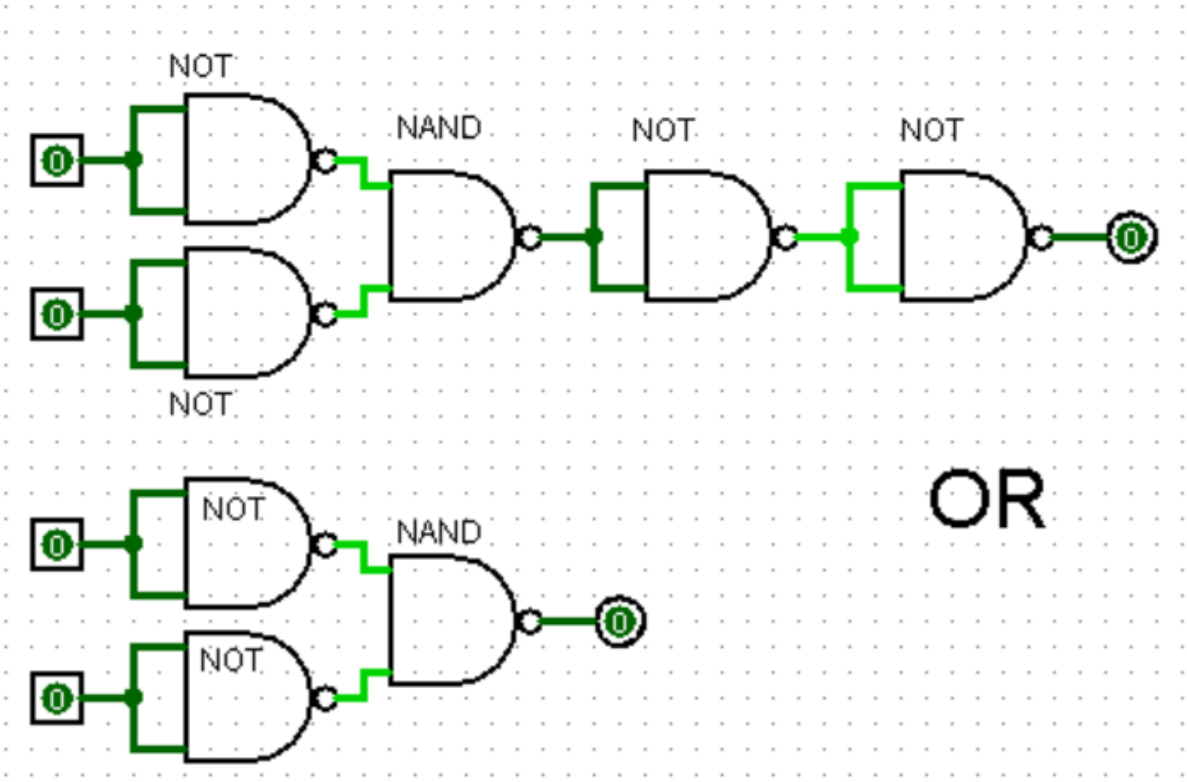
② 与非门

> NAND



> 双输入自身构造 NOT，自己取反构造 AND，德摩根律构造 OR.
>
> $\overline{x+y}=\overline{x}\cdot\overline{y}\Rightarrow x+y=\overline{\overline{x}\cdot\overline{y}}$.

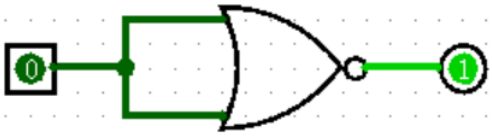
注意，德摩根律构造 OR 门时，AND + NOT 可以直接用 NAND 表示：



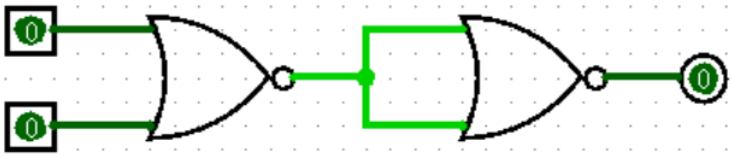
> OR 的构造法也可以参考 `2.3.8 替代逻辑门`，OR 变 AND，加三个 bubble，左边两个 bubble 用 NOT 实现，右边的 bubble 用 NAND 自带的 bubble 实现.

③ 或非门

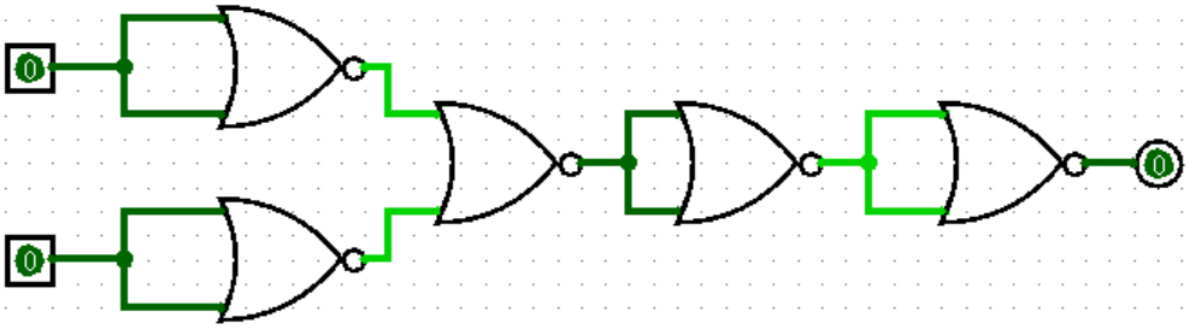
> NOR



NOT



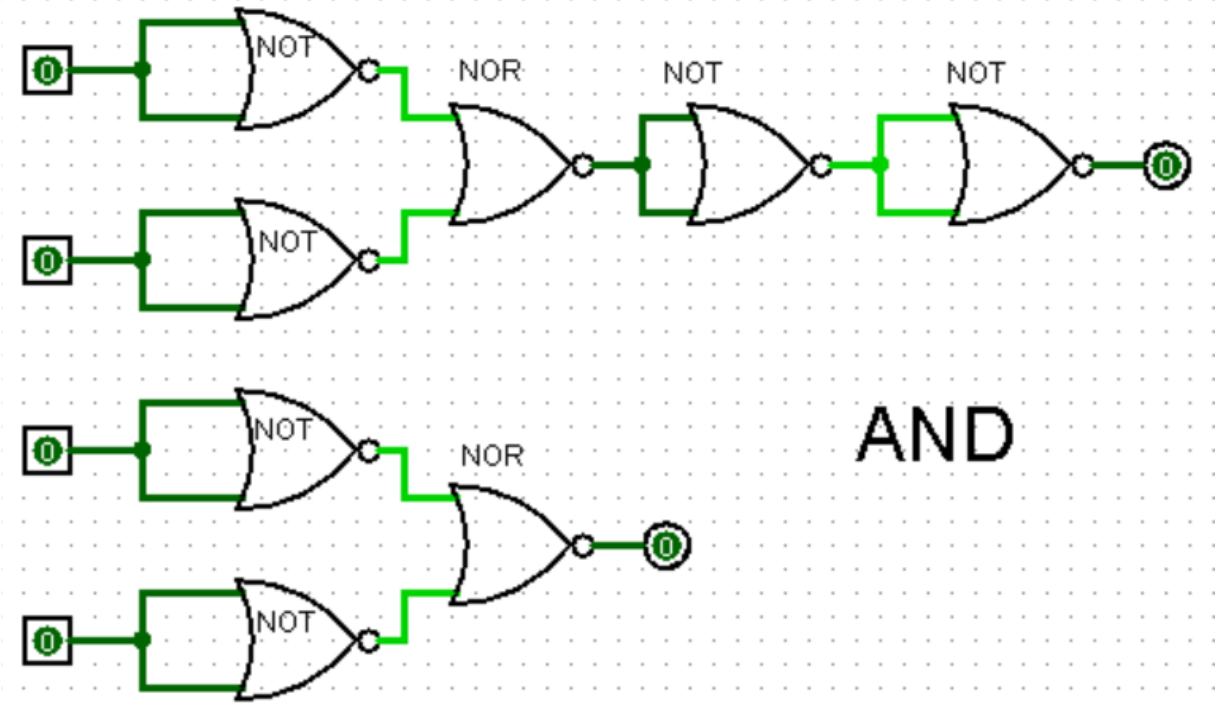
OR



AND

> 双输入自身构造 NOT，自己取反构造 OR，德摩根律构造 AND.
>
> $\overline{x \cdot y} = \overline{x} + \overline{y} \Rightarrow x \cdot y = \overline{\overline{x} + \overline{y}}.$

注意，德摩根律构造 AND 门时，OR + NOT 可以直接用 NOR 表示：



> AND 的构造法也可以参考 `2.3.8 替代逻辑门`，AND 变 OR，加三个 bubble，左边两个 bubble 用 NOT 实现，右边的 bubble 用 NOR 自带的 bubble 实现.

④ 门组合

> 待证明.

AND 和 NOT

OR 和 NOT

XOR 和 AND

XOR 和 OR

非功能完备：例如单独使用 AND 或 OR，无法实现 NOT 运算.

2.3.8 替代逻辑门

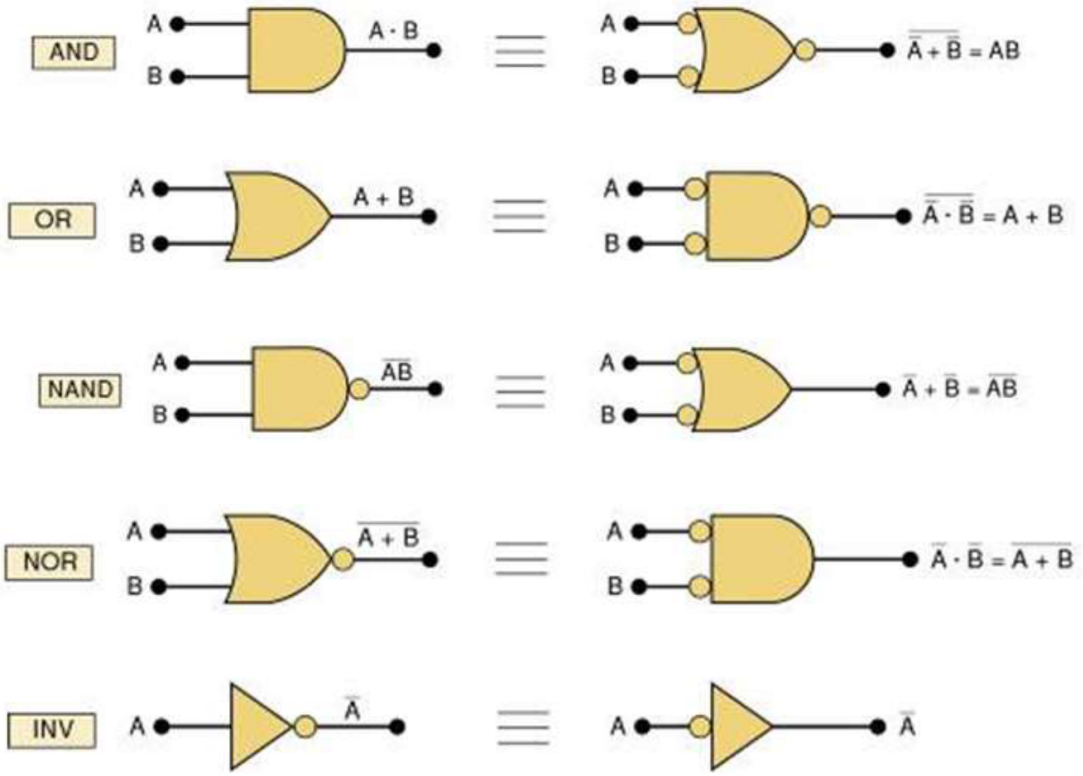
> Alternative Logic Gates

Alternative logic gate is another type of logic gate that produces the same output as the original logic gate

It will be very useful when the original logic gate is not available

To convert a standard symbol to an alternate for AND, OR, NAND, NOR gates

- Change the basic gate from AND to OR, or OR to AND
- Add an inversion bubble if there is no inversion
- Remove the inversion bubble if there is an inversion



2.4 布尔表达式

> Boolean Expression

布尔表达式（Boolean Expression）是用来表示逻辑值的表达式，其值通常为 **真（True）** 或 **假（False）**.

2.4.1 组成

布尔表达式由以下几部分组成：

① 布尔变量

- 变量的值只能是 True 或 False.
- 例如：A, B, C

② 布尔常量

- 固定的布尔值：True 或 False.
- 例如：1 表示 True, 0 表示 False.

③ 布尔运算符

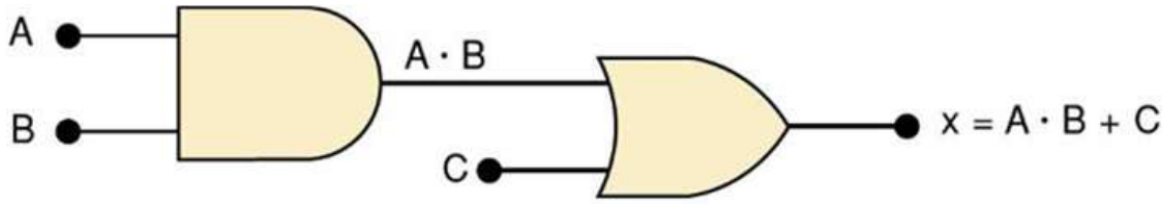
- 用于连接布尔变量或常量的运算符.
- 与或非，异或，条件，等价

④ 括号

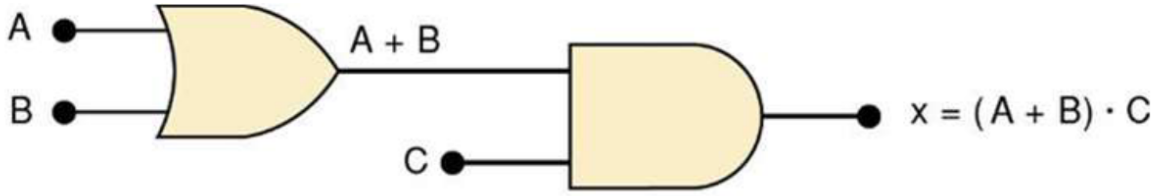
- 明确表达式优先级.

2.4.2 运算规则

If an expression contains both AND and OR gates, the AND operation will be performed first.



Unless there is a parenthesis in the expression



Rules of evaluating a Boolean expression:

- Perform all inversions of single terms
- Perform all operations with parenthesis
- Perform AND operation before an OR operation unless parenthesis indicate otherwise
- If an expression has a bar over it, perform operations inside the expression, and then invert the result

> 优先级：取反 > 括号 > 与 > 或
>
> 对表达式取反：先计算该式，然后将结果取反.

2.4.3 化简

> Boolean Algebra

① 与零律

$$x \cdot 0 = 0$$

② 与一律

$$x \cdot 1 = x$$

③ 幂等律

> Idempotence Law
>
> 这里的「幂等」和数学的「幂」不一样. 幂等的含义：元素 s 满足 $s * s = s$，则称 s 相对二元运算符 $*$ 是幂等的. 若集合 S 所有元素都相对 $*$ 幂等，则称二元运算符 $*$ 是幂等的.
>
> 简单来说：自己作用于自己，结果还是自己，这就是幂等性.

$$
\begin{aligned}
x \cdot x &= x \\
x + x &= x
\end{aligned}
$$

④ 互补律

> 公理，又称「补元律」，公理假设是对于任意元素 x，都存在其补 $\overline{x}$，满足：

$$
\begin{aligned}
x \cdot \overline{x} &= 0 \\
x + \overline{x} &= 1
\end{aligned}
$$

⑤ 或零律

$$
x + 0 = x
$$

⑥ 或一律

$$
x + 1 = 1
$$

⑦ 交换律

> 公理

$$
\begin{aligned}
x + y &= y + x \\
x \cdot y &= y \cdot x
\end{aligned}
$$

⑧ 结合律

> 公理

$$
\begin{aligned}
& x + (y + z) \\
= \ & (x + y) + z \\
= \ & x + y + z
\end{aligned}
$$

$$
\begin{aligned}
& x(yz) \\
= \ & (xy)z \\
= \ & xyz
\end{aligned}
$$

⑨ 分配律

> 公理

$$
\begin{aligned}
x(y + z) &= xy + xz \\
(w + x)(y + z) &= wy + xy + wz + xz
\end{aligned}
$$

⑩ 吸收律

$$
\begin{aligned}
x + xy &= x \\
x + \overline{x}y &= x + y \\
\overline{x} + xy &= \overline{x} + y
\end{aligned}
$$

⑪ 德摩根律

$$
\begin{aligned}
\overline{x + y} &= \overline{x} \cdot \overline{y} \\
\overline{x \cdot y} &= \overline{x} + \overline{y} \\
\overline{x + y + z} &= \overline{x} \cdot \overline{y} \cdot \overline{z} \\
\overline{x \cdot y \cdot z} &= \overline{x} + \overline{y} + \overline{z}
\end{aligned}
$$

⑫ 双重否定律

$$\overline{\overline{x}} = x$$

⑬ 恒等式一

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

证明：

$$\begin{aligned}
A + B &= \overline{\overline{A + B}} && \text{双重否定律} \\
&= \overline{\overline{A}\,\overline{B}} && \text{德摩根律} \\
&= \overline{\overline{A}\,\overline{B} + 0} && \text{或零律} \\
&= \overline{\overline{A}\,\overline{B} + \overline{A}A} && \text{互补律} \\
&= \overline{\overline{A}(\overline{B} + A)} && \text{分配律} \\
&= A + \overline{\overline{B} + A} && \text{德摩根律} \\
&= A + \overline{A}B && \text{德摩根律} \\
&= A(B + \overline{B}) + \overline{A}B && \text{互补律} \\
&= AB + A\overline{B} + \overline{A}B && \text{分配律}
\end{aligned}$$

> 注意，标红部分是吸收律，前面都是吸收律的证明.

法二：

$$\begin{aligned}
A + B &= A(B + \overline{B}) + B(A + \overline{A}) && 1 \text{ 的代换 / 互补律} \\
&= AB + A\overline{B} + BA + B\overline{A} && \text{分配律} \\
&= AB + A\overline{B} + \overline{A}B && \text{幂等律 / 交换律}
\end{aligned}$$

> 核心：1 的代换.
>
> 思考：这里想到用 1 的代换，其实是想还原成规范形 SOP，如果一个恒等式是正确的，那么证明它可以从把两边都还原成规范形入手，因为规范形只有一个，天然适合用来作为恒等变换的桥梁.

⑭ 恒等式二

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

证明：

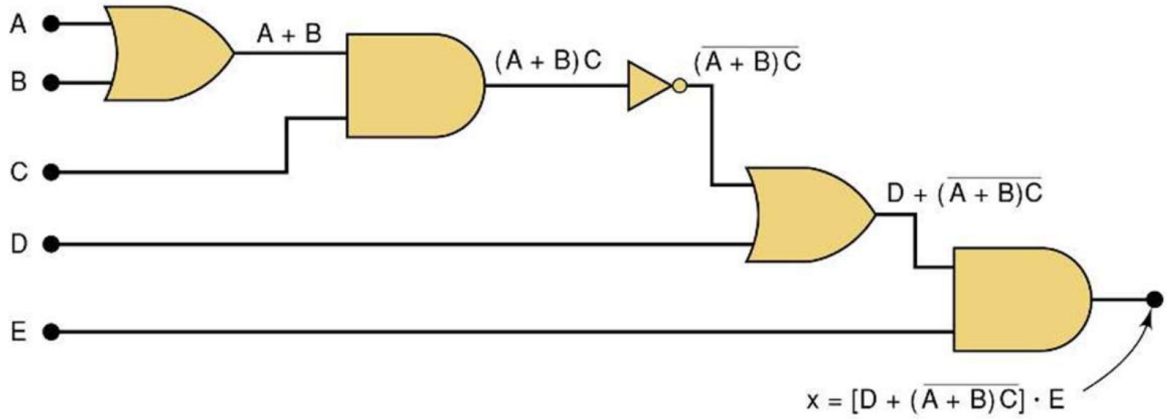
$$\begin{aligned}
AB + BC + AC &= AB(C + \overline{C}) + BC(A + \overline{A}) + AC(B + \overline{B}) \\
&= ABC + \overline{A}BC + A\overline{B}C + AB\overline{C}
\end{aligned}$$

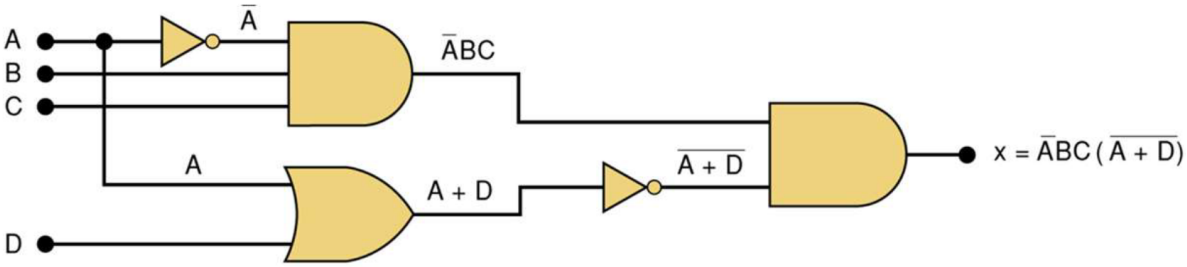
> 思想和上面法二类似.

2.4.4 表示逻辑电路

> Logic Circuits to Boolean Expression

Logic circuits can be expressed by Boolean expressions.





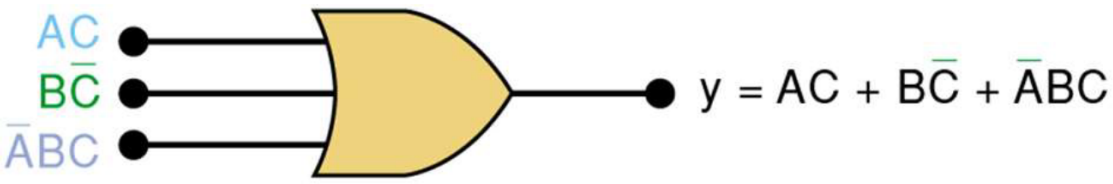
2.4.5 构造逻辑电路

> From the Boolean expression, we can also construct the logic circuit directly.

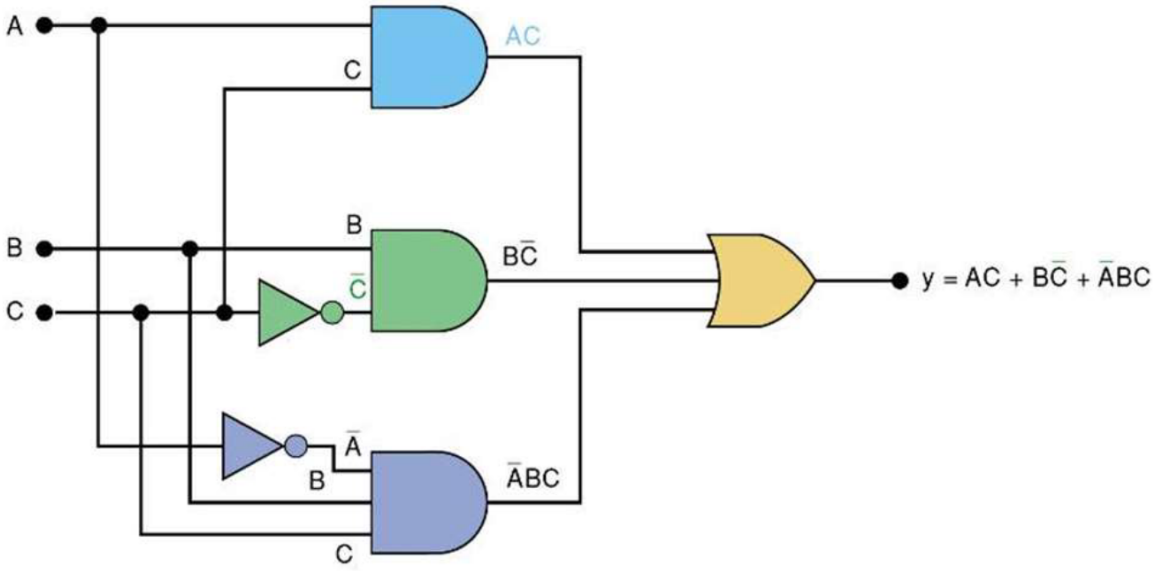
构造逻辑：从右往左.

Example: 构造 $Y = AC + BC' + A'BC$

Step 1: Use 3-input OR gate as below.



Step 2: Further implement the 3 inputs by using 2-input and 3-input AND gates accordingly.



2.5 逻辑电平

An input/output can be active-HIGH or active-LOW

- Active-HIGH is indicated by an input/output has no inversion bubble
- Active-LOW is indicated by an input/output has an inversion bubble

For exampl, the Enable (E) signal of a memory

- A bar over a signal means active low
- Absence of a bar means active high

2.5.1 高电平有效

> Active-high

当信号处于高电平（通常代表逻辑 1）时，表示该信号处于激活或有效状态.

例:

- 如果一个使能信号（Enable）为高时启动电路功能，那么这个使能信号就是高电平有效.
- 一个 LED 电路中，当控制信号为高电平时 LED 点亮，那么该信号也是高电平有效.

标识方式：在一些电路图或芯片手册中，通常标记为 “EN” 或 "Enable"，默认理解为高电平激活.

2.5.2 低电平有效

> Active-low

当信号处于低电平（通常代表逻辑 0）时，表示该信号处于激活或有效状态.

举例：

- 复位信号（Reset）常常设计为低电平有效，即当复位信号拉低时，系统将执行复位操作.
- 一个报警系统，如果检测到低电平输入则触发报警，则该输入信号为低电平有效.

标识方式：通常在名称前加斜杠 `/` 或波浪号 `~`，例如 `/RESET` 或 `~CS` （片选信号），以表明该信号为低电平有效.

2.6 集成电路

2.6.1 数据手册

> Datasheet

2.6.2 传播延迟

> Propagation Delay

2.6.3 噪声裕度

> Noise Margin

2.6.4 扇入扇出

> Fan-in and Fan-out

Lec 3 门级优化

> Gate Level Minimization

3.1 最小项和最大项

> Minterms & Maxterms

在布尔函数表达中，最小项（minterm）和最大项（maxterm）是构成规范形/标准形表达式的基本单元.

- A binary variable may appear either in its normal form x, or in its complement form x'

> 每个二元变量都以原变量或补变量的形式出现.

3.1.1 最小项

> Minterm

对于 n 个布尔变量的系统，每个最小项是所有变量的一个与（AND）运算，其中每个变量以原变量或其补变量的形式出现. 最小项在真值表中仅在唯一的一组输入组合下输出 1，其余情况下都输出 0.

任意布尔函数都可以写成若干最小项的或（OR），这就是 SOP（与项之和）形式.

- If there are 2 variables x and y, there are 4 possible combinations: $x'y', x'y, xy', xy$
- Each of these 4 AND terms is called a minterm, or standard product
- If there are N variables, there are 2^N minterms

  > 每个变量有原变量和补变量两种选择，共 N 个变量，因此是 2^N 种组合.

- A symbol for each minterm is denoted as m_j, where the subscript j denotes the decimal equivalent of the binary number of the minterm designated

3.1.2 最大项

> Maxterm

对于 n 个布尔变量的系统，每个最大项是所有变量的一个或（OR）运算，其中每个变量以原变量或其补变量的形式出现. 最大项在真值表中仅在唯一的一组输入组合下输出 0，其余情况下都输出 1.

任意布尔函数都可以写成若干最大项的与（AND），这就是 POS（或项之积）形式.

- If there are 2 variables x and y, there are another 4 possible combinations:
  $x+y, x+y', x'+y, x'+y'$
- Each of these 4 OR terms is called a maxterm, or standard sum
- If there are N variables, there are 2^N maxterms
- A symbol for each maxterm is denoted as M_j, where the subscript j denotes the decimal equivalent of the binary number of the maxterm designated

- m_j is the complement of M_j

> 对于相同变量组合，最小项和最大项的逻辑值正好相反. 也就是说，最小项的值等于对应最大项的逻辑非（补运算）.

3.2 规范形和标准形

> Canonical and Standard Forms

数学和计算机科学中，数学对象的**标准形**、**规范形**是将该对象作为表达式呈现的标准方式. 通常来说，它提供了对象的最简单的表示，并允许以独特的方式识别.

「规范（canonical）」与「标准（normal）」的区别因领域而异，大多数时候规范形都规定了对象的唯一表示形式，而标准形则不要求唯一性.

例子：三变量布尔函数

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

For example, we have the truth table of the following functions f_1

Functions of Three Variables

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A Boolean function can be expressed algebraically in terms of either minterms or maxterms

Functions of Three Variables

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

In terms of minterms, we have

$$\begin{aligned} f_1(x,y,z) &= x'y'z + xy'z' + xyz \\ &= m_1 + m_4 + m_7 \\ &= \sum m(1,4,7) \\ f_1'(x,y,z) &= \sum m(0,2,3,5,6) \end{aligned}$$

> $f_1'(x,y,z) = \sum m(0,2,3,5,6)$ 的意思是，只要 m_0, m_2, m_3, m_5, m_6 中有一个取 1，等号右边就取 1，即 $f_1'(x,y,z) = 1$，对应 $f_1(x,y,z) = 0$.
>
> $f_1'(x,y,z)$ 也可以理解为把真值表中 f_1 的值全部进行 $0, 1$ 互换，然后用新的 SOP 表示.

Functions of Three Variables

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

In terms of maxterms, we have

$$\begin{aligned} f_1(x,y,z) &= (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z) \\ &= M_0M_2M_3M_5M_6 \\ &= \prod M(0,2,3,5,6) \\ f_1'(x,y,z) &= \prod M(1,4,7) \end{aligned}$$

> $f_1'(x,y,z) = \prod M(1,4,7)$ 的意思是，只要 M_1, M_4, M_7 中有一个取 0，等号右边就取 0，即 $f_1'(x,y,z) = 0$，对应 $f_1(x,y,z) = 1$.
>
> $f_1'(x,y,z)$ 也可以理解为把真值表中 f_1 的值全部进行 $0,1$ 互换，然后用新的 POS 表示.

综上可以看出最小项和最大项的特点：

最小项是乘积形式，所有输入完全匹配时输出 1，任意布尔函数 f 可以写成若干最小项的或，或的特点是只要有一项为 1 整个值就为 1，当这若干最小项中某一项成功匹配，f 就输出 1；因此采用 SOP 时，最小项的选择就是所有能让 f 输出 1 的 input 组合而成的最小项. 而 f 只能输出 0 或 1，当 f 不输出 1（所有让 f 输出 1 的最小项都没有匹配到），则匹配到了不能让 f 输出 1 的 input 组合，也就是真值表中 f 值为 0 的那些 input 组合，从而实现了真值表的每一行与 SOP 表示 f 的一一映射.

最大项是和的形式，所有输入完全补匹配时输出 0，任意布尔函数可以写成若干最大项的与，与的特点是只要有一项为 0 整个值就为 0，当这若干最大项中某一项成功补匹配，f 就输出 0；因此采用 POS 时，最大项的选择就是所有能让 f 输出 0 的 input 补组合而成的最大项. 而 f 只能输出 0 或 1，当 f 不输出 0（所有让 f 输出 0 的最大项都没有补匹配到），则补匹配到了不能让 f 输出 0 的 input 组合，也就是真值表中 f 值为 1 的那些 input 组合，从而实现了真值表的每一行与 POS 表示 f 的一一映射.

Example 1

Given a truth table:

A	X	
0	0	Condition 0
1	1	Condition 1

By using Minterm, $1 = \text{Normal}$, $0 = \text{Complement}$

- $X = A$
- If $A = 1$, then $X = 1$
- If Condition 1 happens, $X = 1$; if Condition 1 does not happen, $X = 0$.

By using Maxterm, $0 = \text{Normal}$, $1 = \text{Complement}$

- $X = A$
- If $A = 0$, then $X = 0$
- If Condition 0 happens, $X = 0$; if Condition 0 does not happen, $X = 1$.

> SOP 关注让 $f = 1$ 发生的组合，POS 关注让 $f = 0$ 发生的组合.

Example 2

Given a truth table:

A	B	X	
0	0	0	Condition 0
0	1	1	Condition 1
1	0	0	Condition 2
1	1	1	Condition 3

By using Minterm, $1 = \text{Normal}$, $0 = \text{Complement}$

- $X = A'B + AB$
- If $A'B = 1$ OR $AB = 1$, then $X = 1$
- If either Condition 1 or Condition 3 happens, $X = 1$; if both Condition 1 and Condition 3 do not happen, $X = 0$.

By using Maxterm, $0 = \text{Normal}$, $1 = \text{Complement}$

- $X = (A + B)(A' + B)$
- If $A + B = 0$ OR $A' + B = 0$, then $X = 0$
- If either Condition 0 or Condition 2 happens, $X = 0$; if both Condition 0 and Condition 2 do not happen, $X = 1$.

3.2.1 标准形

> Standard Forms

Standard Sum-Of-Products (SOP) forms

- Equations are written as AND terms summed with OR operators
- $F = xyz + xy' + z'$

Standard Product-Of-Sums (POS) forms

- Equations are written as OR terms multiplied with AND operators
- $F = (x + y + z)(x + y')(z')$

注意：标准形可能有多种表示，例如

$$\begin{aligned} F &= xyz + xy' + z' \\ &= xyz + xy'(z + z') + z' \\ &= xyz + xy'z + xy'z' + z' \\ &= xz(y + y') + z'(xy' + 1) \\ &= xz + z' \end{aligned}$$

仍然是 SOP 形式. 但如果规定使用规范形 SOP（即 Sum of minterms），则有唯一写法（加号左右互换算同一种），因为真值表是唯一的.

3.2.2 规范形

> Canonical Forms

Sum of minterms: $F = xyz + xy'z + x'yz'$

Product of maxterms: $F = (x + y + z)(x + y' + z)(x' + y + z')$

3.2.3 混合形

> Mixed Forms

Mixed forms is neither SOP nor POS: $F = (xy + z)(x + y'z) + (x'y + z')$

3.3 门级优化

> Gate Level Minimization

门级优化是数字电路设计中对逻辑综合后生成的门级电路网表（gate-level netlist）进行进一步改进的过程. 主要目标在于提高电路性能、降低功耗、减少面积以及优化时序等. 下面介绍一些关键概念和常见方法：

优化目标

- 减小面积：通过去除冗余逻辑、合并等效门电路等方式，降低逻辑门数量，从而减少芯片面积.
- 降低功耗：优化逻辑结构与信号路径，以减少动态功耗和静态功耗，保证电路在低功耗工作环境下运行.
- 提升时序性能：调整逻辑门和互连的分布，优化关键路径，确保时钟频率满足设计要求，同时减少路径延迟和抖动问题.
- 提高电路可靠性：通过冗余消除和故障分析，确保在各种工艺、温度和电压范围内电路的正确性和稳定性.

常见方法

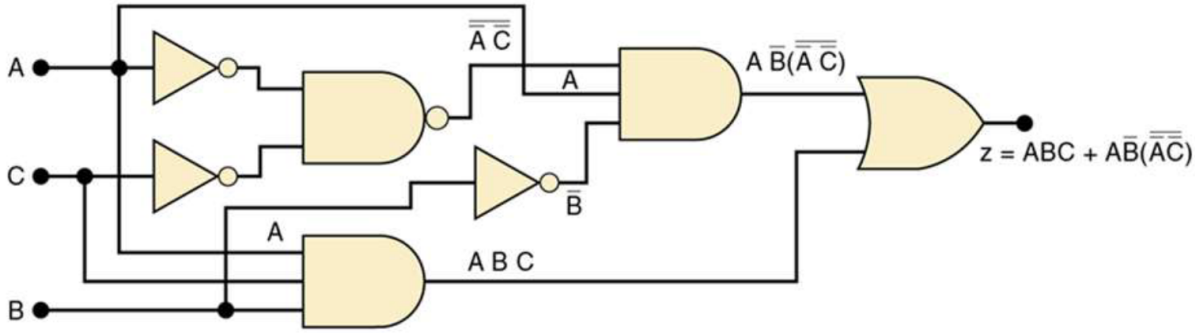
- 逻辑简化：利用布尔代数、卡诺图（Karnaugh Map）和 Quine-McCluskey 算法等方法对逻辑表达式进行化简，从而减少逻辑门的数量和级数.
- 常数传播（Constant Propagation）：在电路中将已知输入或常量信号传递并替换相关的逻辑门操作，消除不必要的逻辑运算.
- 冗余消除：移除电路中不影响输出的多余门，减少资源浪费. 例如，逻辑合理化和故障覆盖率分析都能够帮助发现冗余部分.
- 重新结构化（Restructuring）：根据时序和面积要求对电路结构进行调整，如将深层次逻辑拆解成多级优化或者充分利用并行结构，平衡延迟和面积之间的矛盾.

本节主要关注 `常见方法 - 逻辑简化`，重点介绍使用布尔代数、卡诺图化简布尔表达式.

3.3.1 布尔代数法

> Simplifying A Logic Circuit

To simplify a logic circuit, so as to implement the same function with less logic gates:

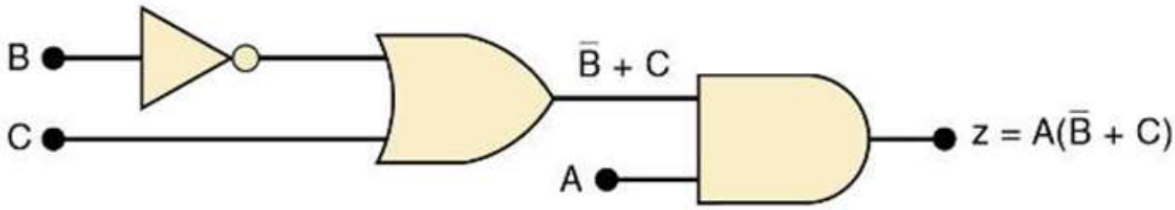


Before simplification: $z = ABC + AB'(A'C')'$

布尔代数化简:

$$
\begin{aligned}
z &= ABC + AB'(A'C')' \\
&= ABC + AB'(A + C) \\
&= ABC + AB'A + AB'C \\
&= ABC + AB' + AB'C \\
&= AC(B + B') + AB' \\
&= AC + AB'
\end{aligned}
$$

After simplification: $z = A(C + B')$



> 注意这里 Before simplification 的表达式是通过电路图看出来的，它可能是标准形 / 规范形 / 混合形 中任意一种，对于复杂电路容易出错. 下面的流程给出另一种写原表达式的方法（Step 1-3），这种方法对任意布尔表达式都给出 SOP 规范形表示，普适性强.

布尔代数法化简流程

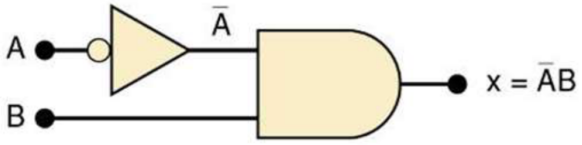
- Step 1: setup its truth table as in Figure (a)

  > 真值表可能由题目提供，或者自己开 Logisim 测.

- Step 2: write down the AND term for each row where output is 1
- Step 3: combine the AND terms in SOP form
- Step 4: simplify the expression by Boolean algebra, if possible
- Step 5: implement the simplified circuit as in Figure (b)

A	B	x
0	0	0
0	1	1
1	0	0
1	1	0

(a)



(b)

Example:

Step 1 to 3

Truth table.

A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \overline{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\overline{B}C$
1	1	0	1	$\rightarrow AB\overline{C}$
1	1	1	1	$\rightarrow ABC$

AND terms for each case where output is 1. (2)

(1)

SOP expression for the output:

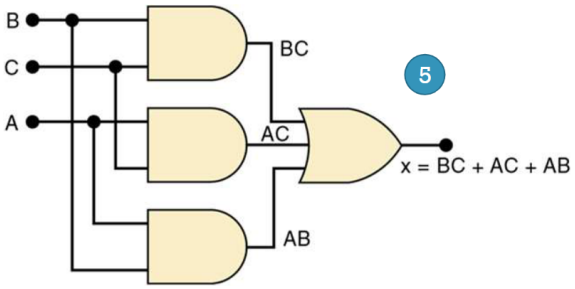
$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$ (3)

Step 4 to 5

A	B	C	X
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

(4)

$x = BC + AC + AB$



> 这里 Step 4 可以理解为输入大于等于两个 1，则 x 为 1. 详细证明见 `2.4.3 化简 ⑩ 恒等式二` .

布尔代数法的缺点：结果不一（相同的规范形，可能化简为不同的标准形）

例如：Simplify the Boolean function, $F(A,B,C) = \sum m(0,2,3,4,5,7)$

Calculation 1:

$$\begin{aligned} F &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + A\overline{B}C + ABC \\ &= (\overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C}) + (\overline{A}BC + ABC) + (A\overline{B}C + A\overline{B}\,\overline{C}) \\ &= \overline{A}\,\overline{C} + BC + A\overline{B} \end{aligned}$$

Calculation 2:

$$
\begin{aligned}
F &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + A\overline{B}C + ABC \\
&= (\overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}) + (\overline{A}BC + \overline{A}B\overline{C}) + (A\overline{B}C + ABC) \\
&= \overline{B}\,\overline{C} + \overline{A}B + AC
\end{aligned}
$$

Both calculations are correct and having the same numbers of terms, is there any systematic way to simplify it?

> 布尔代数化简不成体统，不同人给出不同答案，我们想要统一的标准，因此引入卡诺图法. 当然，即使是卡诺图也难以实现完全唯一的化简结果，因为本质上最简表达式可能存在多个等价形式.
>
> 后面我们可以看到，对于上述两种计算，引入卡诺图也解决不了；但是卡诺图给我们提供了统一的化简思路，比布尔表达式更形象，尤其是项数很多的时候. 由于卡诺图按格雷码顺序排列单元格，相邻的 1 对应的两个最小项可以合并，这是通过拓展维度（卡诺图一般是二维，真值表或布尔表达式都是一维线性排列）实现的，可合并的项无论在布尔表达式中相隔多远，在卡诺图中都会变为相邻.

3.3.2 卡诺图法

> K-map Method / Karnaugh Map Method

卡诺图是把布尔函数的真值表转化为一个二维图形表示的方法. 它的每个单元格代表逻辑函数中的一个特定的变量组合. 通过科学地排列这些组合（通常利用格林码确保相邻单元格仅一位不同），可以方便地发现相邻单元中逻辑值相同的区域.

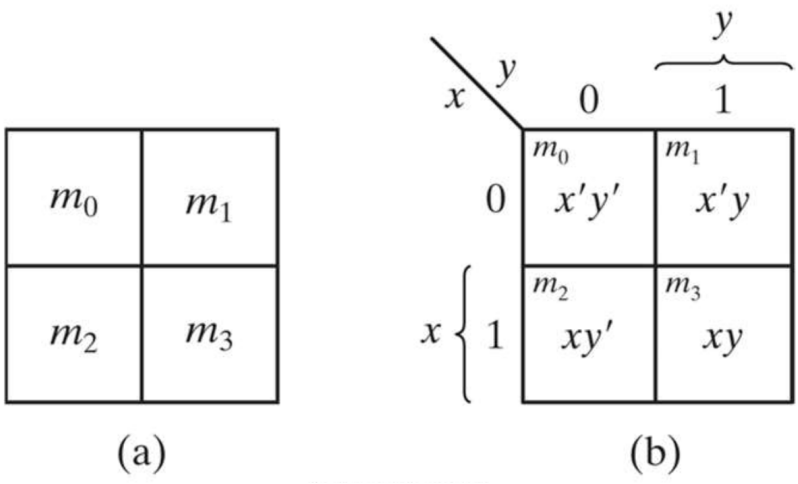
目的：通过合并相邻的 1（或 0，视具体化简目标），从而得到包含较少逻辑变量的表达式，简化逻辑电路的设计，降低硬件实现的复杂度和成本.

- K-map method is a graphical method of simplifying logic equations or truth tables
- Theoretically, it can be used for any number of input varibles, but pratically limited to 5 or 6 variables

① 双变量

> 2-Variable K-map

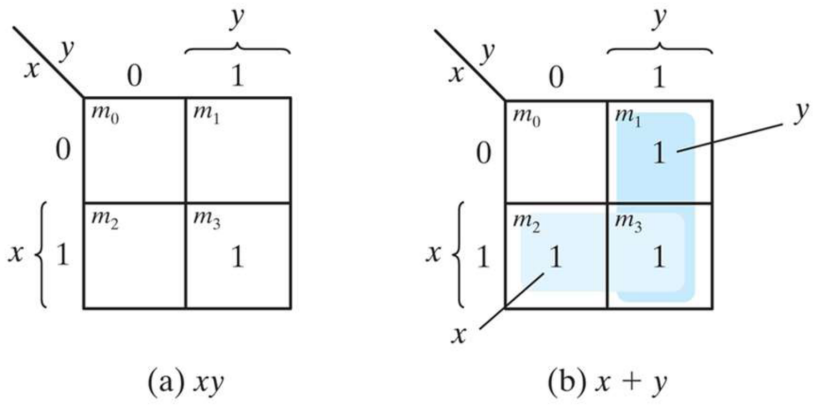
For 2 input variables, x and y



For the following expressions, we have the K-maps below:

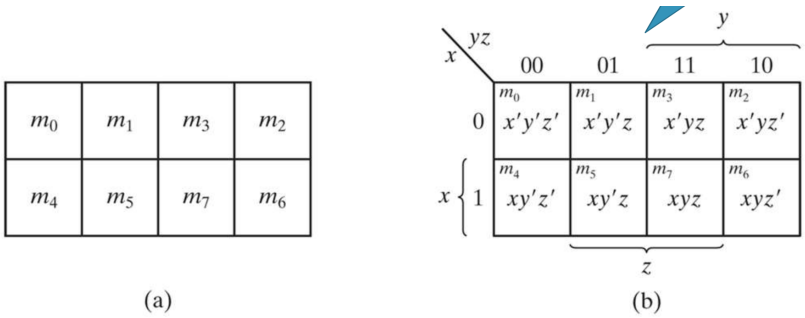
(a) xy

(b) $x + y$

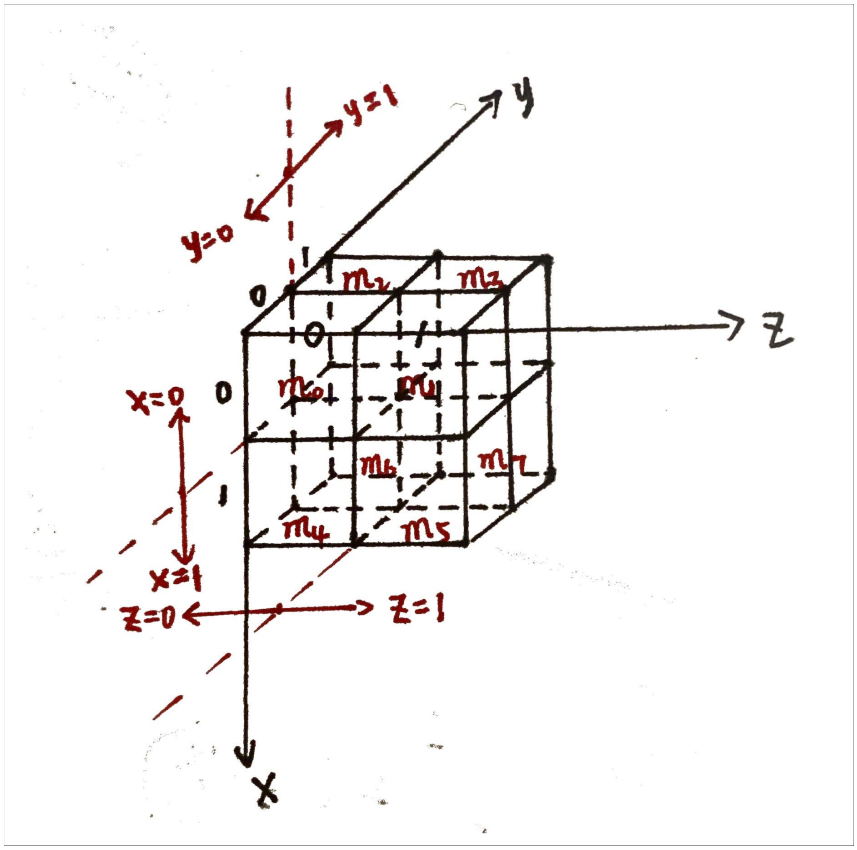


② 三变量

> 3-Variable K-map
>
> 这个竖着（4×2）也可以，看习惯.

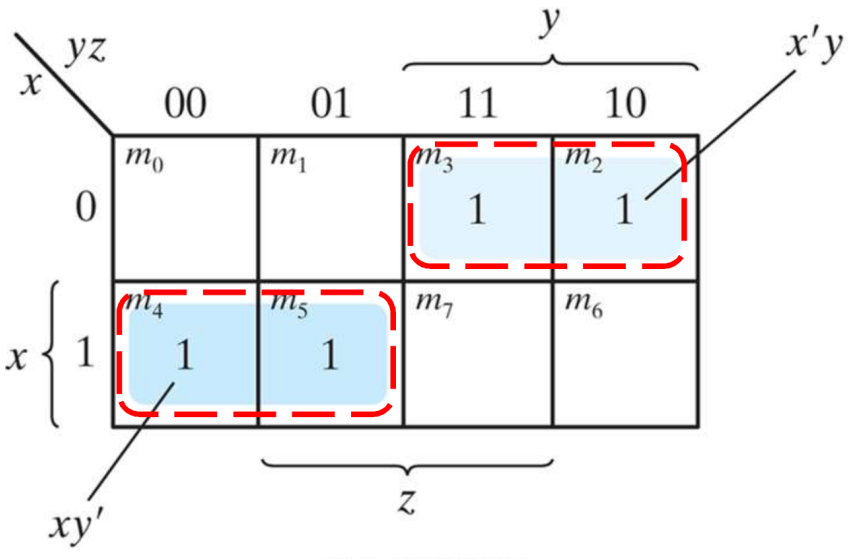


注意最右侧和最左侧也算相邻，可以对折成立方体来理解：



Example 1:

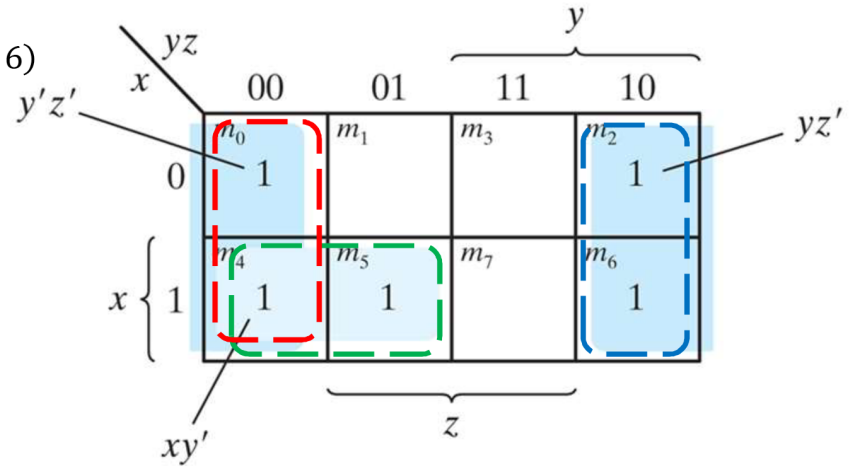
Simplify $F(x, y, z) = \sum m(2, 3, 4, 5)$



Ans: $F(x, y, z) = x'y + xy'$

Example 2:

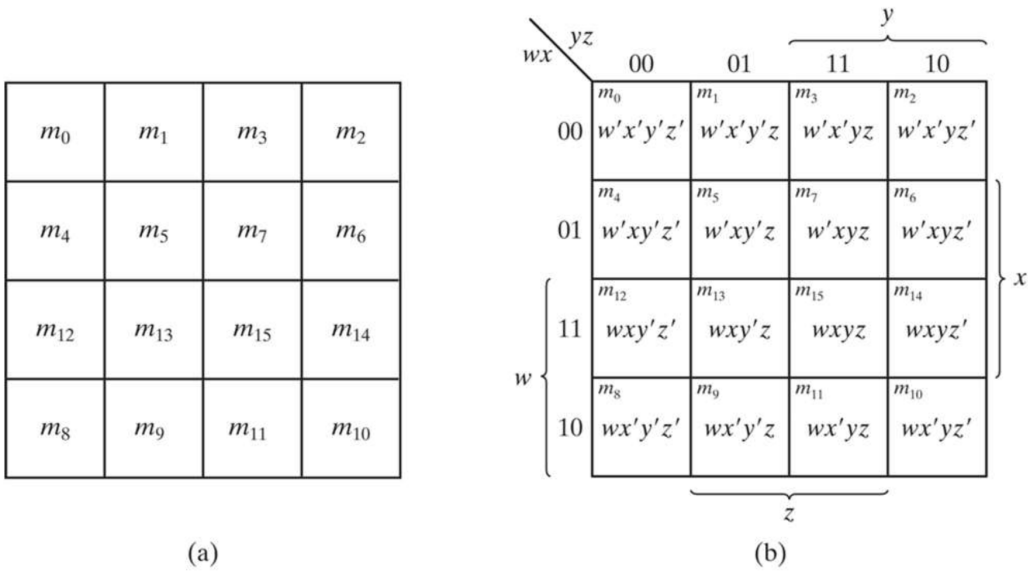
Simplify $F(x, y, z) = \sum m(0, 2, 4, 5, 6)$



Ans: $F(x, y, z) = z' + xy'$

③ 四变量

> 4-Variable K-map



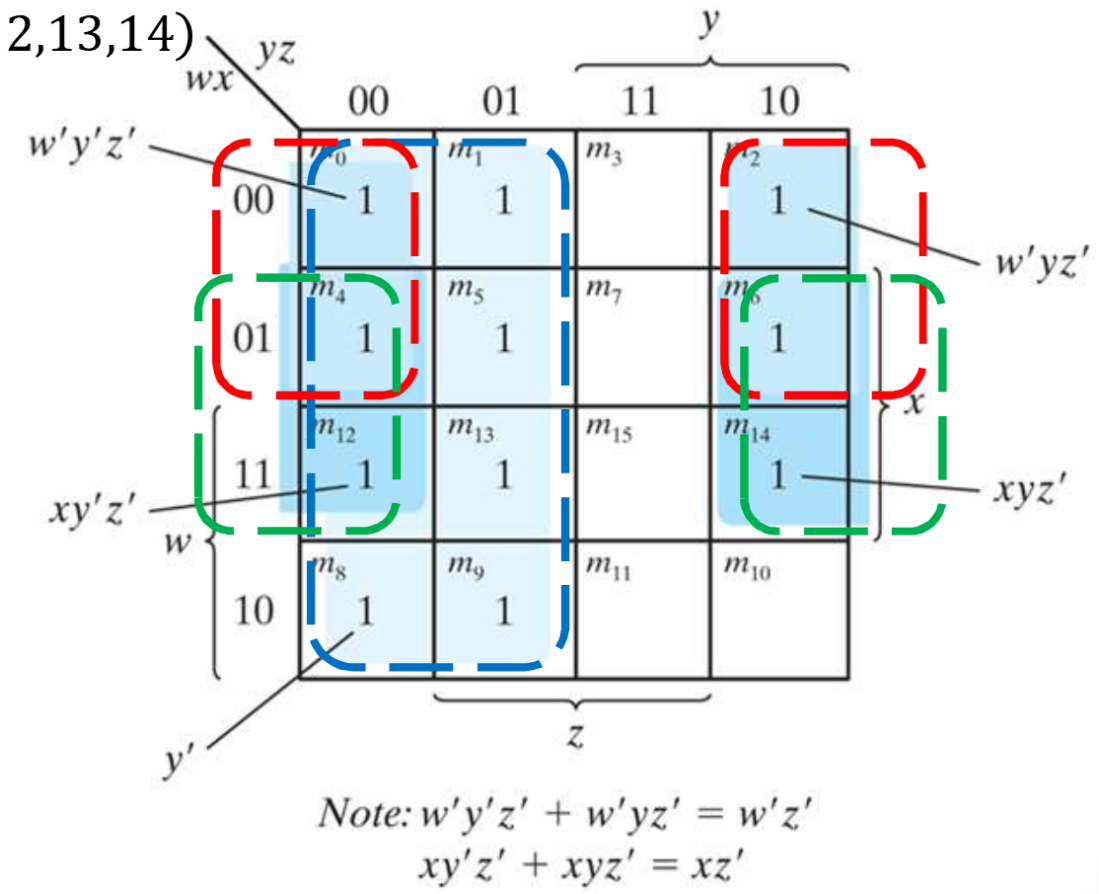
> 注意这里横纵坐标顺序要按标准（$00, 01, 11, 10$），这样可以根据标记的最小项（m_0 - m_{15}），将真值表高效填充进 K-map.

可以看成超立方体，或两个镜面对称的正方体，其中自己和自己的影子相邻.

> 画不出来，不画了.

Example 1:

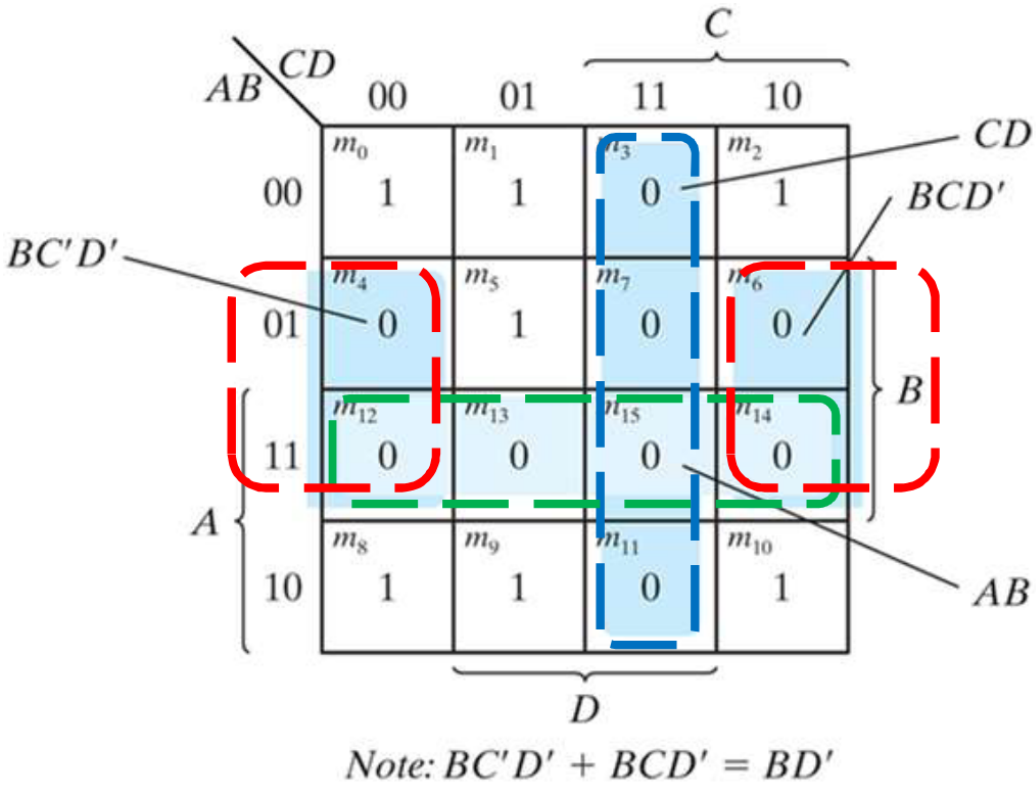
Simplify $F(w, x, y, z) = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$



Ans: $F(x, y, z) = y' + w'z' + xz'$

Example 2:

Simplify $F(A,B,C,D)=\sum m(0,1,2,5,8,9,10)$ in POS form



- $F'(A,B,C,D)=\sum m(3,4,6,7,11,12,13,14,15)$
- $F'(A,B,C,D)=BD'+CD+AB$
- $F(A,B,C,D)=(BD'+CD+AB)'$
- Ans: $F(A,B,C,D)=(B'+D)(C'+D')(A'+B')$

> 三变量德摩根律，见 `2.4.3 化简 ⊕ 德摩根律`.

Example 3:

Simplify $X(A,B,C,D)=\sum m(2,5,7,8,13,15)$ into the forms of

(a) SOP and

(b) POS.

列卡诺图:

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

SOP: $X(A,B,C,D)=BD+A'B'CD'+AB'C'D'$

POS:

法一（非标准答案）：

$X'(A,B,C,D)=B'D+BD'+A'B'C'D'+AB'CD'$

$X(A,B,C,D)=(B+D')(B'+D)(A+B+C+D)(A'+B+C'+D)$

法二（标准答案）：

> 按照化简流程，从多到少合并，所以左上角和右下角应该优先被合并，而不是单独取出.

$X'(A,B,C,D)=B'D+BD'+A'B'C'+AB'C$

$X(A,B,C,D)=(B+D')(B'+D)(A+B+C)(A'+B+C')$

④ 五变量*

> 5-Variable K-map
>
> 不考?

看成立方体对镜面作两次对称（四个立方体）.

Example 1: $F(A,B,C,D,E)=\sum(0,10,11,13,14,15,16,21,23,29,31)$

法一：一个 4×8

AB\CDE	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

> 待完成.

$$\begin{aligned}F(A,B,C,D,E)=&=\sum(0,10,11,13,14,15,16,21,23,29,31)\\&=B'C'D'E'+A'BD+BCE+ACE\end{aligned}$$

法二：两个 4×4

$A=0$

BC\DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$A=1$

BC\DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

想象把第二层叠到第一层上面. 重叠部分可以合成. 结果和法一一致.

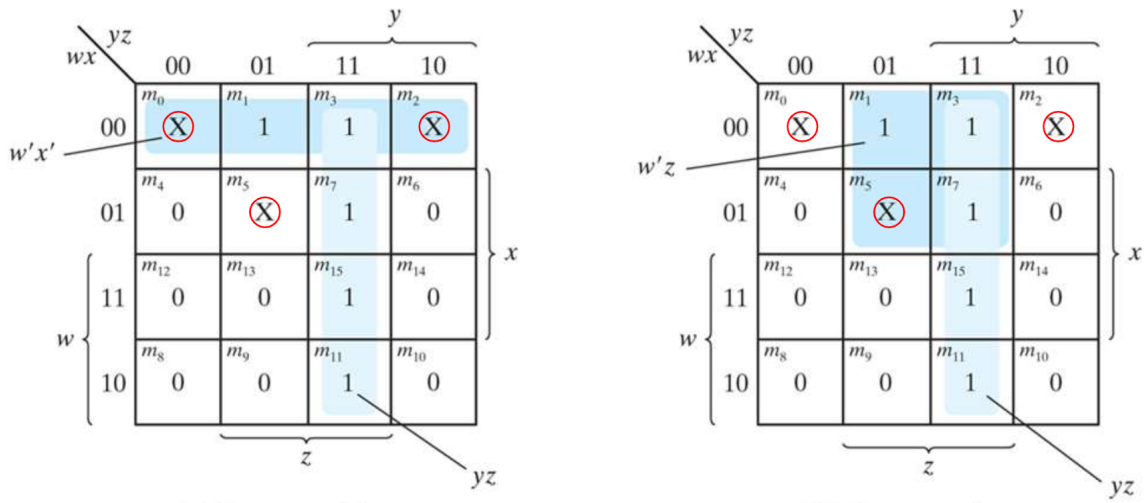
⑤ Don't Care

> K-map with "don't care"

当某些输入组合的对应输出是无关紧要的（输出 0 或 1 都可以），这些组合在 K-map 上对应单元格可以作为万能格，帮助化简.

Example:

Simplify $F(w,x,y,z)=\sum m(1,3,7,11,15)$ with m_0, m_2 and m_5 being "Don't Care".



(a) $F = yz + w'x'$ (b) $F = yz + w'z$

⑥ 化简流程

> Guidelines of K-map

- Group number of cells in powers of $2 \rightarrow 1, 2, 4, 8, 16, \ldots$
- Group as many cells as possible

  > 从多到少合并，先合并最多的.

- The larger the group, the smaller number of variables in that term
- Make use of the "Don't Care" states to get a larger group
- Make as few groups as possible to reduce the number of terms

⑦ 应用：警报设计

> Example: Alarm Design
>
> 待补充.

3.3.3 两级实现

> Two-Level Implementation

利用两个逻辑门层次来实现一个布尔函数. 即 SOP 和 POS.

SOP：第一层使用 AND gate 将 inputs 组合，生成各个乘积项；第二层使用 OR gate 对所有乘积项求和，得到最终逻辑输出.

POS：第一层使用 OR gate 将 inputs 组合，生成各个和项；第二层使用 AND gate 对所有和项求积，得到最终逻辑输出.

两级实现的优点是结构清晰、便于分析和简化；不过在处理较大规模逻辑函数时，可能会产生大量的门电路与连线，导致面积增大和延时问题. 因此，在实际设计中需要综合考虑简化逻辑和电路性能.

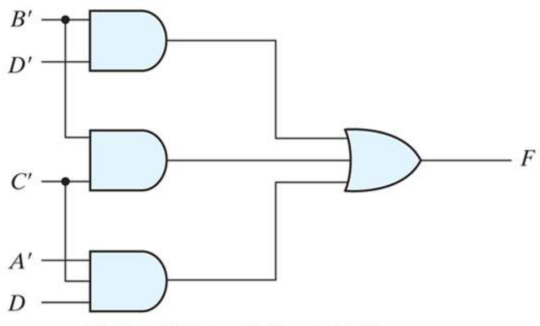
From the previous example, $F(A,B,C,D)=\sum m(0,1,2,5,8,9,10)$ can be simplified

- In SOP form by collecting the 1s as $F(A,B,C,D)=B'D'+B'C'+A'C'D$
- In POS form by collecting the 0s as $F(A,B,C,D)=(B'+D)(C'+D')(A'+B')$

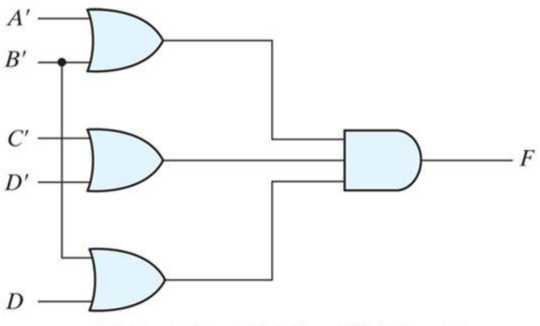
The implementation of a function in a standard form is said to be a two-level implementation.

(a) In SOP form, AND gates are connected to a single OR gate

(b) In POS form, OR gates are connected to a single AND gate



(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

Lec 4 组合逻辑电路

> Combinational Logic
>
> 与其相对的是时序逻辑电路.

4.1 数字电路分类

> Digital Circuit Categories
>
> 本章关注组合逻辑电路.

Combinational logic circuits

- The outputs at any instant of time depend upon the inputs present at that time instant
- No memory in these circuits

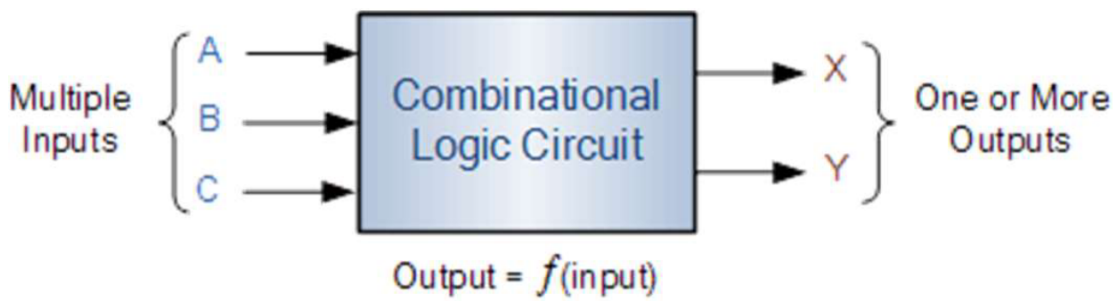
Sequential logic circuits

- The outputs at any instant of time depend upon the present inputs as well as the past inputs/outputs
- There are memory elements used to store past information

4.1.1 组合逻辑电路

> Combinational Logic Circuits

- The outputs at any instant of time depend upon the inputs present at that time instant
- No memory in these circuits



- A combinational circuit consists of input variables, logic gates, and output variables

- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal

Example：BCD 转 加三码

> Design a combinational logic circuit to convert Binary Code Decimal (BCD) to Excess-3 Code

Excess-3 Code（也称为**XS-3 Code**）是一种**自补码编码（self-complementary code）**，主要用于数字电子系统中表示十进制数字. 它在原始的 **BCD 编码（Binary Coded Decimal）**基础上加上了一个偏移量 3.

自补码性质：对某个十进制数的 Excess-3 编码按位取反，就可以得到该十进制数对 9 的补码的 Excess-3 编码.

下面说明为满足对 9 的自补码性质，基于 BCD 的偏移量只能为 3，而不能为 $1, 2$ 或其他值：

记 x 为原十进制数（$0 \sim 9$），y 为对应的 Excess-3 二进制数，y' 为按位取反后的 Excess-3 二进制数，x' 为 y' 对应的十进制数. 设偏移量为 δ.

$$\begin{cases}(x+\delta)_{10}=y_2\\(x'+\delta)_{10}=y'_2\end{cases}\Rightarrow(x+x'+2\delta)_{10}=(y+y')_2=1111_2=15_{10}$$

Let $(x+x')_{10}=9_{10}$, we have $9+2\delta=15\Rightarrow\delta=3$.

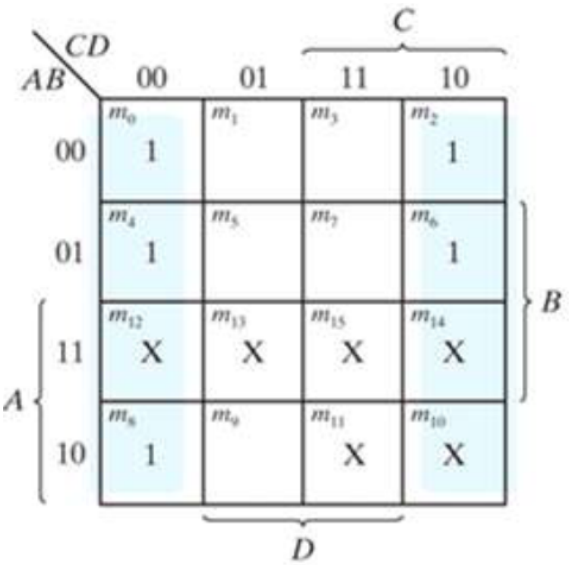
四输入四输出，列真值表：

Table 4.2
Truth Table for Code Conversion Example

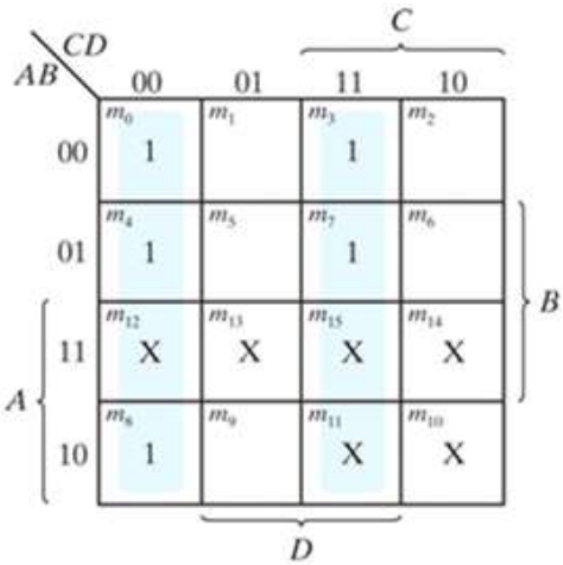
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

列卡诺图化简：

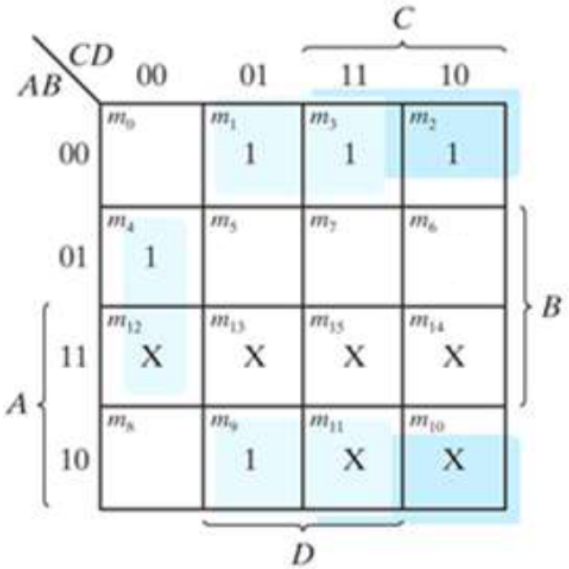
> 注意 Some of cells are X (i.e., "Don't Care") because there are only 10 digits (i.e., $0 \sim 9$) in BCD.



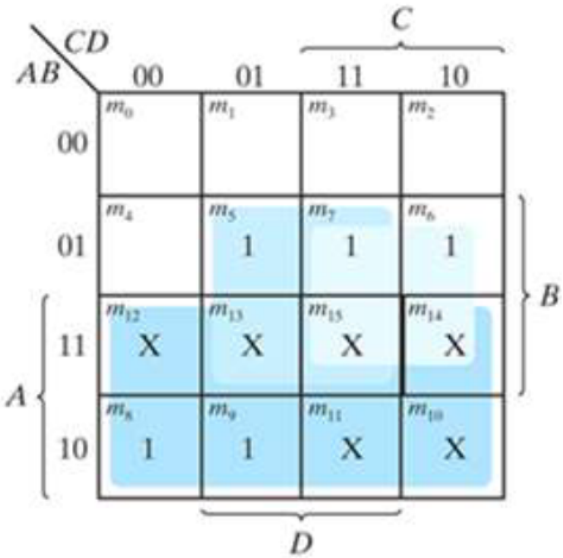
$z = D'$



$y = CD + C'D'$



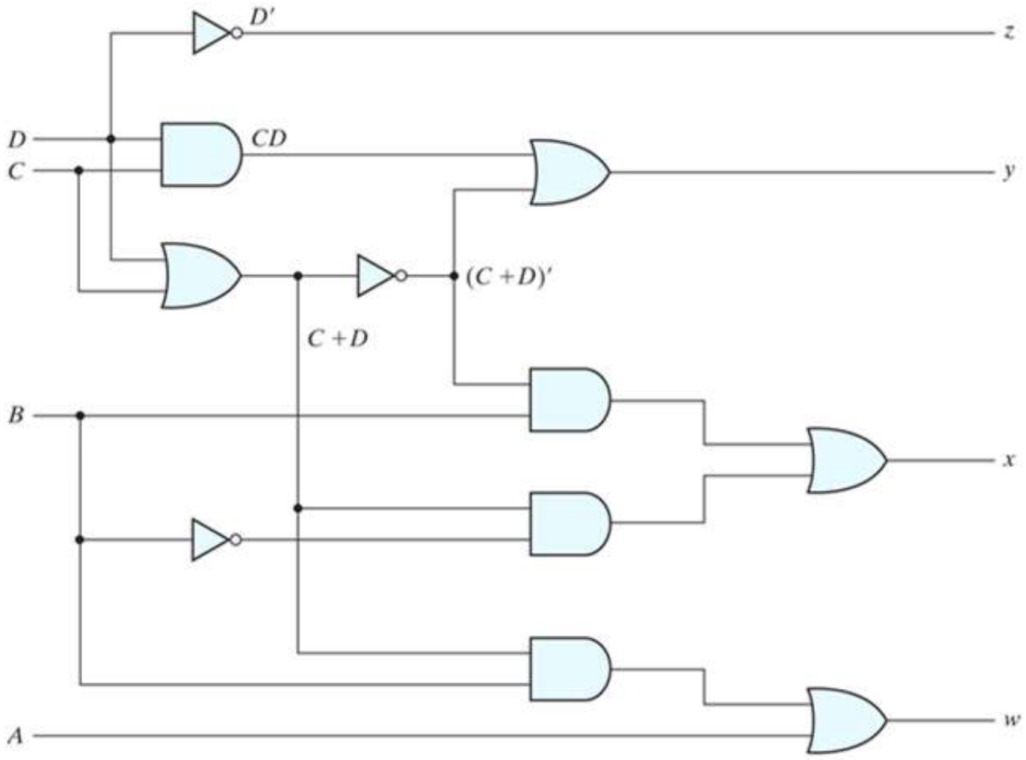
$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

根据化简后的表达式，建立组合逻辑电路：

> Use the simplified logic functions, the combinational logic circuit can be built as below:



4.1.2 时序逻辑电路

> Sequential Logic Circuits
>
> 本章不讨论，$4.2 \sim 4.6$ 均为组合逻辑电路.

- The outputs at any instant of time depend upon the present inputs **as well as the past inputs/outputs**
- There are memory elements used to store past information

4.2 加法器 & 减法器

> Binary Adder and Subtractor

4.2.1 半加器

> Half Adder (HA)

A half adder is a 1-bit adder which do **NOT** take carry-in into consideration

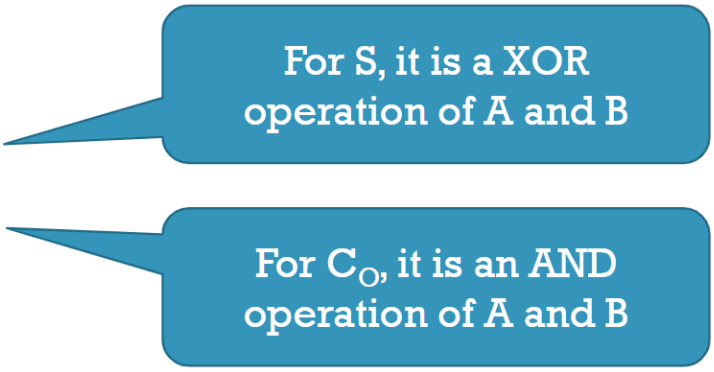
> 仅限于单位加法，不能处理进位输入（如果要处理来自前一位的进位，必须使用全加器）.
>
> 两个输入，两个输出.

The sum of two 1-bit binary numbers A and B, can be represented as $S = A + B$:

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=10$, where 1 is known as carry-out bit, C_o

the truth table of HA is shown below:

A	B	C_o	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



半加器由异或门（XOR）和与门（AND）组成.

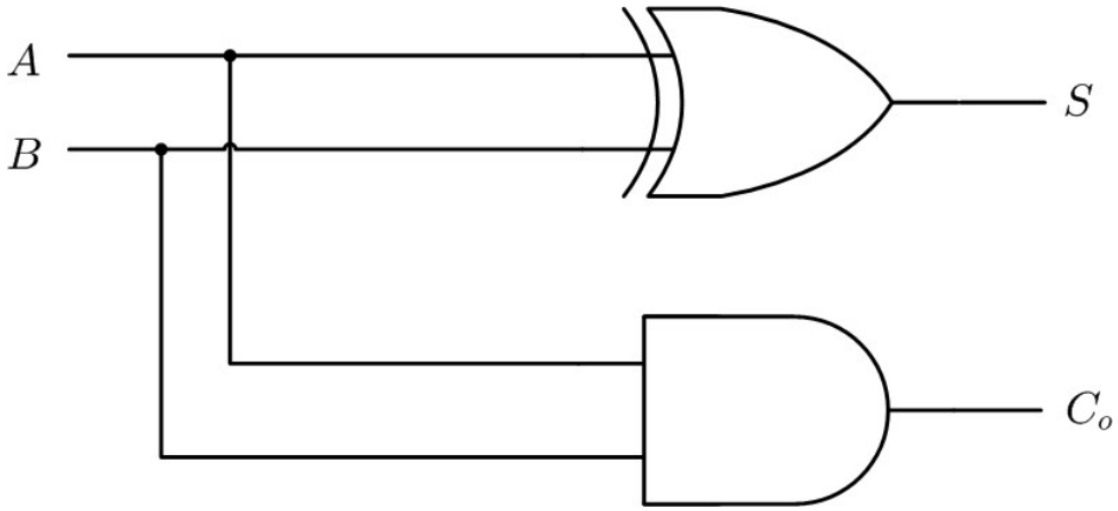
> XOR 计算 Sum，AND 门计算 Carry out.

The Boolean functions of HA:

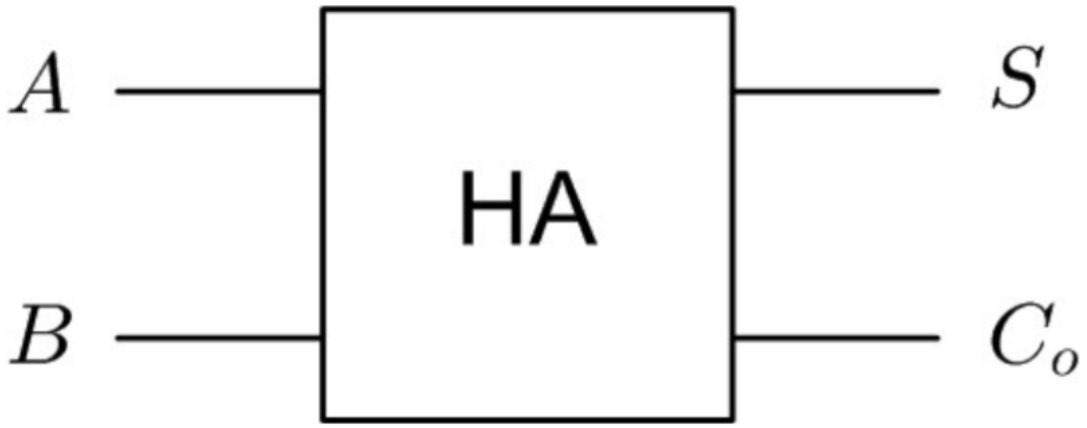
$$S=\overline{A}B+A\overline{B}=A\oplus B$$
$$C_o=AB$$

> C_o 的 C 是 "Carry"，o 是 out. 表示 A 和 B 相加时产生的进位.
>
> S 是 sum，表示 A 和 B 的和，不考虑进位.

The logic circuit of HA:



The block diagram of HA:



应用：半加器作为基本的加法单元，常用于

- 全加器的构建.
- 简单的数字电路，如计数器和基本的运算逻辑.
- 计算器和微处理器中的算数逻辑单元（ALU）.

4.2.2 全加器

> Full Adder (FA)

When the operations require more than 1-bit, HA is not good enough

- Carry-out bits from the lower significant bits are not considered/used at the higher bit operations

By extending the HA, one can implement a full adder (FA) which takes a carry-in bit as an extra input

> 可以处理进位输入. 三输入两输出.

- The carry-out bit of the lower order bit can be connected to the carry-in bit of the immediate higher order bit

> 适合级联扩展.

FA becomes the basic unit for a multiple-bit adder

A	B	C_{in}	C_o	S
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

Three inputs:

- Two 1-bit numbers, A and B
- One Carry-in, C_{in}

Two outputs:

- Sum, S
- Carry-out, C_o

Equations of FA:

$$S = \sum(1,2,4,7)$$
$$C_o = \sum(3,5,6,7)$$

> Sum of Product 的形式.

Simplify the Boolean functions of FA:

① $S = \sum(1,2,4,7)$

列卡诺图

	C_{in}' (0)	C_{in} (1)
A'B' (00)		1
A'B (01)	1	
AB (11)		1
AB' (10)	1	

无法合并，尝试用布尔表达式化简：

$$\begin{aligned} S &= A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in} \\ &= A \oplus B \oplus C_{in} \end{aligned}$$

> 见 `2.3.6 异或门 （2025.2.24）例` .

② $C_o = \sum(3,5,6,7)$

列卡诺图

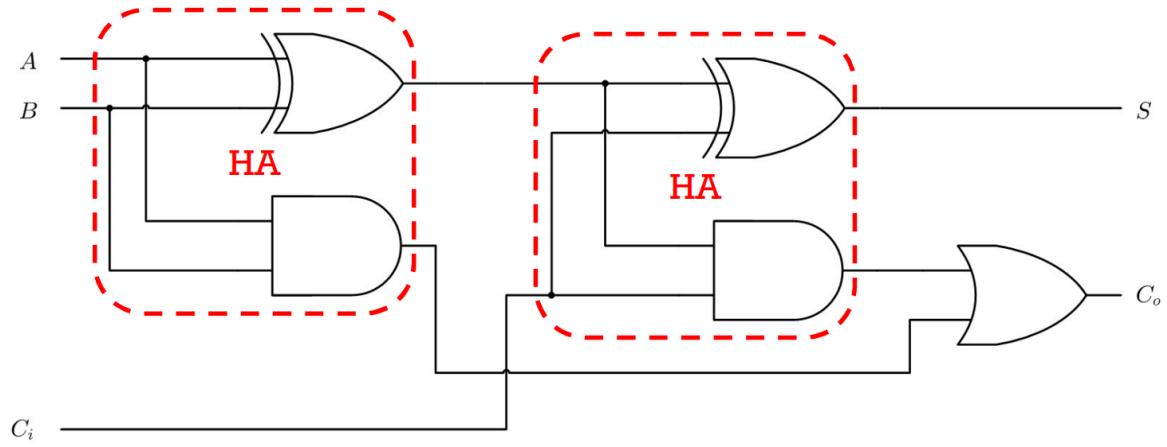
	C_{in}' (0)	C_{in} (1)
A'B' (00)		
A'B (01)		1
AB (11)	1	1
AB' (10)		1

合并化简：

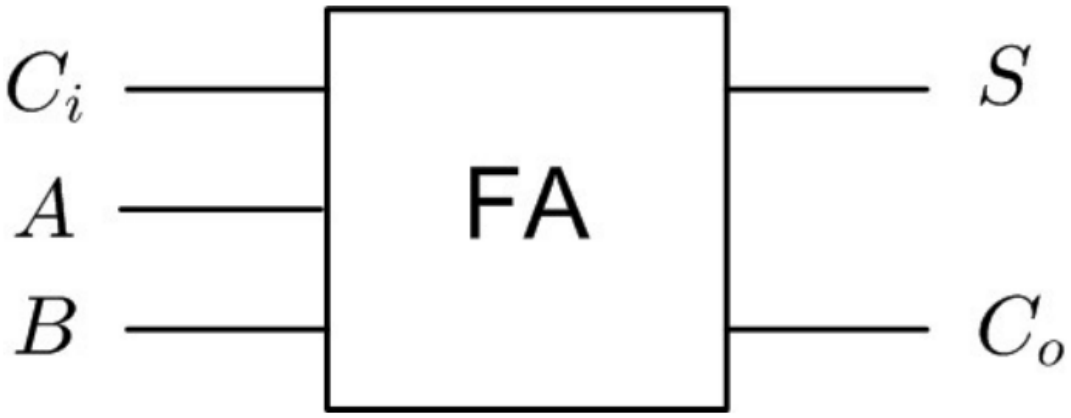
$$\begin{aligned} C_o &= AB + BC_{in} + AC_{in} \\ &= AB + (A \oplus B)C_{in} \\ &= C_1 + C_2 \end{aligned}$$

> 从卡诺图可以得到第二个等号的表示，即红色框合并为 AB，另外两个单元不进行合并，为 $(A \oplus B)C_{in}$. 而 $S = (A \oplus B) \oplus C_{in}$，结合半加器两个输出的形式，可以发现全加器可以拆成两个半加器. 其中，第一个半加器的输入为 A 和 B，第二个半加器的输入为 $A \oplus B$（第一个半加器的 S 输出）和 C_{in}. 第二个半加器的 S 输出作为最终的 S，对第一个半加器和第二个半加器的进位进行 OR 操作得到最终的进位 C_0.
>
> C_1 是第一个半加器的进位，C_2 是第二个半加器的进位.

The logic circuit of FA:



The block diagram of FA:



4.2.3 半减器

> Half Substractor (HS)

A half subtractor is a 1-bit subtractor which do not take borrow bit into consideration.

A	B	Di	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

For Di, it is a XOR operation of A and B

For B_O, it is an AND operation of A' and B

The difference between two 1-bit binary numbers A and B, can be represented as $D_i = A - B$:

$$
\begin{aligned}
&0 - 0 = 0 \\
&0 - 1 = \dot{0}1 \text{ or } \dot{1}1 \\
&1 - 0 = 1 \\
&1 - 1 = 0
\end{aligned}
$$

where we need to indicate a borrowing, as the output borrow bit, B_o. 当遇到输入是 $0-1$ 时（$A=0$ 且 $B=1$），需要向前借一位，记为 $B_o=1$. 其余情况不需要借位，$B_o=0$. 借到一位后，本位减法变为 $2-1=1$. 前一位被借的可能是 0 或 1，但这是后续的计算，不归本半减器处理.

半减器由异或门（XOR）和与门（AND）组成.

> XOR 计算 Difference，AND 门计算 Borrow out.

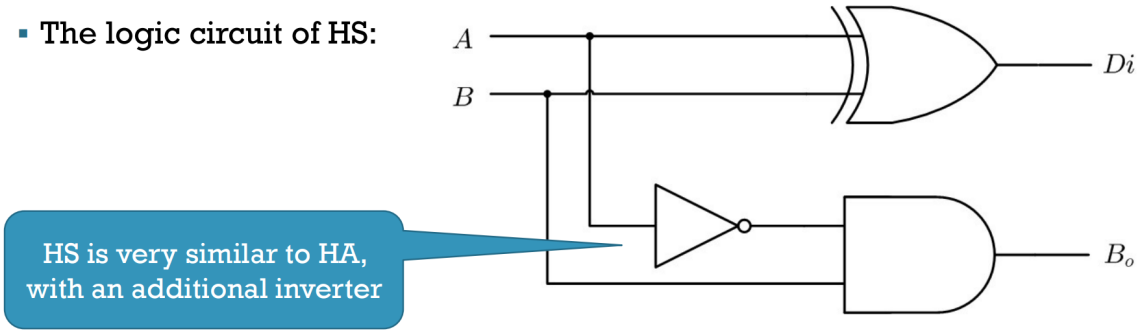
The Boolean functions of HS:

$$
\begin{aligned}
D_i &= \overline{A}B + A\overline{B} = A \oplus B \\
B_o &= \overline{A}B
\end{aligned}
$$

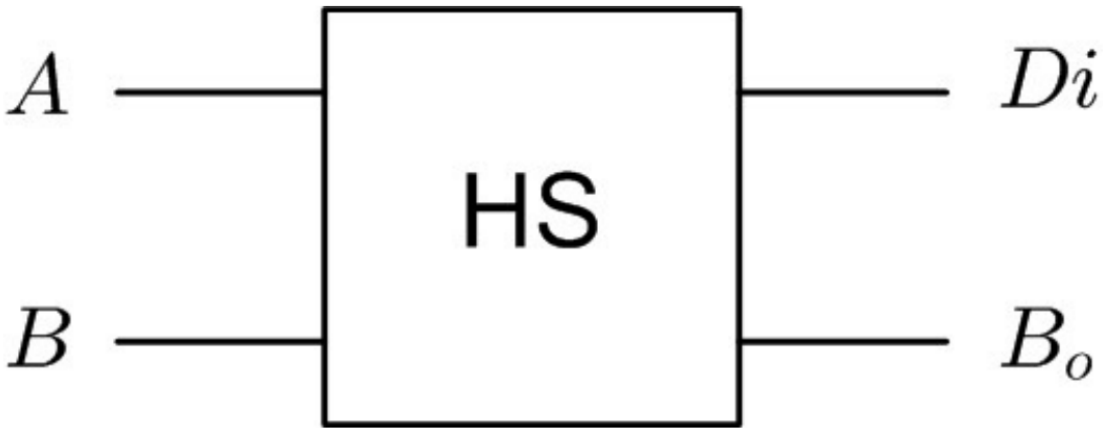
> B_o 的 B 是 "Borrow"，o 是 out. 表示 A 和 B 相减时产生的借位.

> D_i 是 Difference，表示 A 和 B 的差，不考虑借位.

The logic circuit of HS:



The block diagram of HS:



4.2.4 全减器

> Full Substractor (FS)

When the operations require more than 1 bit, HS is not useful.

- Borrow-out bits from the lower significant bits are not considered/used at the higher bit operations

By extending the HS, one can implement a full substractor (FS) which takes a **borrow-in** bit as an extra input

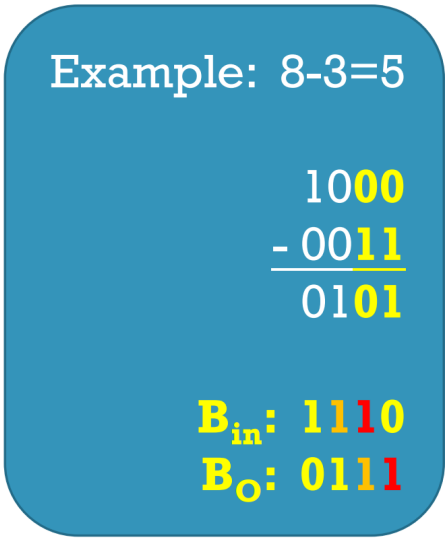
> 可以处理借位输入. 三输入两输出.

- The borrow-out bit of the lower order bit can be connected to the borrow-in bit of the immediate higher order bit

> 适合级联扩展.

FS becomes the basic unit for a multiple-bit subtractor.

$D_i = A - B - B_{in}$:



The truth table of FS:

A	B	B_{in}	Di	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Three inputs:

- Two 1-bit numbers, A and B
- One Borrow-in, B_{in}

Two outputs:

- Difference, D_i
- Borrow-out, B_o

Equations of FS:

$$D_i = \sum(1, 2, 4, 7)$$
$$B_o = \sum(1, 2, 3, 7)$$

> Sum of Product 的形式.

Simplify the Boolean functions of FS:

① $D_i = \sum(1, 2, 4, 7)$

列卡诺图

	B_{in}' (0)	B_{in} (1)
A'B' (00)		1
A'B (01)	1	
AB (11)		1
AB' (10)	1	

无法合并，尝试用布尔表达式化简：

$$\begin{aligned} D_i &= A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in} \\ &= A \oplus B \oplus B_{in} \end{aligned}$$

> 见 `2.3.6 异或门 （2025.2.24）例`.

② $B_o = \sum(1, 2, 3, 7)$

列卡诺图

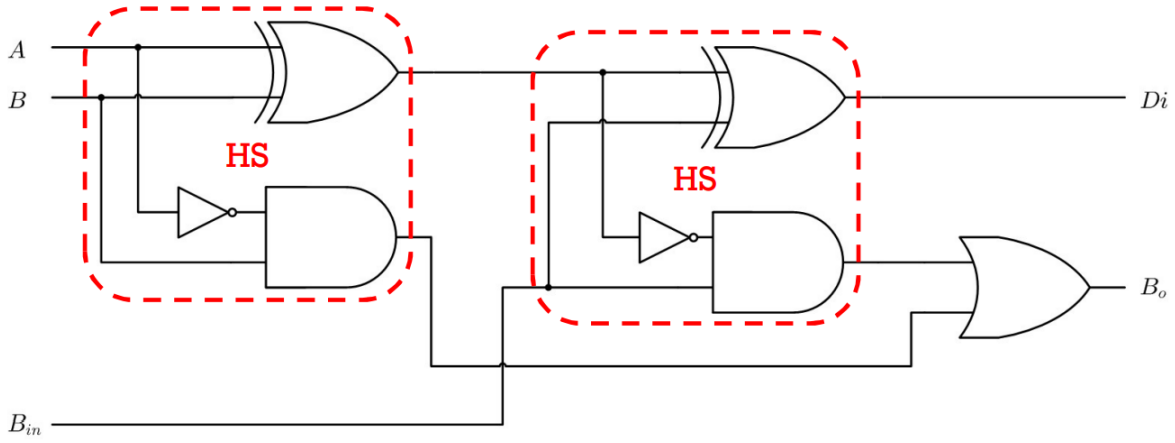
	B_{in}' (0)	B_{in} (1)
A'B' (00)		1
A'B (01)	1	1
AB (11)		1
AB' (10)		

合并化简：

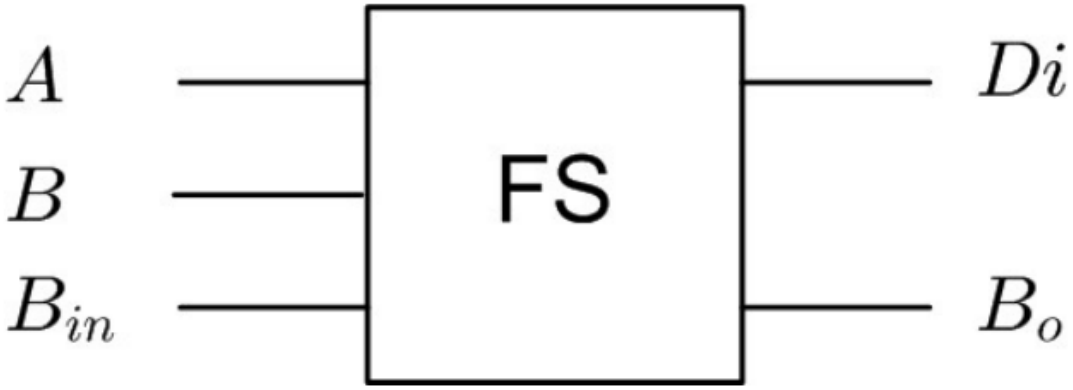
$$\begin{aligned} B_o &= A'B + BB_{in} + A'B_{in} \\ &= A'B + (A' \oplus B)B_{in} \\ &= A'B + (A \oplus B)'B_{in} \\ &= B_1 + B_2 \end{aligned}$$

> 从卡诺图可以得到第二个等号的表示，即红色框合并为 $A'B$，另外两个单元不进行合并，为 $(A' \oplus B)B_{in}$. 而 $D_i = (A \oplus B) \oplus B_{in}$，结合半减器两个输出的形式，可以发现全减器可以拆成两个半减器. 其中，第一个半减器的输入为 A 和 B，第二个半减器的输入为 $A \oplus B$（第一个半减器的 D_i 输出）和 B_{in}. 第二个半减器的 D_i 输出作为最终的 D_i，对第一个半减器和第二个半减器的借位进行 OR 操作得到最终的借位 B_o.
>
> B_1 是第一个半减器的借位，B_2 是第二个半减器的借位.

The logic circuit of FS:



The block diagram of FS:



Example: $A - B = 1110101_2 - 0011100_2 = 117_{10} - 28_{10} = 89_{10} = 1011001_2$

	A	1	1	1	0	1	0	1
-	B	- 0	0	1	1	1	0	0
	Di	1	0	1	1	0	0	1
	B_{in}	0	1	1	0	0	0	
		B_{in}	B_{in}	B_{in}	B_{in}	B_{in}	B_{in}	
		B_o	B_o	B_o	B_o	B_o	B_o	B_o
	B_o	0	0	1	1	0	0	0

4.2.5 并行加法器 & 减法器

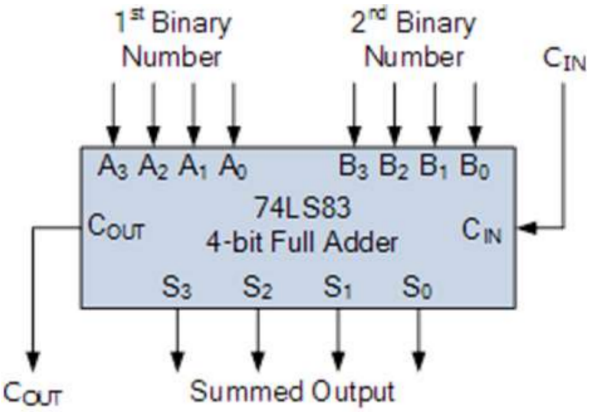
> Parallel Adder and Subtractor
>
> 与并行（Parallel）相对的概念：串行（Serial）

- Binary additions or substractions could be implemented in either serial or parallel fashion
- Serial addition takes longer time to process as at any given time only a pair of bits are processed
- Parallel addition is much faster as all bits are processed nearly at the same time
- We can use FA and FS as the basic building blocks

① 4 位并行加法器

> 4-bit Full Adder

Consider a 4-bit FA to process two 4-bit binary numbers: $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.



Input: 4 bits for A, 4 bits for B, 1 bit for Carry-in (C_{in}).

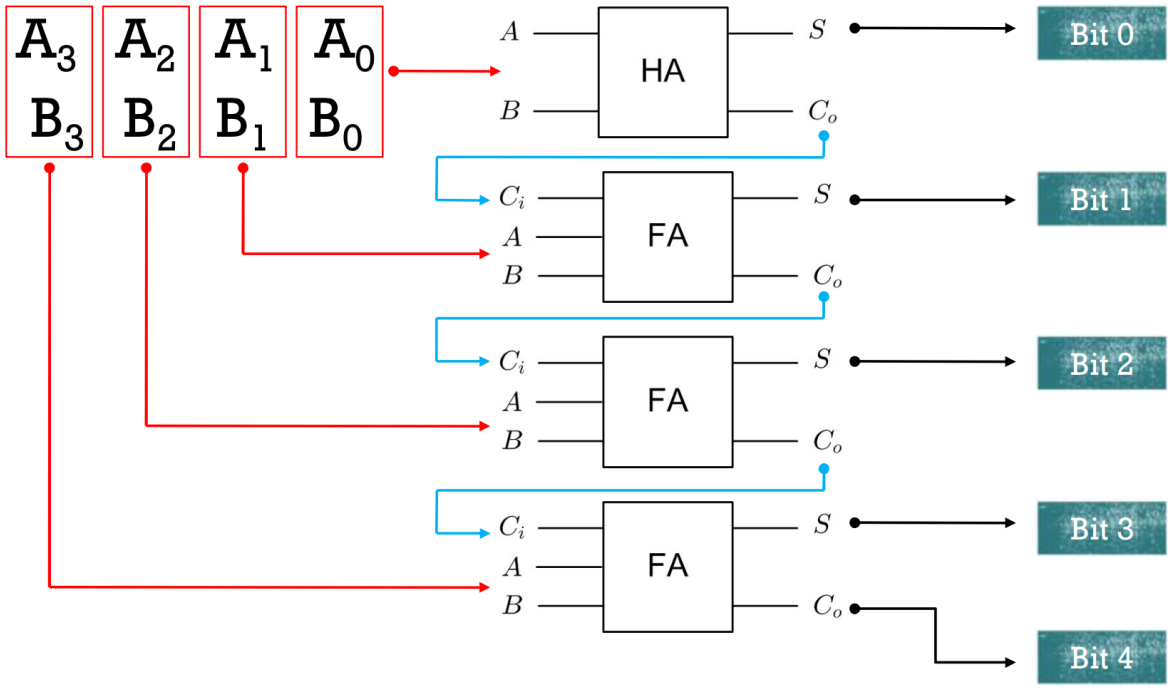
Output: 4 bits for Sum (S), 1 bit for Carry-out (C_{out}).

To build a 4-bit FA, 1x HA and 3x FAs, or 4x FAs are required

> 两种实现方法：一个半加器三个全加器（无进位），或四个全加器（有进位）.

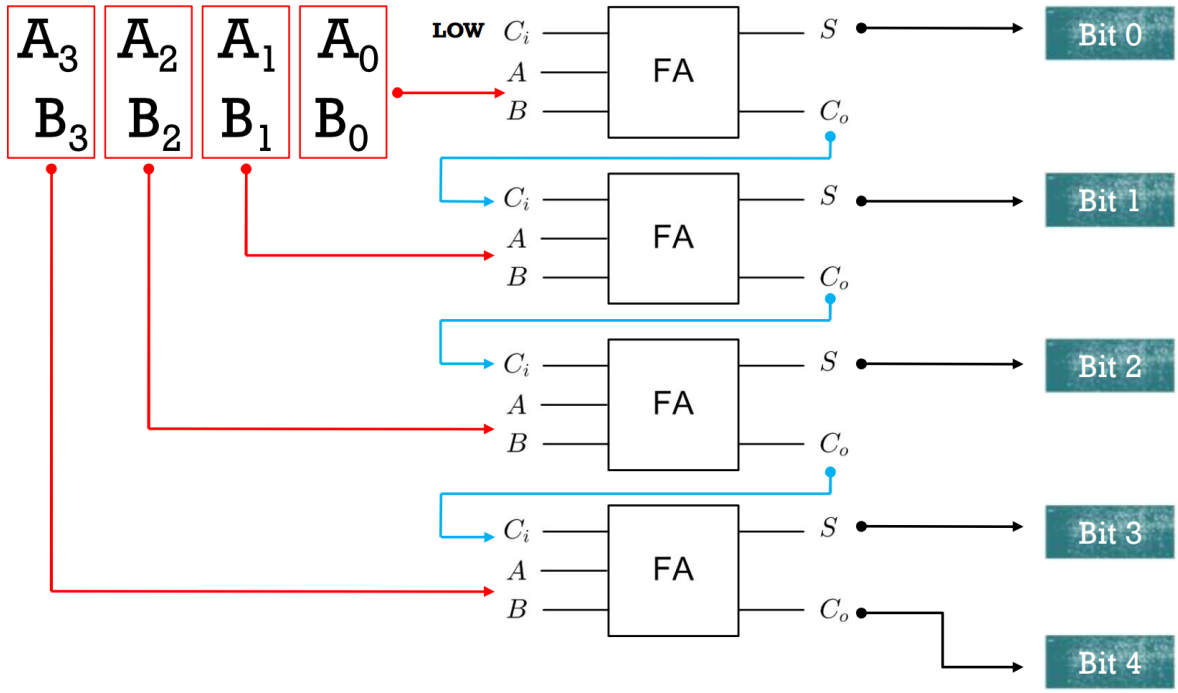
4-bit Parallel Adder - Hybrid

> 混合实现. 这个电路假设最初的进位为 0. 通常用于末端. 例如，使用 4 个 4-bit Parrallel Adder 组成一个 16 位加法器，负责最低 4 位的那个可以使用混合实现.

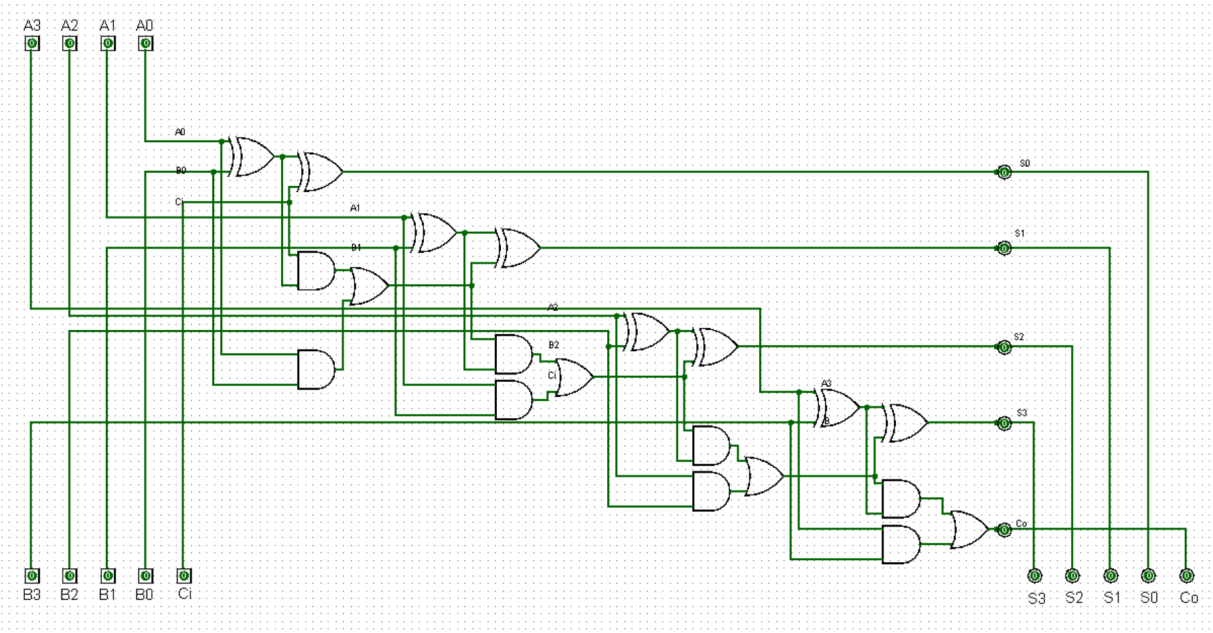


4-bit Parallel Adder - Standard

> 用于非末端 4-bit 加法.

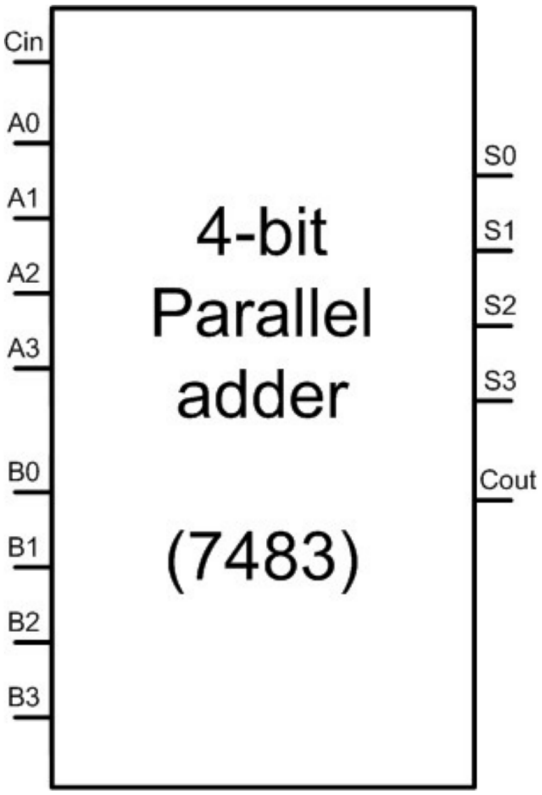


Logisim 实现如下:



Commercial Package of 4-bit Adder

A commercial IC 7483 is a 4-bit parallel adder that implements a FA.



A total of 9 input pins

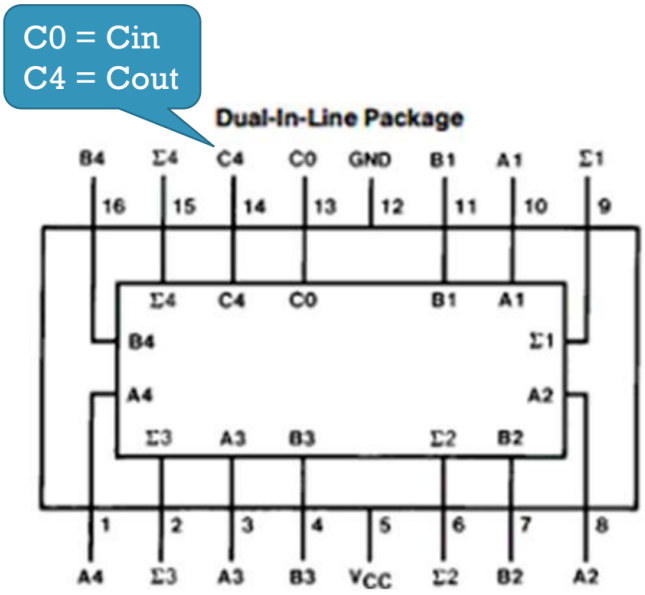
- 4 bits per binary number
- 1 bit for carry-in

A total of 5 output pins

- 4 bits for the sum

- 1 bit for carry-out

Pin Assignment of 7483



② 4 位并行减法器

> 4-bit Parallel Substractor

Consider a 4-bit FS to process two 4-bit binary numbers: $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

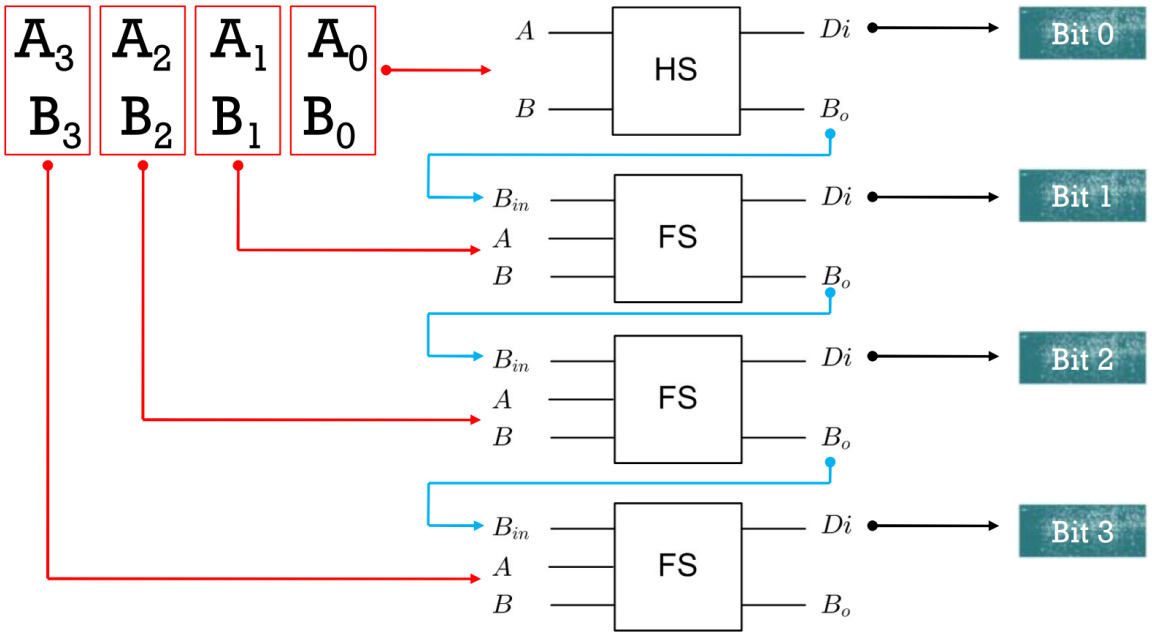
Input: 4 bits for A, 4 bits for B, 1 bit for Borrow-in (B_{in}).

Output: 4 bits for Difference (D_i), 1 bit for Borrow-out (B_{out}).

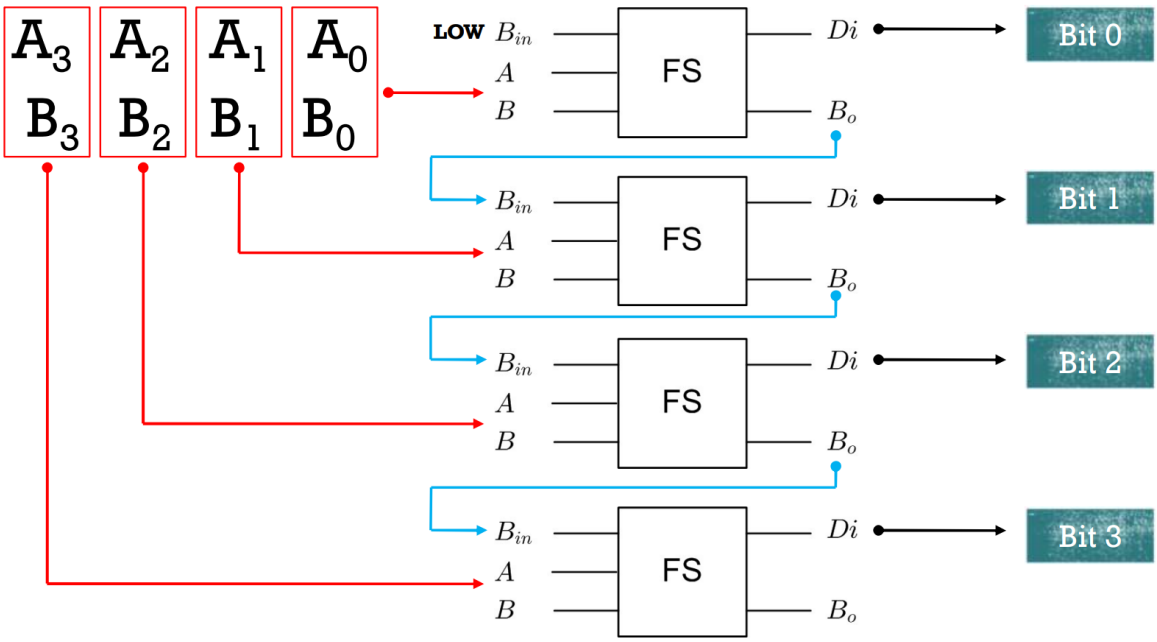
To build a 4-bit FS, 1x HS and 3x FSs, or 4x FSs are required

> 两种实现方法：一个半减器三个全减器（无借位），或四个全减器（有借位）.

4-bit Parallel Substractor - Hybrid



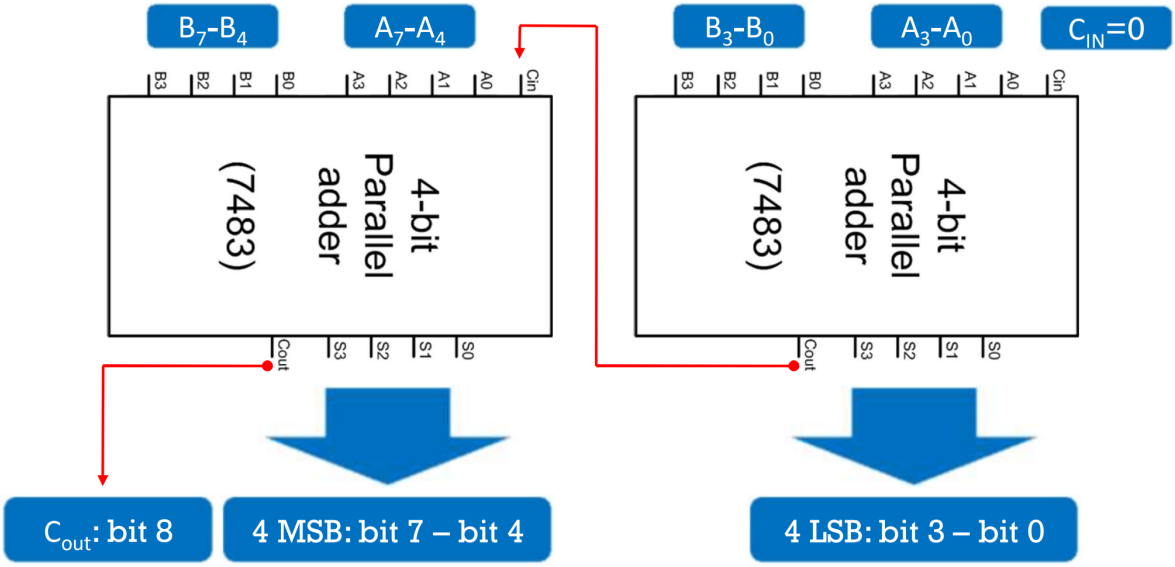
4-bit Parallel Substractor - Standard



③ 8 位并行加法器

> 8-bit Parallel Adder Using 7483

Let's connect the following 2x 7483 to form an 8-bit parallel adder



$4n$ 位二进制加法可以拆成 n 组 4 位二进制加法，然后用 n 个 4-bit 全加器处理. 对于两个相邻的全加器，处理低 4 位的全加器的 carry-out 接处理高 4 位的全加器的 carry-in，实现级联扩展. 对于处理最小 4 位的全加器，可以使用混合实现，或者使用标准实现并把 carry-in 手动置零.

④ 8 位并行减法器

同理，$4n$ 位二进制减法可以拆成 n 组 4 位二进制减法，然后用 n 个 4-bit 全减器处理. 对于两个相邻的全减器，处理低 4 位的全减器的 borrow-out 接处理高 4 位的全减器的 borrow-in，实现级联扩展. 对于处理最小 4 位的全减器，可以使用混合实现，或者使用标准实现并把 borrow-in 手动置零.

⑤ 加法器实现减法器

> Parallel Subtractor Using 2's complement

Binary additions can be used for subtractions.

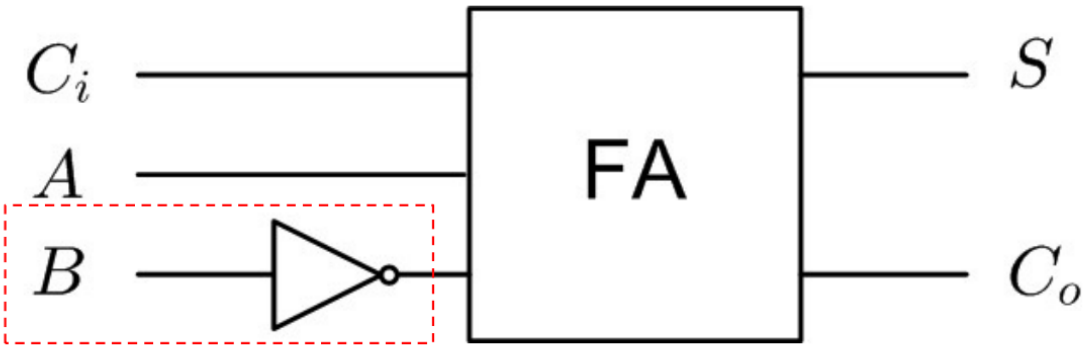
2's complement is generated by

- Flipping the bit value from 1 to 0 or 0 to 1.
- Followed by an addition of 1.

> 见 `1.3.3 有符号二进制系统 ◎ 2's 补码` .

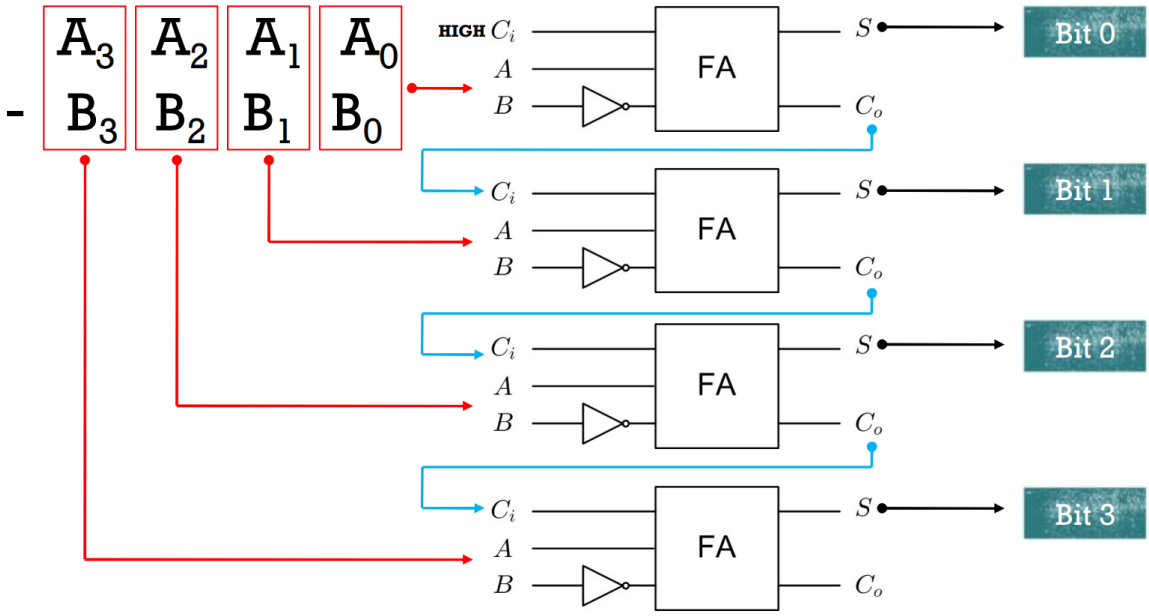
In other words, all multi-bit substractor can be implemented using the commercial package parallel adder, such as 7483.

1-Bit Subtractor Using FA & 2's complement



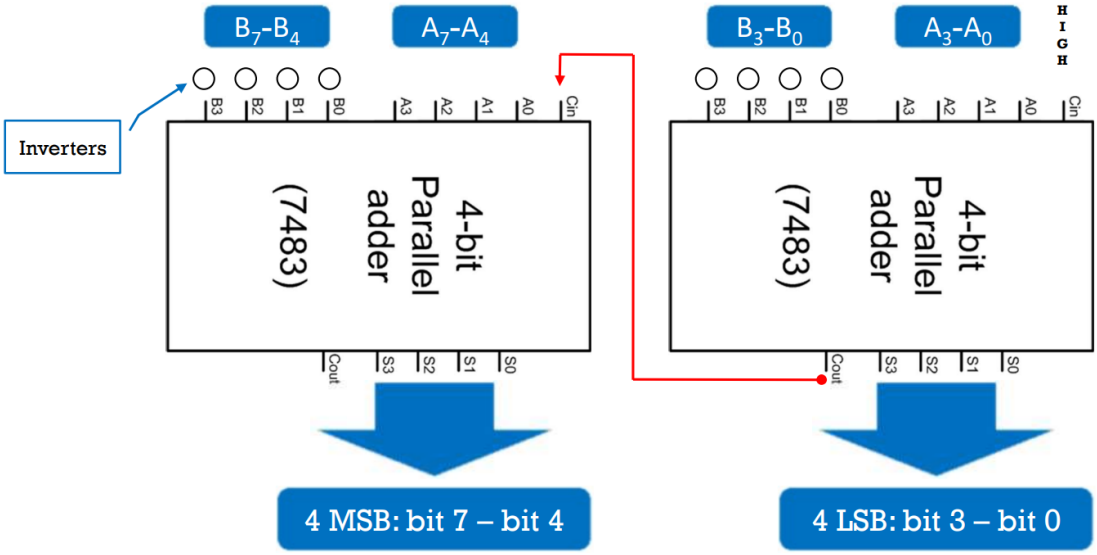
> 若使用 2's complement：$C_i = 1$，恒为高电位.
>
> 若使用 1's complement：不考.

4-Bit Subtractor Using FA & 2's complement



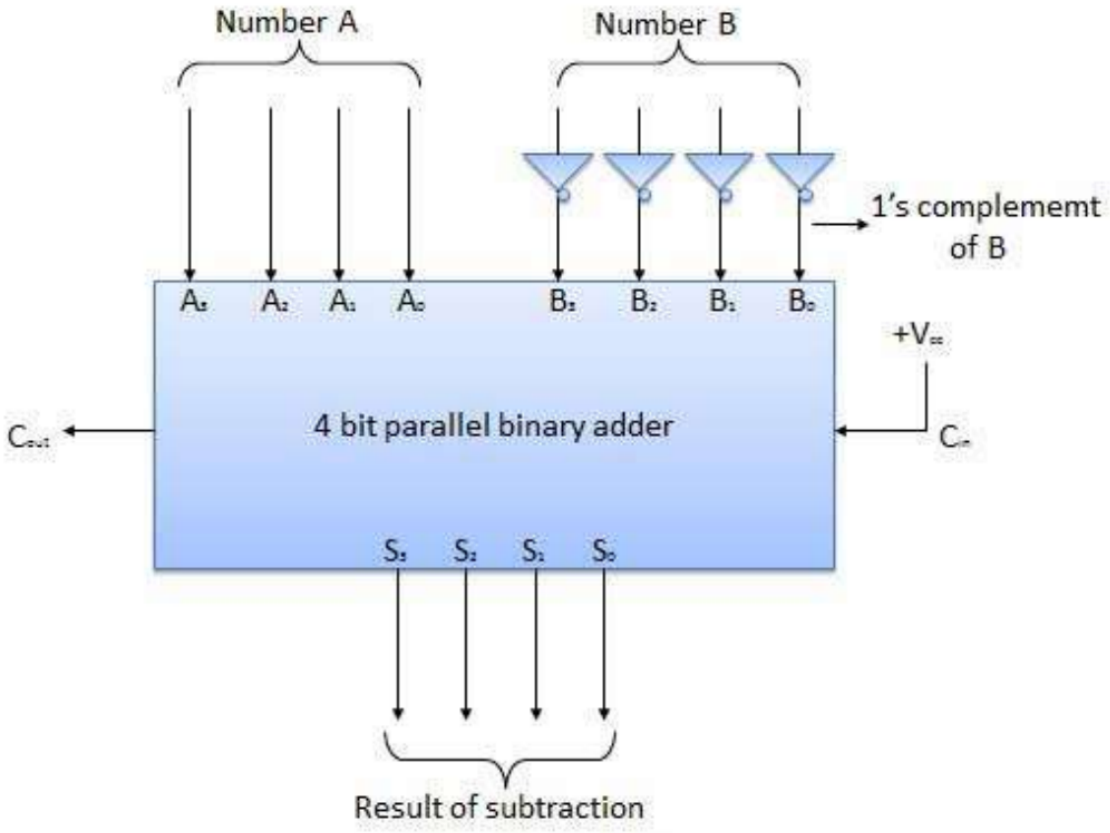
> 注意，2's complement 对 B 反向再加一，只需要把 B 的所有位取反，然后把最小位加法器的 C_i 置 1 即可（因为整个数字只加了 1）.

8-bit Subtractor Using 7483



> 级联方式和普通加法器类似. 注意 B 的所有位都要取反，最小位加法器的 C_i 置 1.

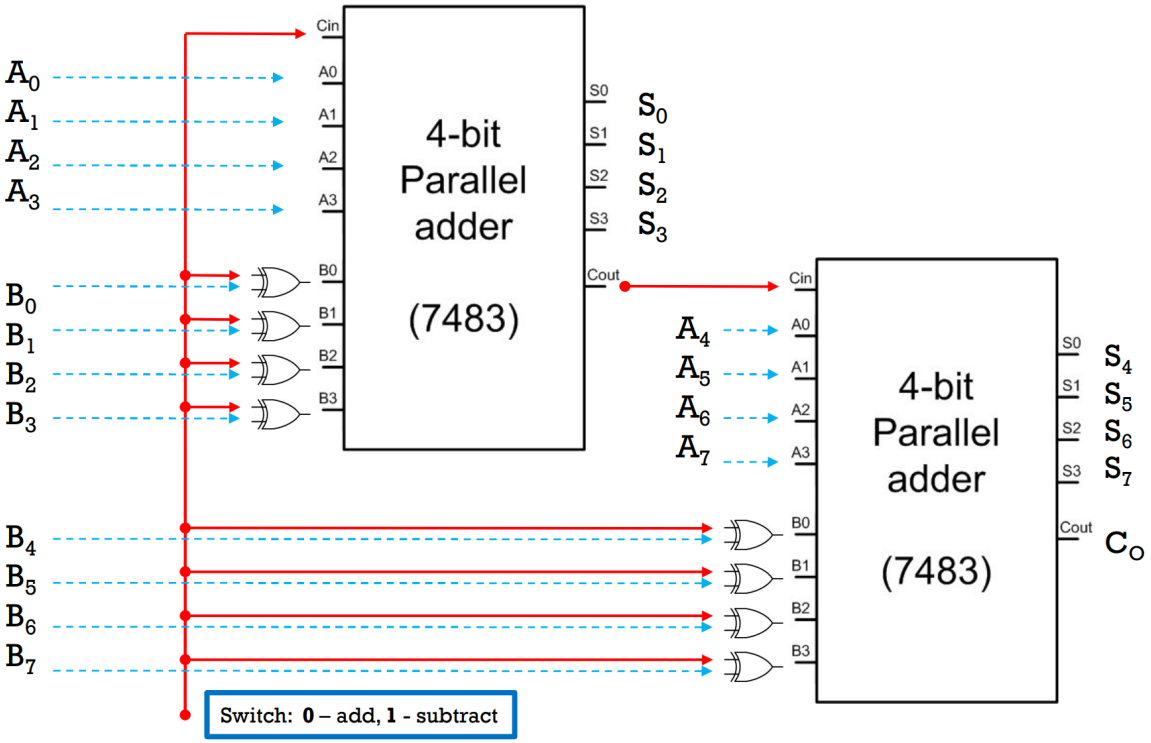
4-Bit Subtractor Using FA & 1's complement



> 若使用 1's complement：不考.

⑥ 通用加法器 & 减法器

> Universal 8-bit Adder / Substractor



设计一个滑动开关，开关为 0 时，执行 $A+B$；开关为 1 时，执行 $A-B$.

> 注意这个异或门的设计. 开关为 0 时，异或门输出 B；开关为 1 时，异或门输出 $\overline{B}$，相当于把 B 反向.

4.3 比较器

> Comparator

比较器（Comparator）是数字电路中一种重要的逻辑元件，用于比较两个二进制数的大小关系. 它的主要功能是判断两个输入数据是否相等，或者比较它们的大小关系，并输出相应的结果.

4.3.1 4 位比较器

> 4-bit Comparator

Consider a 4-bit comparator to compare the magnitute of two 4-bit binary numbers, A and B

After comparison, only one of the following three outputs will be HIGH

- D_1 represents $A=B$
- D_2 represents $A>B$
- D_3 represents $A<B$

Assume $A=A_3A_2A_1A_0$ and $B=B_3B_2B_1B_0$

If $A = B$, D_1 is HIGH if and only if $A_3 = B_3, A_2 = B_2, A_1 = B_1, A_0 = B_0$

In Boolean function, if $A = B$, $x_i = 1$ for $i = 0, 1, 2, 3$

$$x_i = A_i B_i + \overline{A}_i \overline{B}_i$$

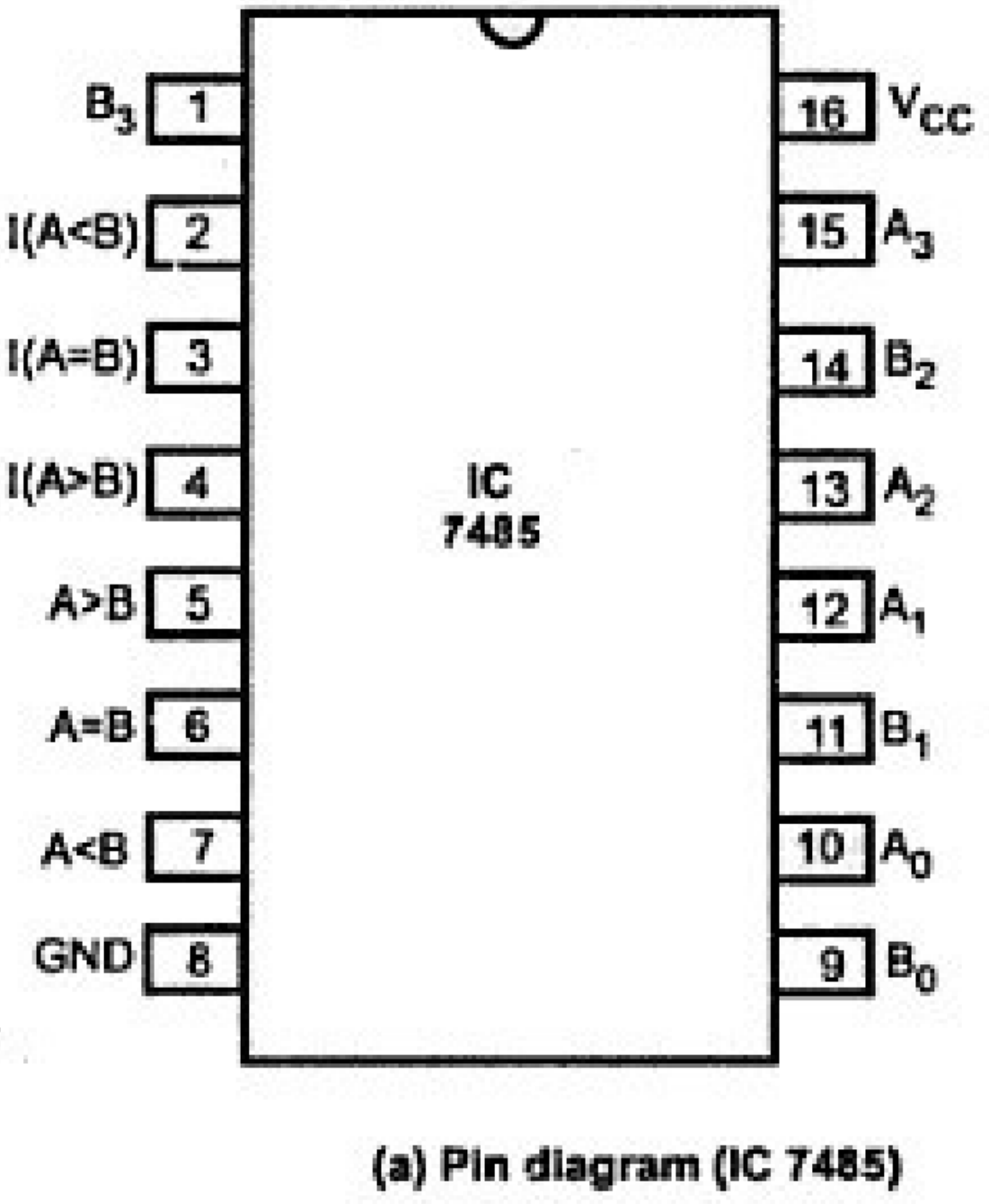
$D_1 : A = B \Rightarrow x_3 x_2 x_1 x_0$

$D_2 : A > B \Rightarrow A_3 \overline{B}_3 + x_3 A_2 \overline{B}_2 + x_3 x_2 A_1 \overline{B}_1 + x_3 x_2 x_1 A_0 \overline{B}_0$

$D_3 : A < B \Rightarrow \overline{A}_3 B_3 + x_3 \overline{A}_2 B_2 + x_3 x_2 \overline{A}_1 B_1 + x_3 x_2 x_1 \overline{A}_0 B_0$

Commercial Package of Comparator

A commercial IC 7485 is a 4-bit comparator



(a) Pin diagram (IC 7485)

> 高电平有效，例如 7 号引脚，当 $A < B$ 时为 1.
>
> $A < B$：输出位.
>
> V_{CC}：接 $+5V$ 电源.
>
> GND：接地 $0V$.
>
> I：级联输入，输入给负责更高四位的比较器.

注意，比较逻辑是从最高位开始比，但硬件连接 / 信号传递逻辑是从 LSB 往 MSB 传. 最后只需要看负责最高四位的比较器的输出位哪个为 1. 每个节点的输出位和级联输入位都是相同的，但中间节点的输出位相当于本节点及更小位的比较结果，对于整个数据的比较没有意义.

To compare two 1-byte binary numbers, two 7485 can be cascaded with each other.

4.4 译码器 & 编码器

> Decoder and Encoder

Encoder and Decoder are combinational logic circuits, they are working in pair.

They are used to convert information to codes (encoding) or codes to information (decoding).

Applications: Encryption, decryption, efficient transmission, and error detection and correction

4.4.1 译码器

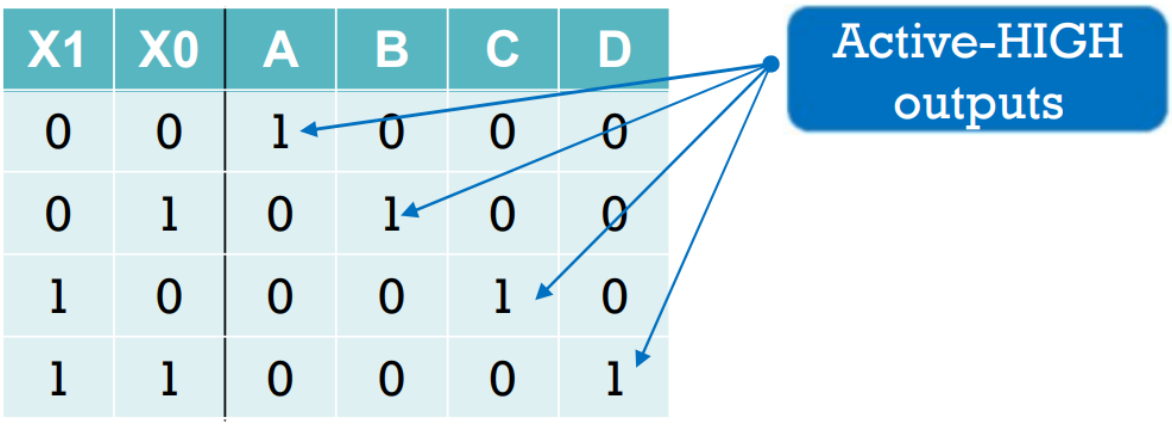
> Decoder

n 位输入，最多 2^n 路互斥输出，每个输入只对应一条有效.

用于地址译码，数码管驱动等.

① 2×4 译码器

> 2-Bit Decoder (Active-High)



In order to make an efficient hardware implementations, we have modified the truth table as below, by adding the complements of $X0$ and $X1$

X1	X0	X1’	X0’	A	B	C	D
0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0
1	0	0	1	0	0	1	0
1	1	0	0	0	0	0	1

Active-HIGH outputs

By making such modification, only 4 AND gates and 2 NOT gates are required.

其实就是让你更好列最小项，结果是一样的，例如 $A = X1'X0'$.

- Active-HIGH

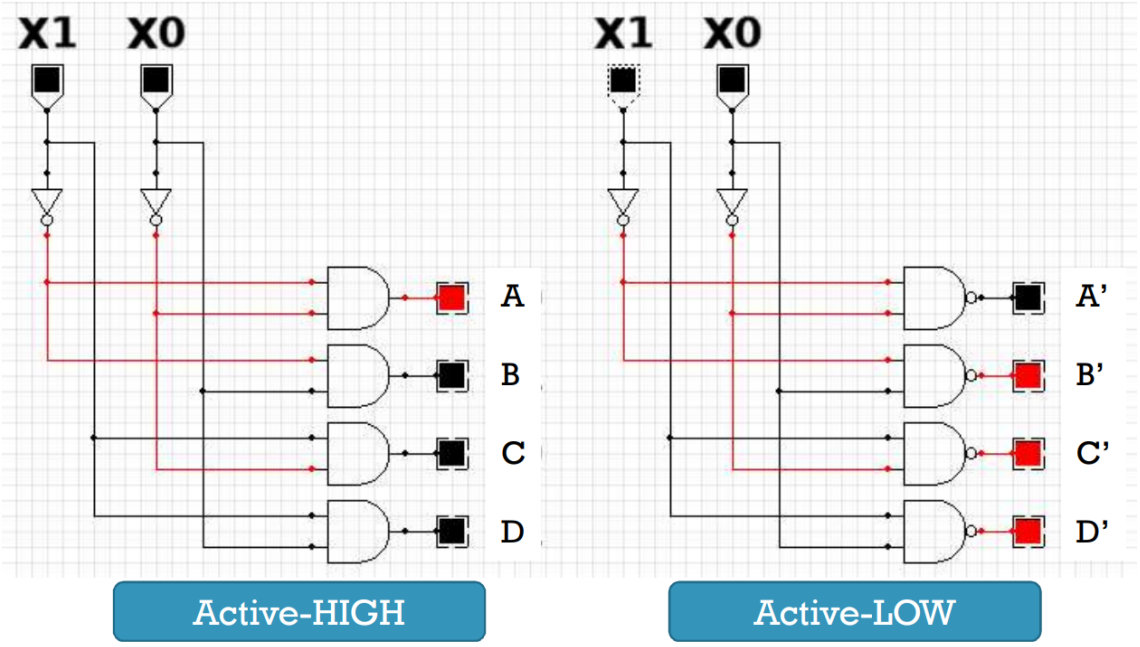
X1	X0	X1’	X0’	A	B	C	D
0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0
1	0	0	1	0	0	1	0
1	1	0	0	0	0	0	1

- Active-LOW

X1	X0	X1’	X0’	A’	B’	C’	D’
0	0	1	1	0	1	1	1
0	1	1	0	1	0	1	1
1	0	0	1	1	1	0	1
1	1	0	0	1	1	1	0

> Active-LOW：输出为 0 表示被选中，常见于 TTL 器件. TTL 家族低电平驱动能力强.

2×4 译码器电路如下：



② 3×8 译码器

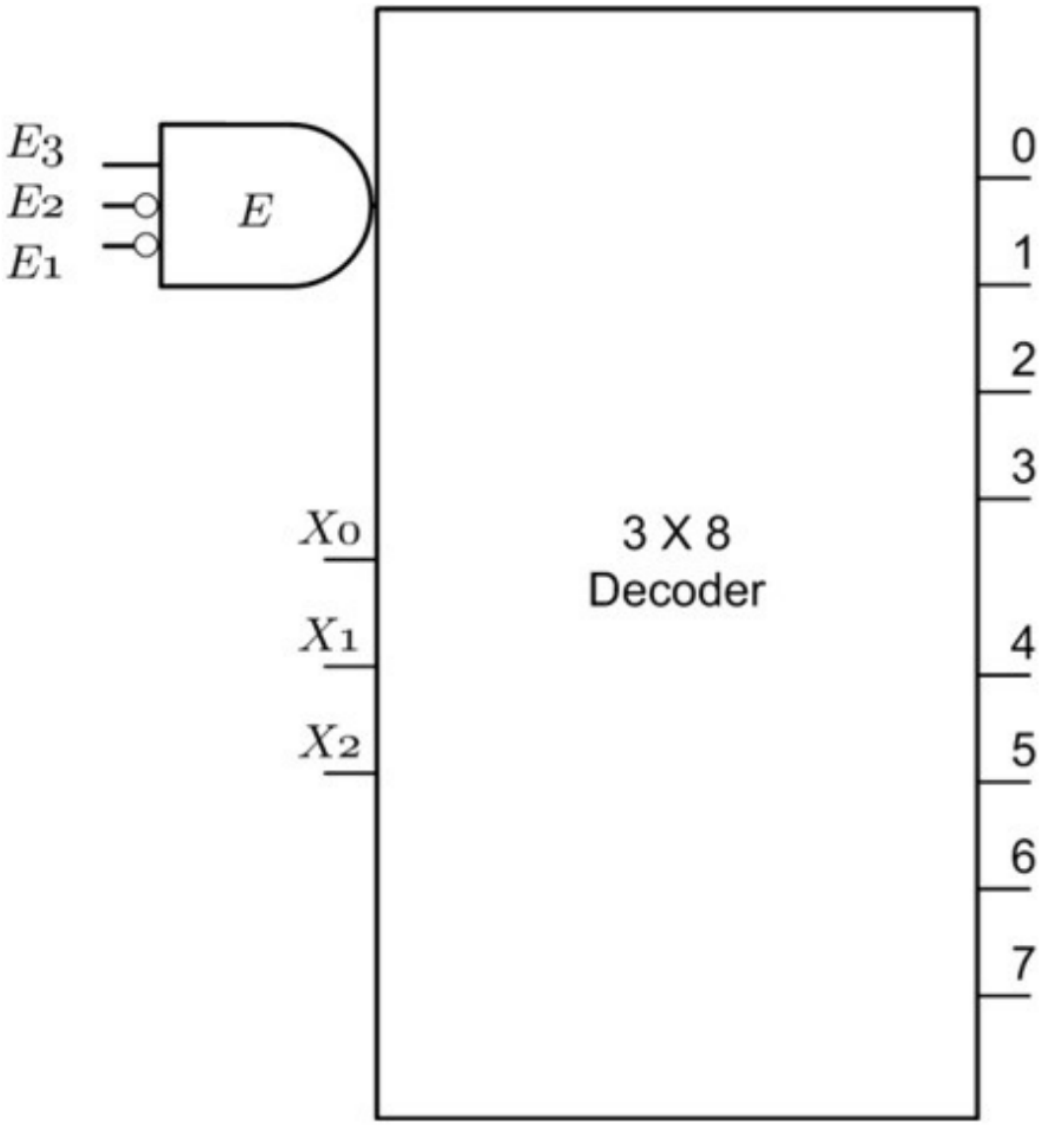
> 3-Bit Decoder

原理相同，只是多了一位：

X2	X1	X0	X2'	X1'	X0'	A	B	C	D	E	F	G	H
0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0	0	0	0	0
0	1	0	1	0	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	1	0	0	0	0	1	0	0	0
1	0	1	0	1	0	0	0	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	0	0	0	1

Commercial Package of 3×8 Decoder

74138 is a 3×8 decoder.



- 3-bit code input
- 8-bit Active-LOW output

In addition, there are 3 enabling bits which is used to provide an external control to the device

- Chip selection
- Reset

真值表:

* = Don't Care

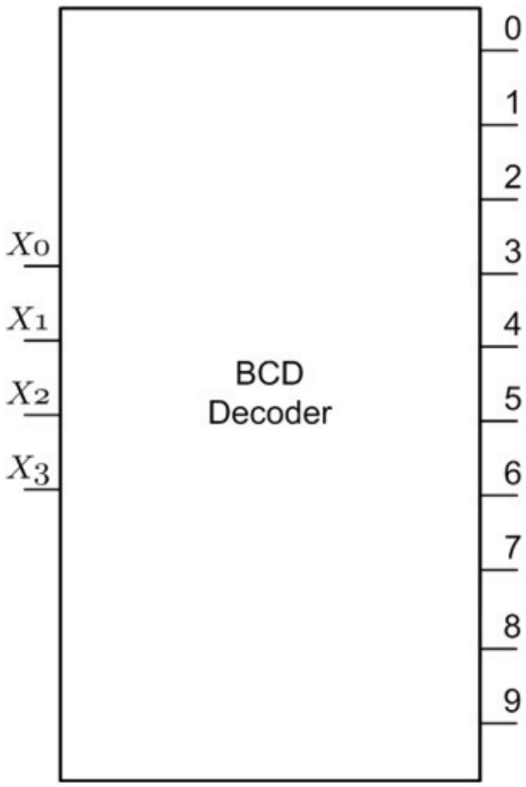
Disabled

Enabled

E1	E2	E3	X0	X1	X2	0	1	2	3	4	5	6	7
H	*	*	*	*	*	H	H	H	H	H	H	H	H
*	H	*	*	*	*	H	H	H	H	H	H	H	H
*	*	L	*	*	*	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

③ BCD-Decimal 译码器

Commercial Package of BCD Decoder



7442 is a BCD to DEC decoder

- 4-bit code input
- 10-bit Active-LOW output

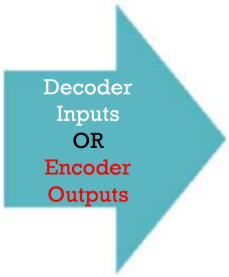
真值表:

X3	X2	X1	X0	0	1	2	3	4	5	6	7	8	9
L	L	L	L	**L**	H	H	H	H	H	H	H	H	H
L	L	L	H	H	**L**	H	H	H	H	H	H	H	H
L	L	H	L	H	H	**L**	H	H	H	H	H	H	H
L	L	H	H	H	H	H	**L**	H	H	H	H	H	H
L	H	L	L	H	H	H	H	**L**	H	H	H	H	H
L	H	L	H	H	H	H	H	H	**L**	H	H	H	H
L	H	H	L	H	H	H	H	H	H	**L**	H	H	H
L	H	H	H	H	H	H	H	H	H	H	**L**	H	H
H	L	L	L	H	H	H	H	H	H	H	H	**L**	H
H	L	L	H	H	H	H	H	H	H	H	H	H	**L**
H	L	H	L	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	H	L	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H

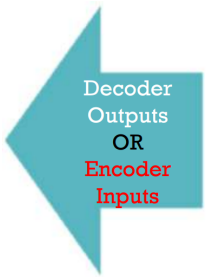
4.4.2 编码器

> Encoder

3×8 译码表翻转即可得 8×3 Encoder.

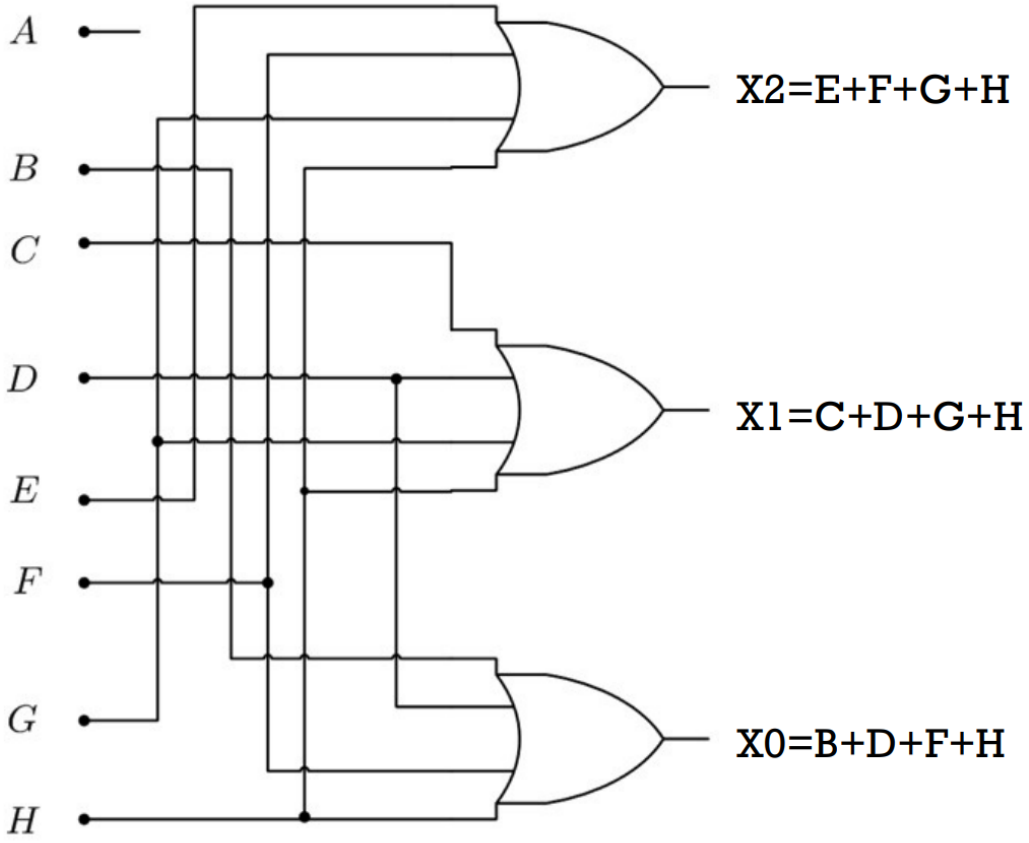


X2	X1	X0	A	B	C	D	E	F	G	H
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



译码器输出经 OR 门汇总即可恢复输入位.

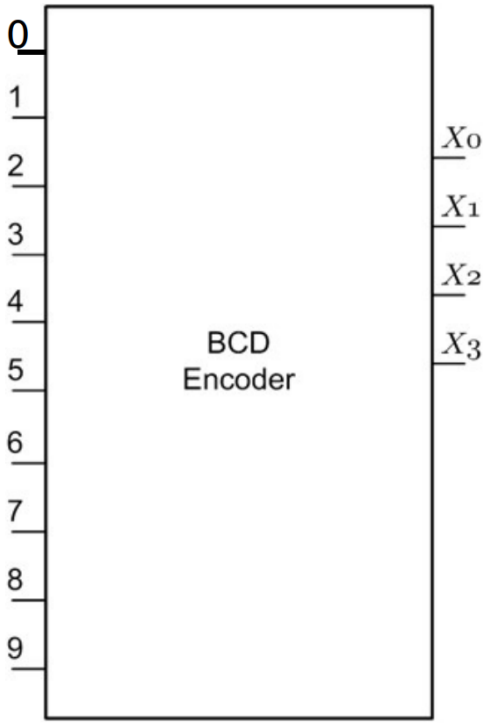
例：8×3 Encoder 的电路



Commercial Package of Encoder

74147 is a commercial package of a DEC to BCD encoder

- 10 decimal inputs from 0 to 9 (Active-LOW)
- 4 bits BCD output (Active-LOW)



74147 是 Priority Encoder

- When there are more than one inputs active, higher priority will be given to the larger decimal input.

真值表：

1	2	3	4	5	6	7	8	9	X3	X2	X1	X0
H	H	H	H	H	H	H	H	H	H	H	H	H
*	*	*	*	*	*	*	*	**L**	L	H	H	L
*	*	*	*	*	*	*	**L**	H	L	H	H	H
*	*	*	*	*	*	**L**	H	H	H	L	L	L
*	*	*	*	*	**L**	H	H	H	H	L	L	H
*	*	*	*	**L**	H	H	H	H	H	L	H	L
*	*	*	**L**	H	H	H	H	H	H	L	H	H
*	*	**L**	H	H	H	H	H	H	H	H	L	L
*	**L**	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

4.5 多路复用器 & 多路分配器

> Multiplexer (MUX) & Demultiplexer (DEMUX)
>
> 多路复用器也被称为「数据选择器」
>
> 多路分配器也被称为「数据分配器」

Multiplexing means transmitting a large number of information over a smaller number of channels or lines

- Data selection is involved by a control signal
- Implemented in the transmitting end

多路复用：$N \rightarrow 1$，把多路信号通过较少线路发送.

Demultiplexing means distributing a single source of information from a large number of channels or lines

- Work like a decoder
- Implemented in the receiving end

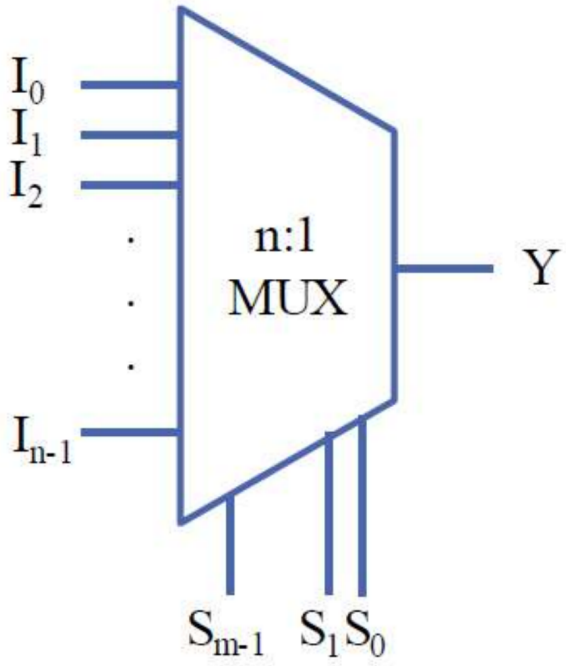
解多路：$1 \rightarrow N$，在接收端按同样控制信号还原.

Given a multiplexer circuit, the information cannot be accurately demultiplexed, unless exact knowledge of the multiplex circuit is known.

4.5.1 多路复用器

> Multiplexer (MUX)

$n \times 1$ MUX

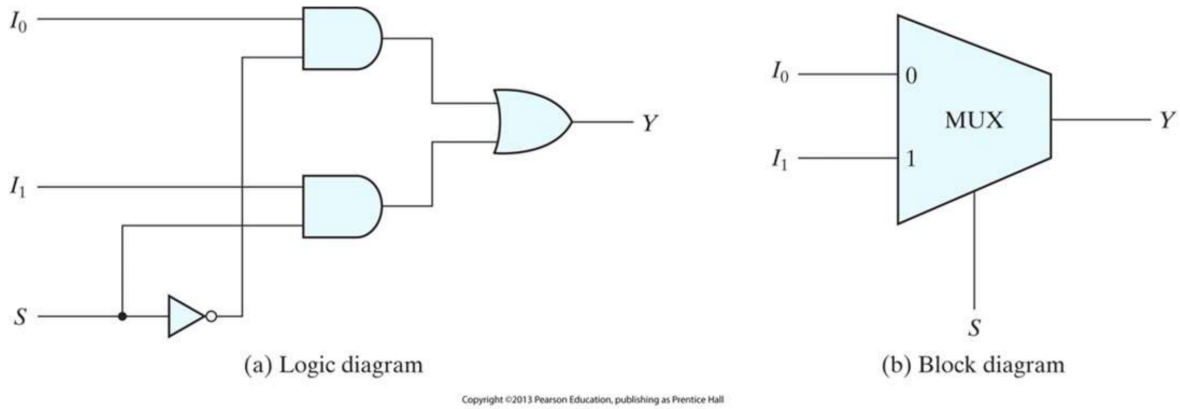


- n 条数据输入
- $\lceil \log_2 n \rceil$ 条 select line

  > 至少有 n 种组合，且条数越少越好，故 2 的对数上取整.

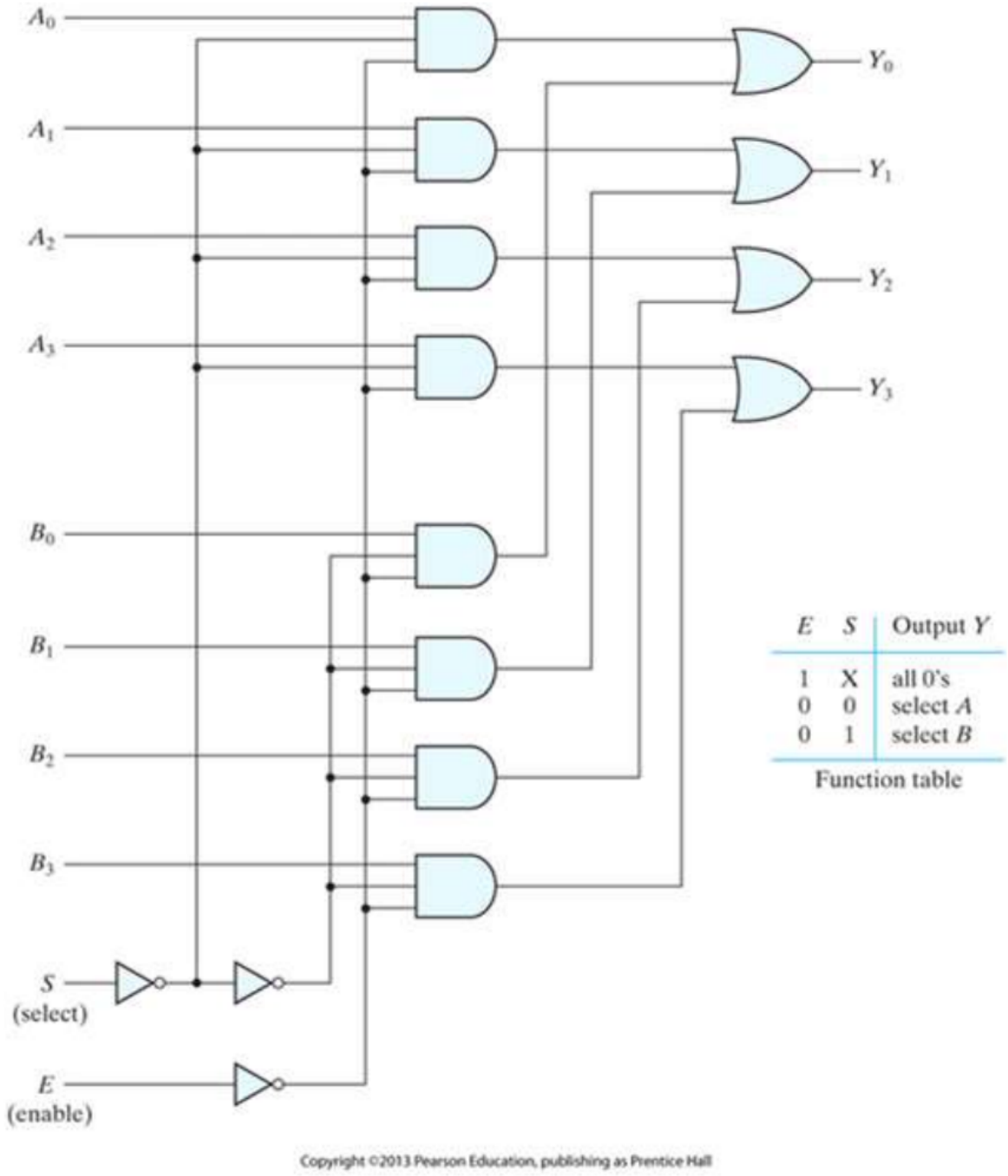
- 1 个输出

2×1 MUX



> 注意 select line 的构造，和通用加减法器的滑动开关类似，但这里是用与门，通用加减法器是用异或门.

Quadruple 2×1 Multiplexer



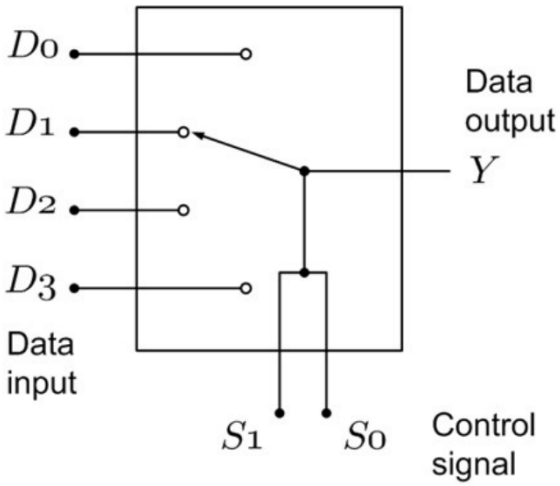
4×1 Multiplexer

> 2 个控制信号.

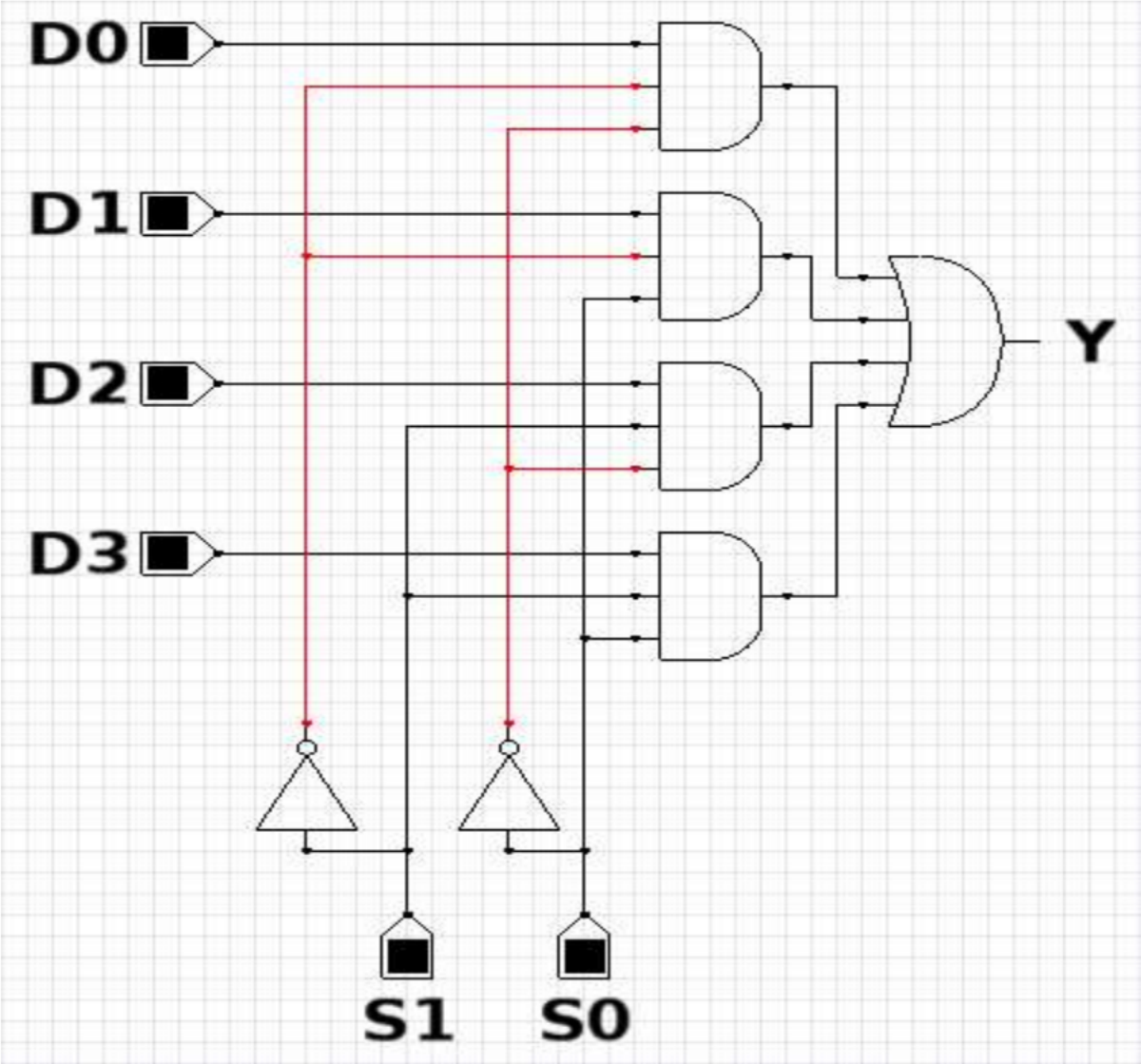
真值表

S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

the block diagram



逻辑电路

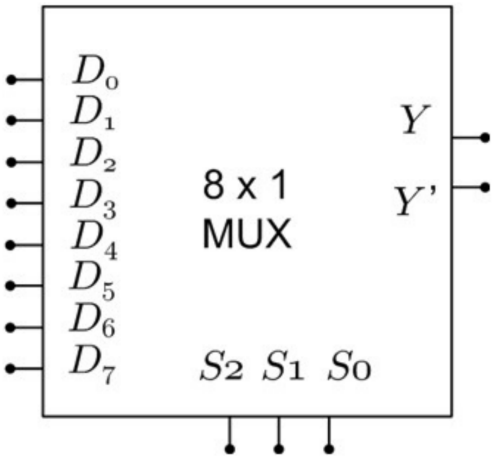


> 注意这里最终汇总使用了四输入或门，其位宽和输入信号（D0, D1, D2 或 D3）位宽必须一致. 多位宽的或门如果接收到多个高电位，会把它们逐位或操作，然后输出结果.

Commercial Package of 8×1 MUX

74151 is a commercial package of 8×1 MUX

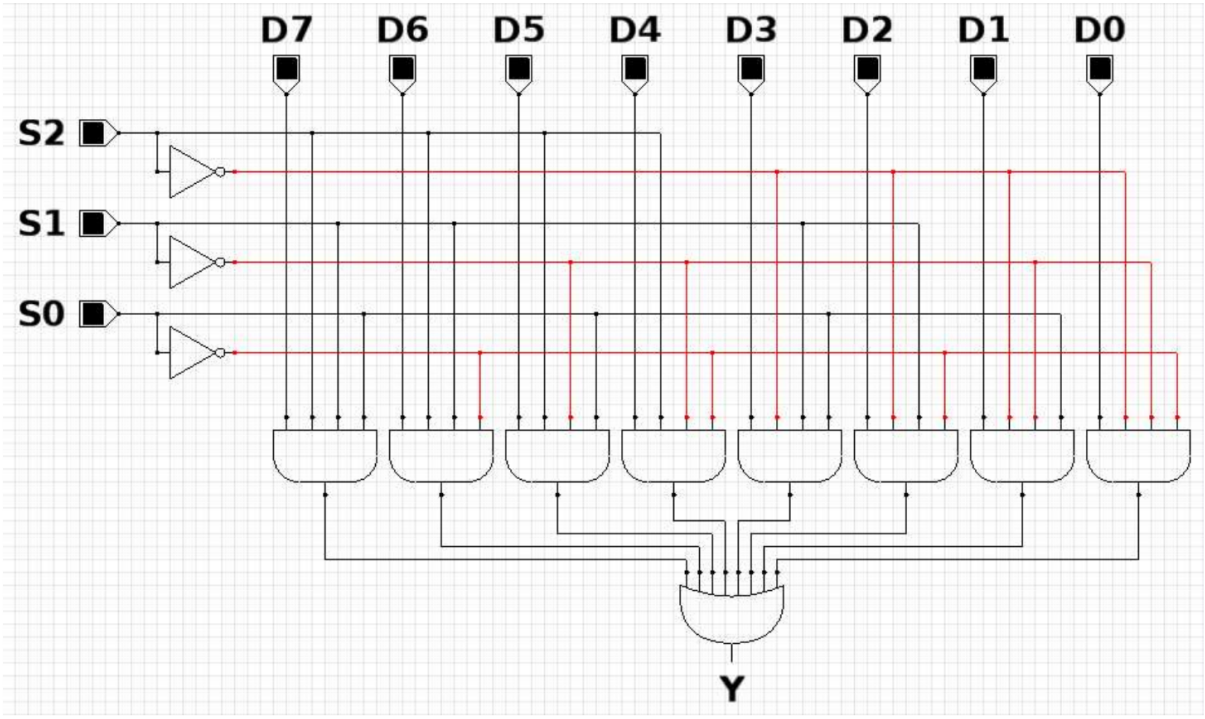
the block diagram



真值表

S2	S1	S0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

逻辑电路



4.5.2 多路分配器

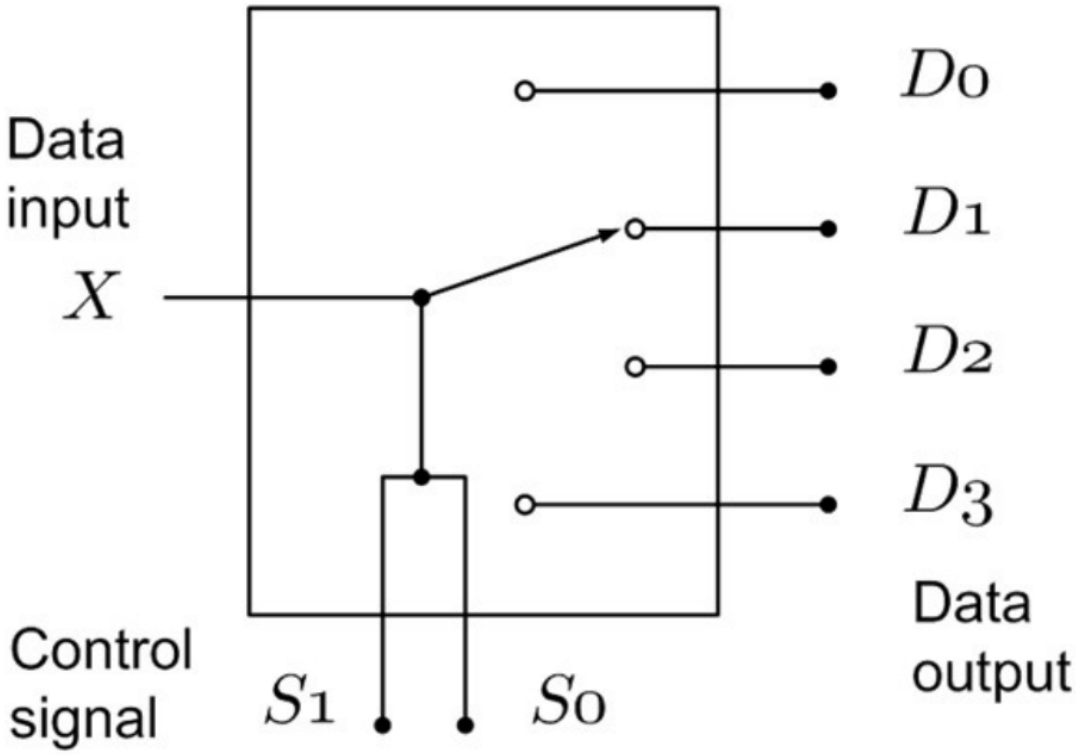
> Demultiplexer (DEMUX)

Consider a 4-output DEMUX

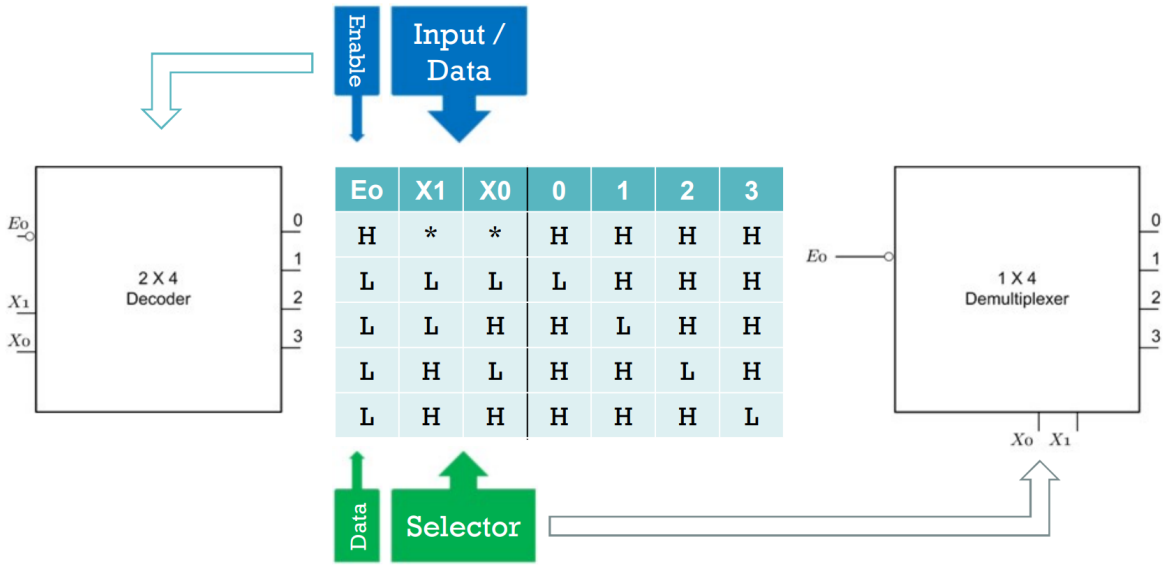
> D0 to D3

Only one output is selected and assigned the value of X

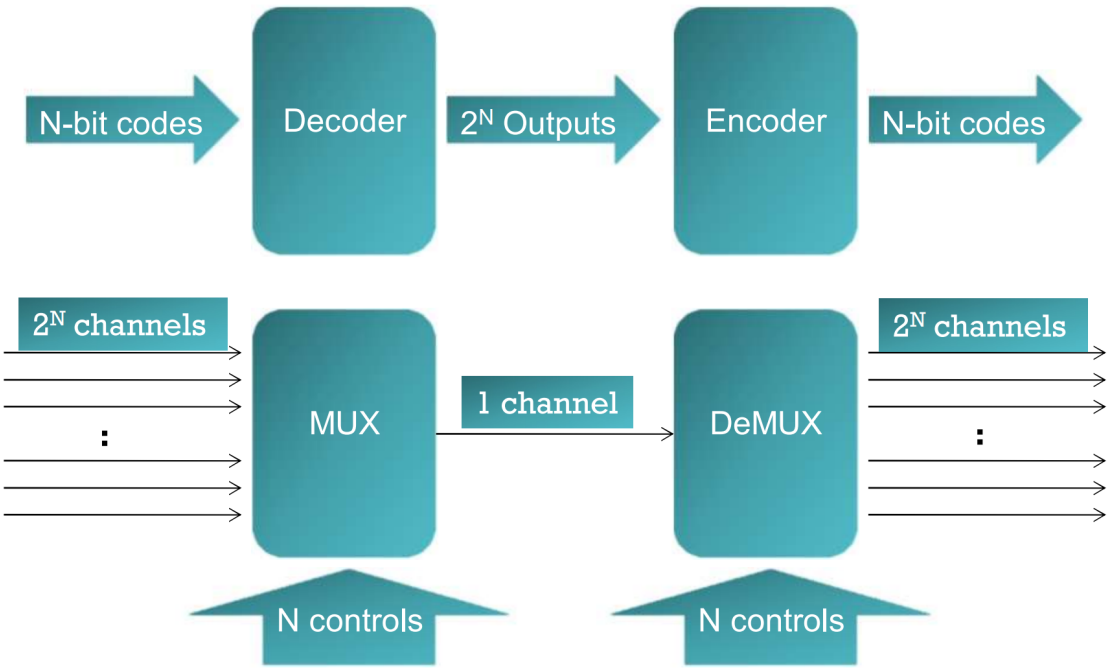
The selection is performed by control signals, S0 and S1



A DEMUX can be implemented by a decoder



4.6 总结



4.7 三态缓冲器

> Appendix: Tri-state Buffer

Lec 5 时序逻辑电路

5.1 时序逻辑类型

> Types of Sequential Circuits

5.1.1 同步时序逻辑

> Synchronous Sequential Logic

行为在离散时刻由时钟（clock）控制.

Its behaviour can be defined from the knowledge of its signals at discreteinstants of time

A timing device is required and called a clock generator

A clock signal is a periodic train of clock pulses

5.1.2 异步时序逻辑

> Asynchronous Sequential Logic

行为取决于输入变化的先后次序，对噪声更敏感，设计难度大.

Its behaviour depends upon the input signals at any instant of time, and the order in which the inputs change

5.2 存储元件

> Storage Elements

There are two types of storage elements in sequential circuits

- Latches
- Flip-flops

5.2.1 锁存器

> Latches

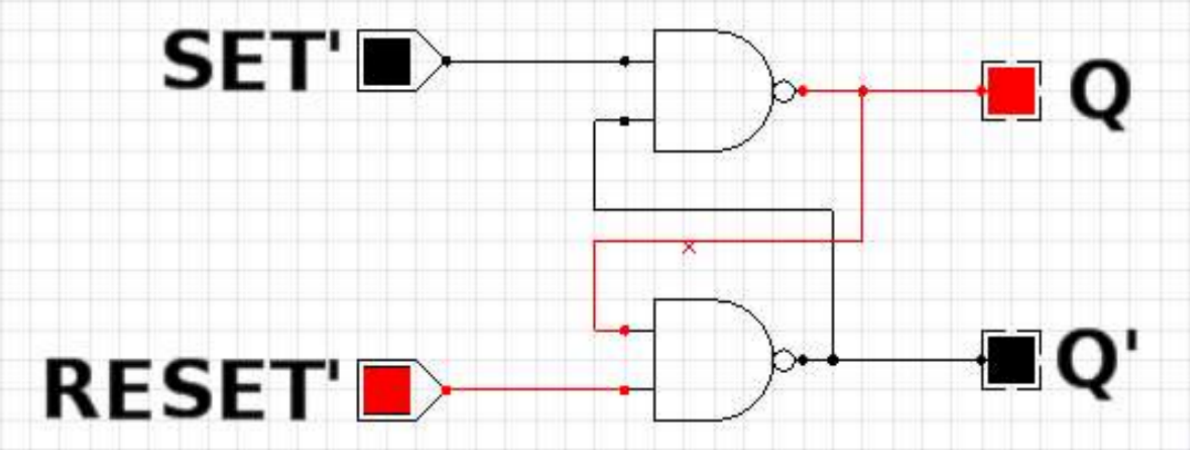
电平敏感（Level-sensitive）.

Not useful for synchronous sequential circuit, but they are the building blocks of flip-flops

① NAND 型 RS Latch

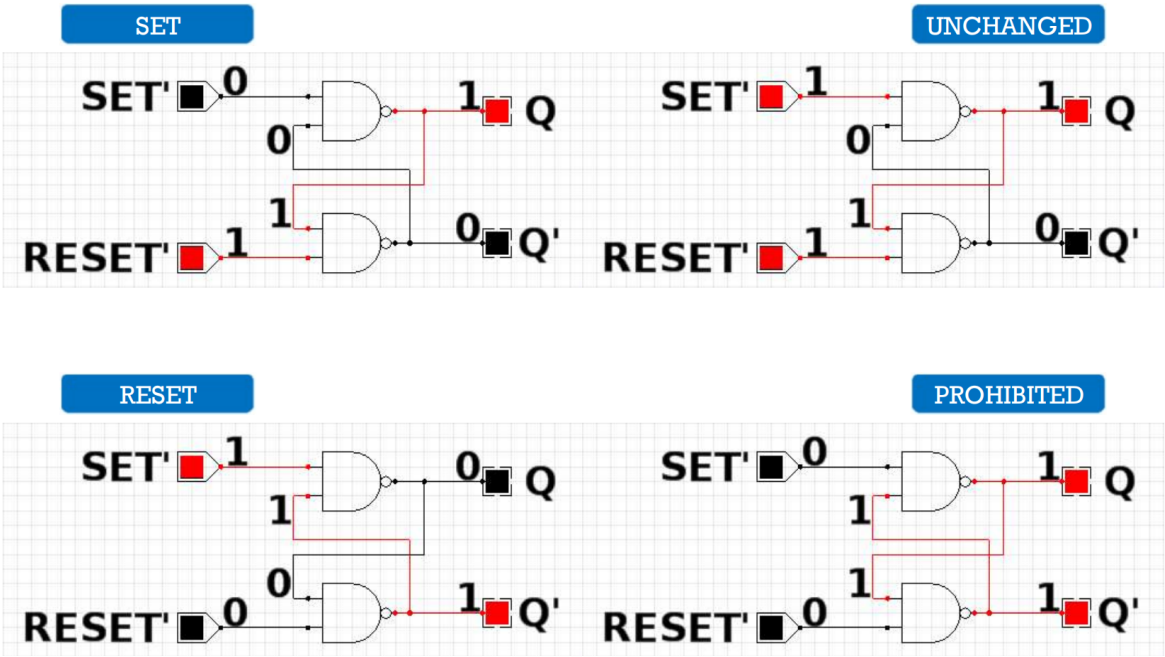
A crossed NAND RS (Reset-Set) latch is a digital circuit whose output is

- Set by the SET’ input (Active-LOW)
- Reset by the RESET’ input (Active-LOW)



> NAND 的特点：有一个为 0 输出就为 1.

State of RS Latch (NAND)



> 当输入发生变化，实际的物理电路会经历瞬态过程，直到两个 NAND 门的输出满足互相一致的逻辑. 但我们不研究瞬态，我们就研究稳定之后的状态.

注意，unchanged 状态有两种情况，$Q=1, Q'=0$ 或 $Q=0, Q'=1$. 首次上电时，其状态无法预测；其后会保持原有状态（Q' 和 Q 都保持）. 具体的物理机制不考.

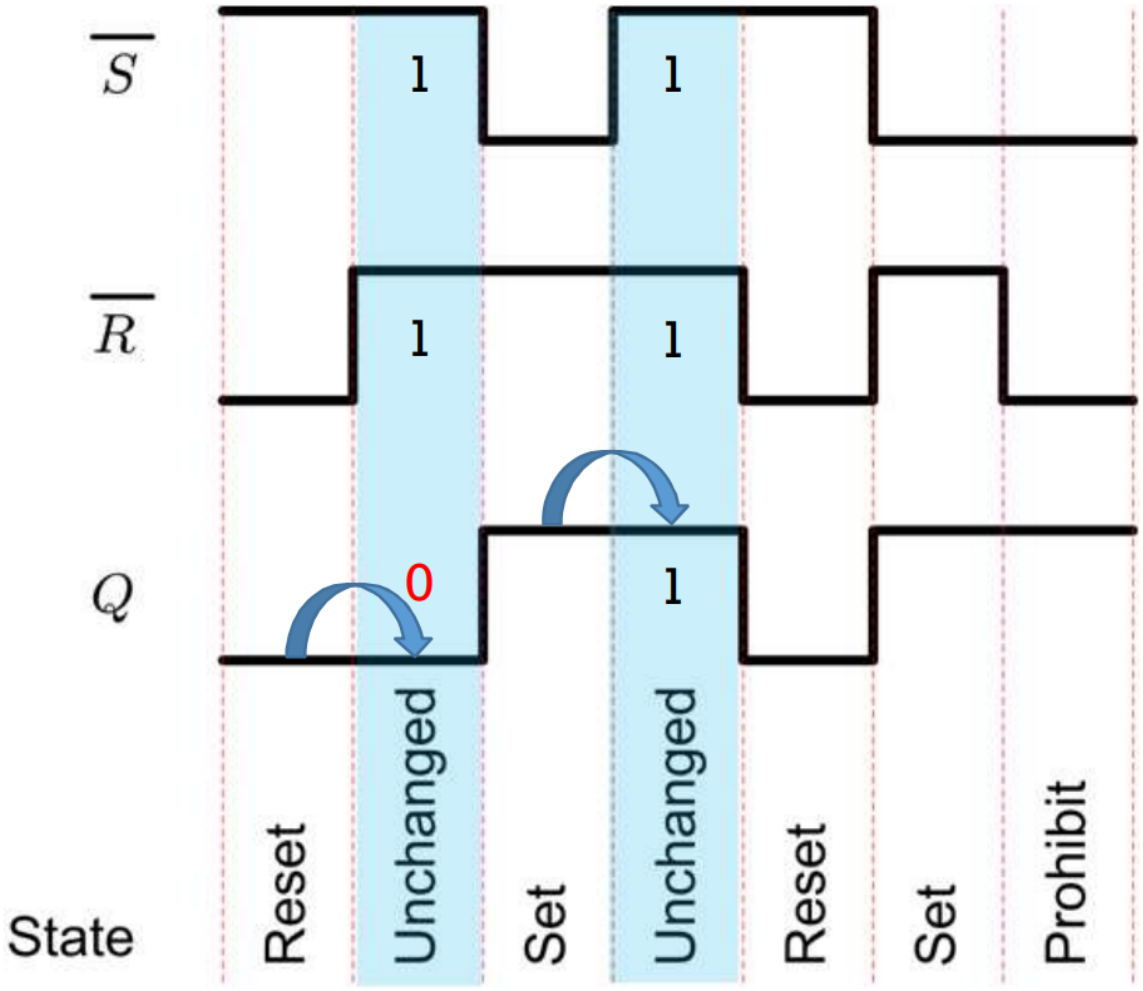
真值表

SET	RESET	SET’	RESET’	Q	Q’	STATE
1	1	0	0	1	1	PROHIBITED
1	0	0	1	1	0	SET
0	1	1	0	0	1	RESET
0	0	1	1	Q	Q’	UNCHANGED

> 注意，Q 和 Q' 不能为相同值.

时序波形图

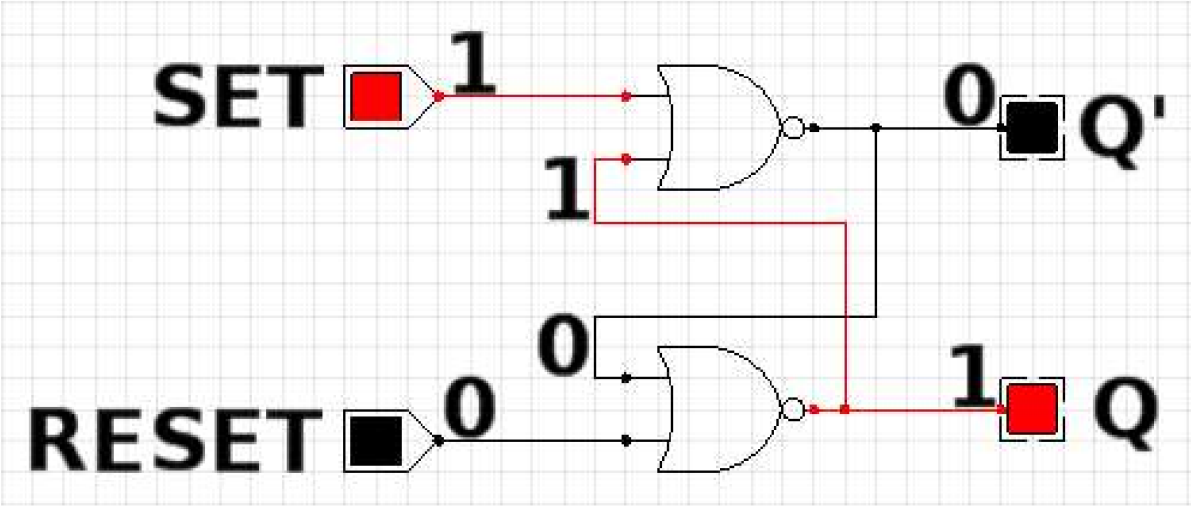
> Timing Diagram of RS Latch (NAND)



② NOR 型 RS Latch

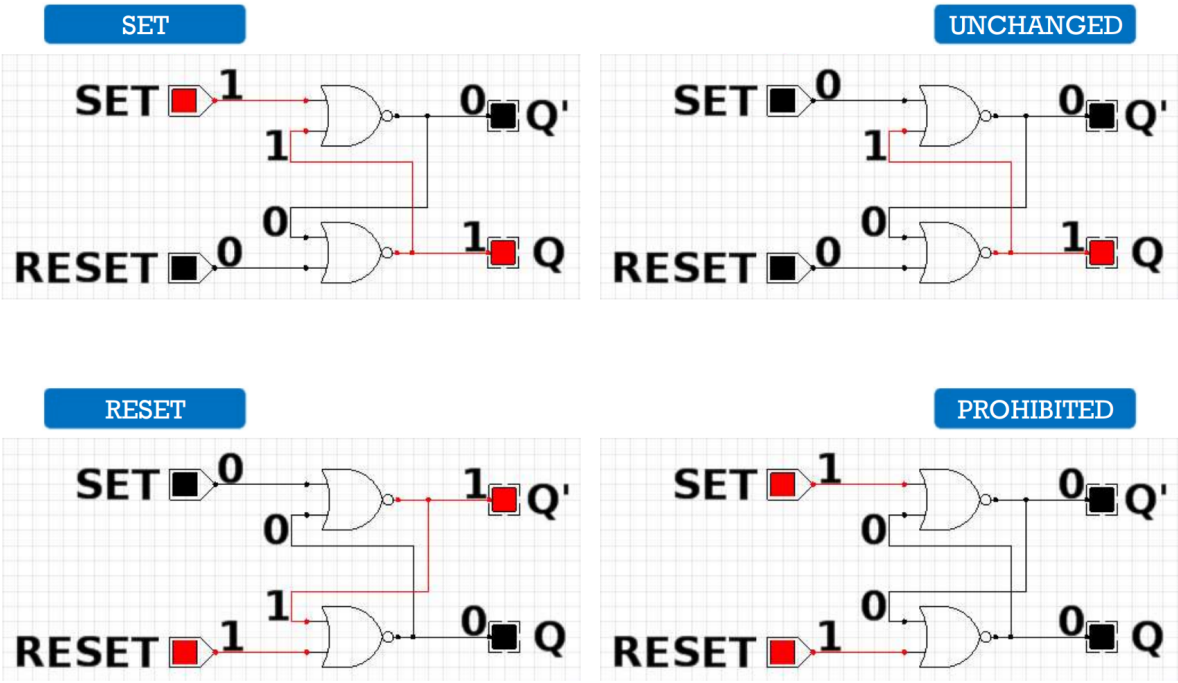
A crossed NOR RS (Reset-Set) latch is a digital circuit whose output is

- Set by the SET input (Active-HIGH)
- Reset by the RESET input (Active-HIGH)



> NOR 的特点：有一个为 1 输出就为 0.
>
> 注意输出是交叉排列.

There are 3 valid states (Set, Reset, Unchanged), and 1 prohibited state

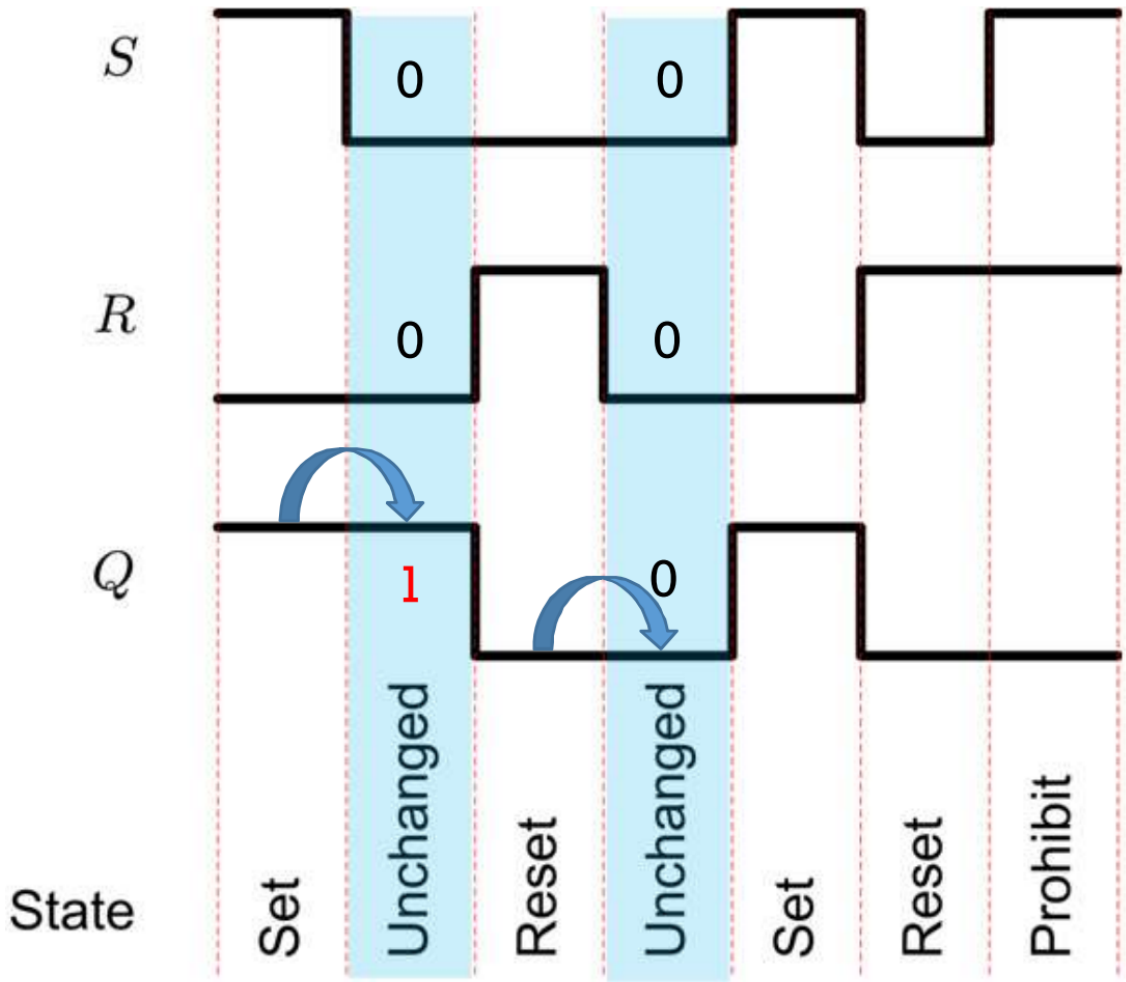


真值表

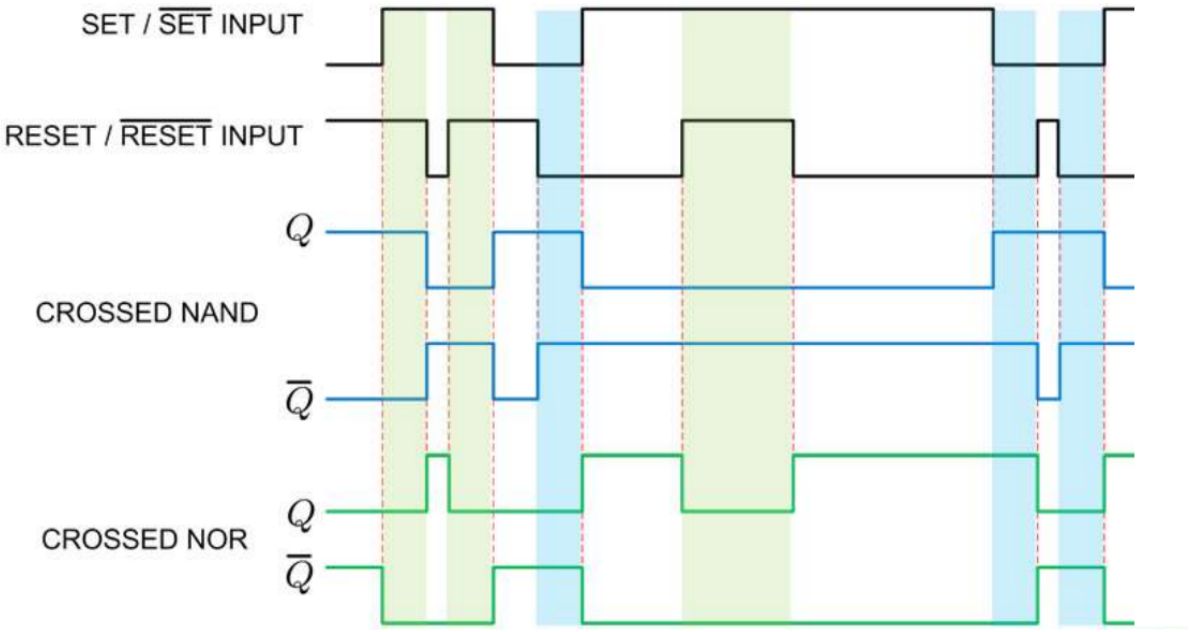
SET	RESET	SET’	RESET’	Q	Q’	STATE
1	1	0	0	0	0	PROHIBITED
1	0	0	1	1	0	SET
0	1	1	0	0	1	RESET
0	0	1	1	Q	Q’	UNCHANGED

> 注意，Q 和 Q' 不能有相同值.

时序波形图



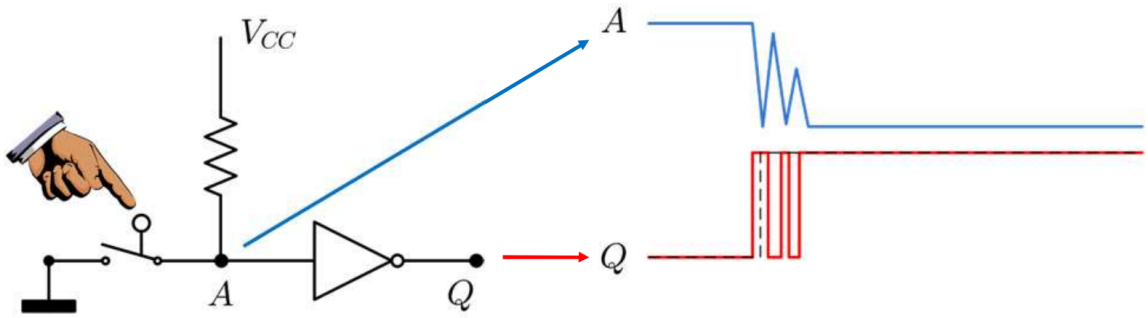
Although they are both RS Latches, the input signals are different (Active-LOW vs. Active-HIGH), the outputs Q will be totally different.



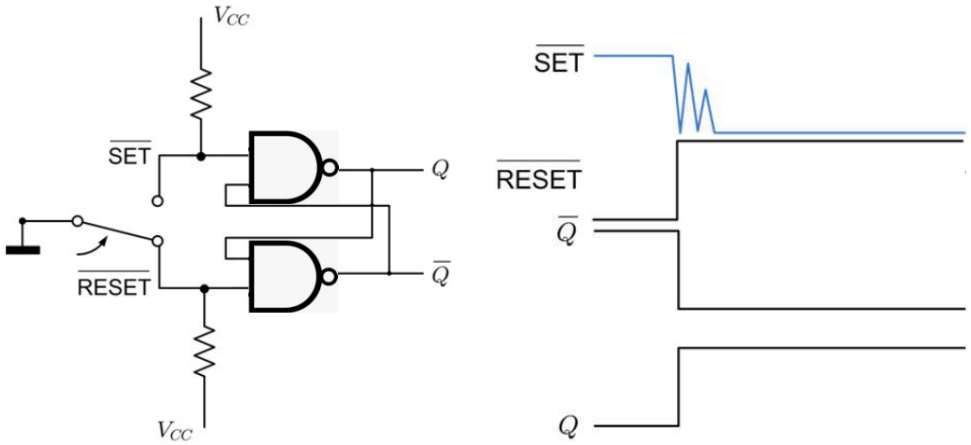
两种锁存器表现完全不同.

应用：电路消抖

机械按键在闭合/断开瞬间会抖动，使电路读到多次脉冲，用 RS Latch 锁存最终稳定状态，滤掉波动.



- To avoid the problem, a RS latch can be used to debounce the switch



5.2.2 触发器

> Flip-flops

Storage elements that controlled by a clock transition

边沿敏感（Edge-sensitive）.

Latch 是 Flip-Flop 的构建块.

时钟脉冲

> Clock Pulses

The clock signal is a rectangular pulse train or square wave

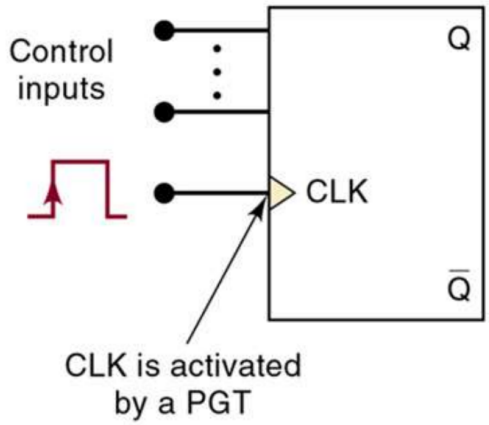
The FFs will change its state when there is a valid clock signal

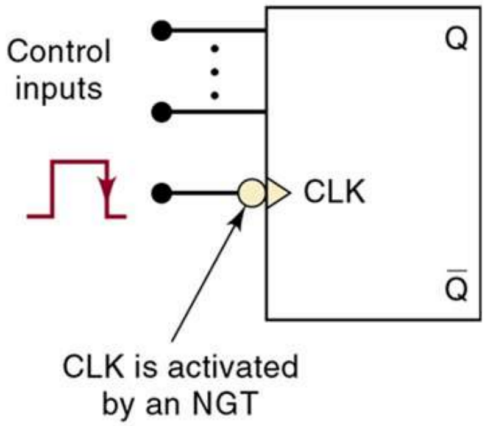
Clock inputs are usually labeled as CLK, CK, or CP

时钟信号是一种方波脉冲信号，周期性地从 0 到 1，再从 1 到 0.

FF 在时钟信号的有效跳变时改变状态.

> 只在时钟沿到来的瞬间更新状态.

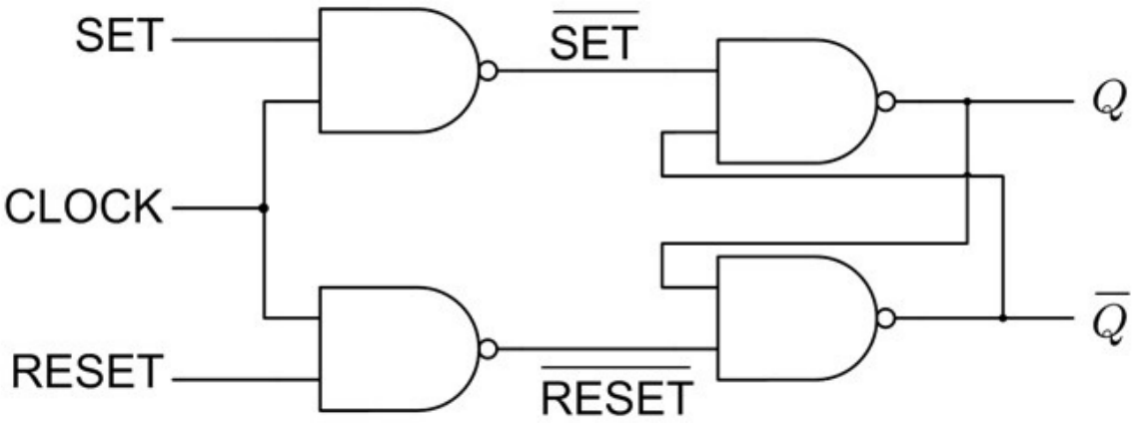




> PGT：上升沿触发（Positive-Going Transition），常用
>
> NGT：下降沿触发（Negative-Going Transition）

① RS-FF

> Clocked Flip-Flop



To make a RS latch be useful in synchronous sequential circuit

Modified version with a clock input and becomes a Flip Flop (FF)

3 inputs: RESET, SET, and CLOCK

2 outputs: Q and Q'

The CLOCK input is also known as ENABLE

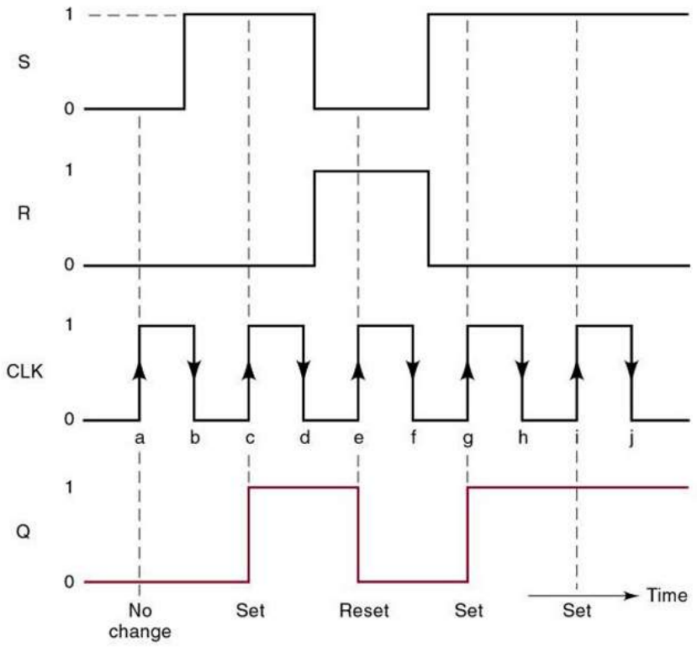
- When the CLOCK is LOW, the FF is disabled
- When the CLOCK is HIGH, the FF is enabled

真值表

CLOCK	S	R	Q	Q'	STATE
0	0	0	Q	Q'	UNCHANGED
0	0	1	Q	Q'	
0	1	0	Q	Q'	
0	1	1	Q	Q'	
1	0	0	Q	Q'	
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	1	1	UNUSED

时序波形图

RS-FF		
S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	?



> 上升沿触发，因为时钟从 1 变 0 时是 unchanged.

注意，这里仍然有无用（非法）的情况（S, R 均为 1 时）. 为了解决这个问题，引入 D-FF.

② D-FF

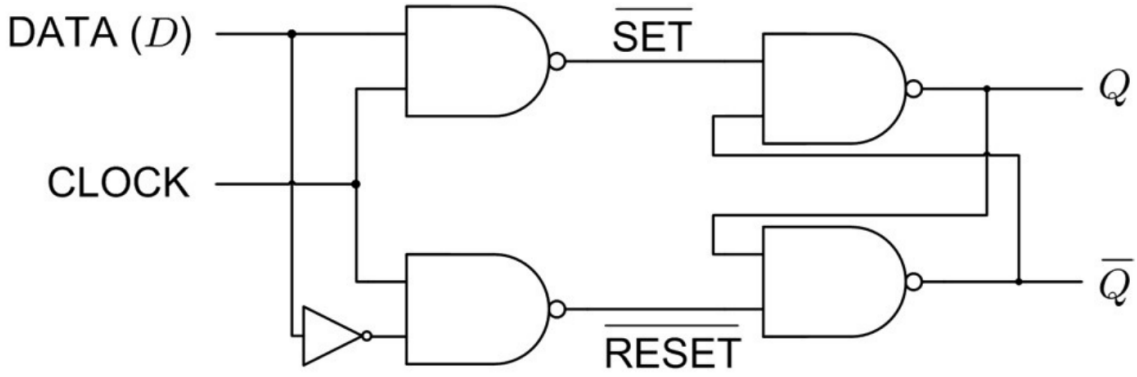
> Data Flip-Flop (DFF)，又称 D 触发器

本质：在 RS-FF 外加一路反相器，把 D 同时送到 SET 和 $\overline{RESET}$

常用于「锁存一位数据、实现寄存器」.

When the CLOCK is HIGH, the output Q will be equals to input D

When the CLOCK is LOW, the output Q will remain unchanged

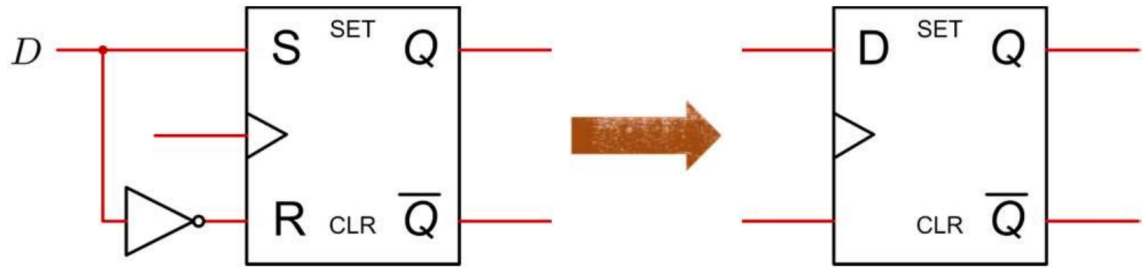


真值表

D	CLOCK	Q	Q’	STATE
0	0	Q	Q’	UNCHANGED
1	0	Q	Q’	
0	1	0	1	
1	1	1	0	

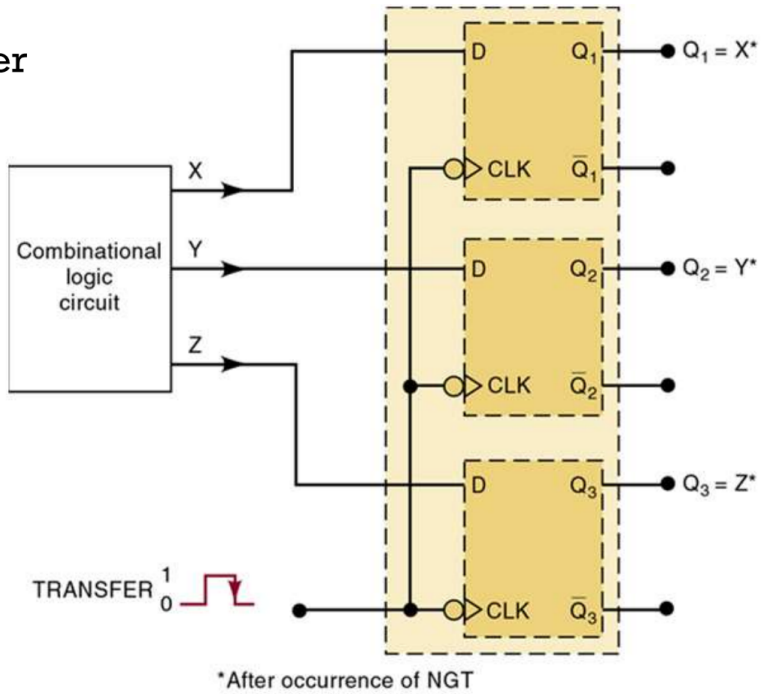
> 上升沿触发.
>
> 注意，Q 和 D 在逻辑上并不总是相等，只有在时钟沿触发时，Q 才会被更新为 D 的值.

D-FF consists of a RS-FF



Use of D-FF

- Parallel data transfer



下降沿触发：CLK 设置反向.

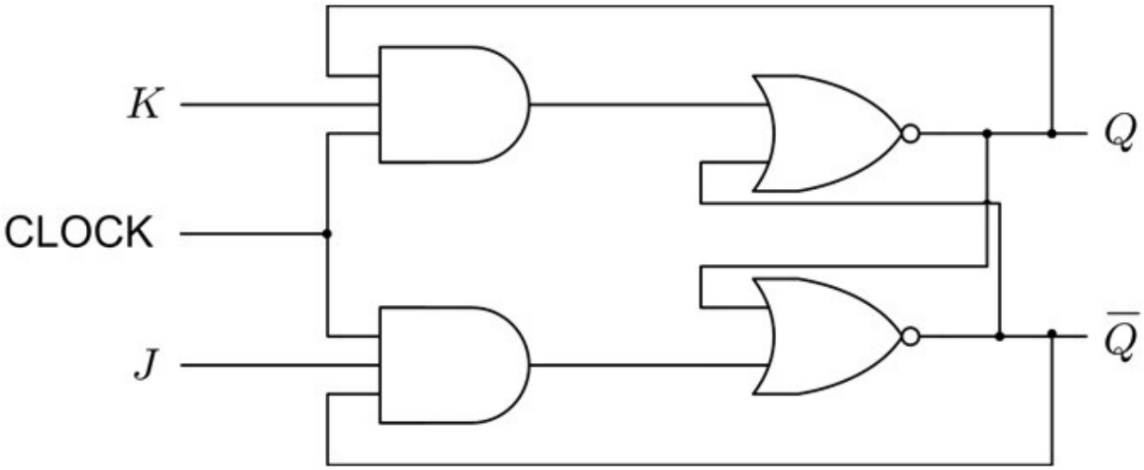
③ JK-FF

> JK Flip-Flop (JK-FF)

To make use of the unused state at NOR RS-FF，引入 JK-FF

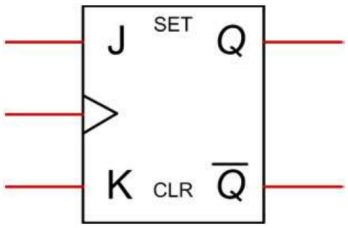
JK-FF is similar to RS-FF, except when both J and K are HIGH, the FF toggles（翻转） its output value

> 通过额外反馈逻辑避免非法状态，比 RS-FF 更强大.
>
> 注意，NAND 型 RS-FF 不能这么接，具体不考.



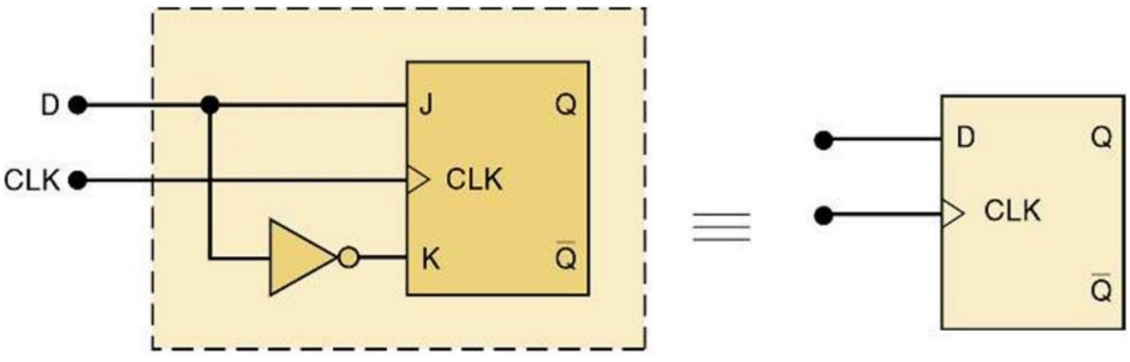
真值表

CLOCK	J	K	Q	Q'	STATE
A	0	0	Q	Q'	**UNCHANGED**
A	1	0	1	0	**SET**
A	0	1	0	1	**RESET**
A	1	1	Q'	Q	**TOGGLE**



> "A" stands for Active, it could be level, rising-edge, or falling-edge trigger
>
> 注意 JK-FF 中 Q 和 Q' 永远相反，所以 Toggle 虽然是翻转，但也等价于各自反转自己的 bit.

An edge-triggered D flip-flop can be implemented by an INVERTER and an edge-triggered J-K flip-flop



> 这时候反馈逻辑用不上，因为 D 和 D' 不会同时为 1.

④ T-FF

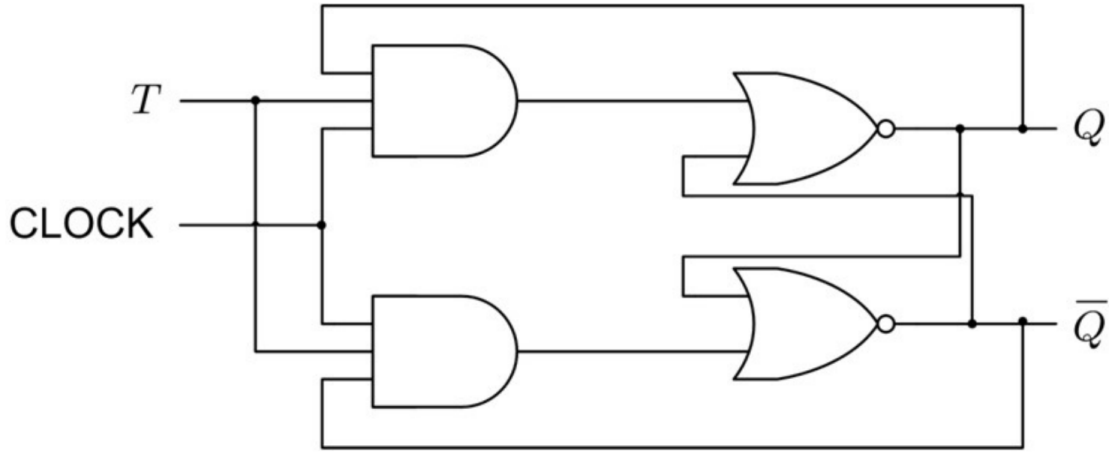
> Toggle Flip-Flop (T-FF)

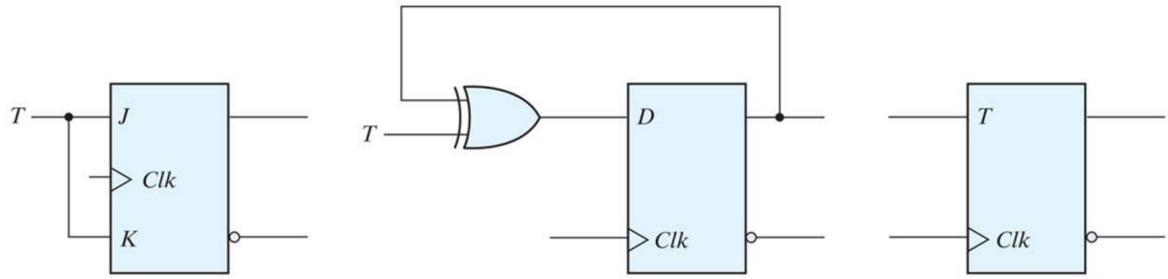
For repeated toggling of the output values, JK-FF is modified as T-FF.

将 JK-FF 两输入短接得到单输入 T (Toggle).

$T=1$ 时，输出翻转；$T=0$ 时保持.

应用：计数器、分频器.





(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

注意 D-FF 构造 T-FF：

T-FF 的翻转行为可以通过 $D=T\oplus Q$ 实现，其中 Q 是上一次激活时的输出值. 在时钟沿来临的那一刻，D 的值根据当前的 Q 和 T 计算出来，然后 D 的值被缩存进 Q. 如果此时 $T=1$，则 Q 会不断翻转.

真值表

CLOCK	T	Q	Q’	STATE
A	0	Q	Q’	**UNCHANGED**
A	1	Q’	Q	**TOGGLE**

总结：FF 的真值表

RS-FF		
S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	?

D-FF	
D	Q
0	0
1	1

JK-FF		
J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	Q’

T-FF	
T	Q
0	Q
1	Q’

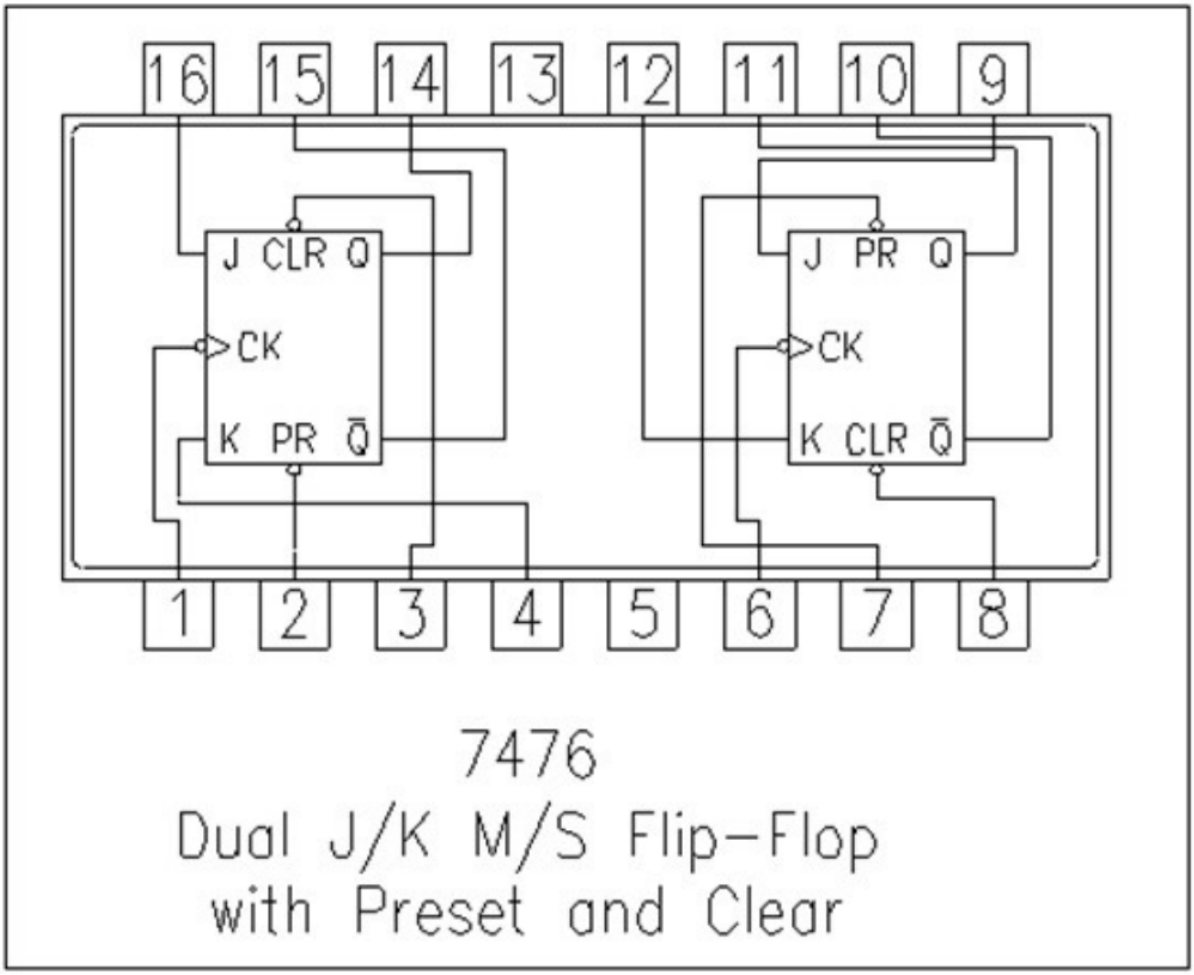
5.3 预置和清零

> Preset and Clear

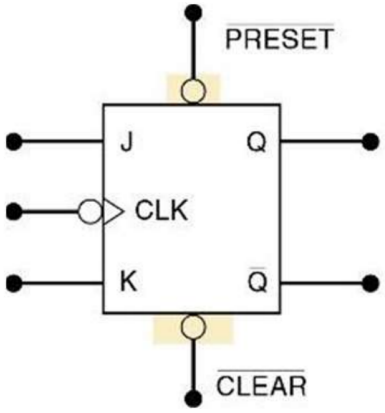
Inputs that depend on the clock are synchronous

触发器除同步输入外，经常带异步 Preset (PR/PRE) 与 Clear (CLR) 端：不受时钟控制，可在任意时刻把 Q 设为 1 或 0.

If the asynchronous inputs are not used, they will be “inactive”



7476
Dual J/K M/S Flip-Flop with Preset and Clear

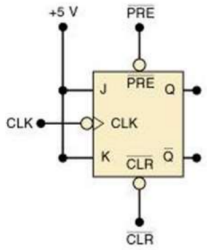


J	K	Clk	$\overline{PRE}$	$\overline{CLR}$	Q
0	0	↓	1	1	Q (no change)
0	1	↓	1	1	0 (Synch reset)
1	0	↓	1	1	1 (Synch set)
1	1	↓	1	1	$\overline{Q}$ (Synch toggle)
x	x	x	1	1	Q (no change)
x	x	x	1	0	0 (asynch clear)
x	x	x	0	1	1 (asynch preset)
x	x	x	0	0	(Invalid)

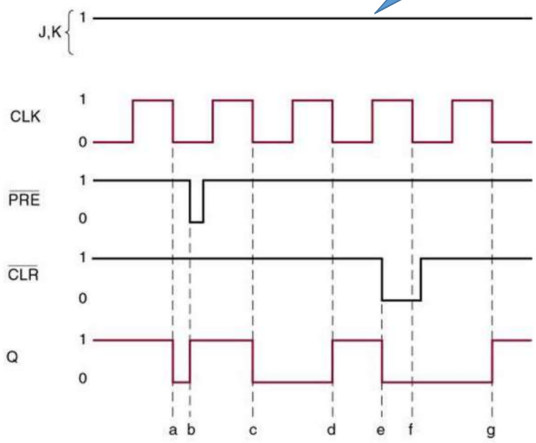
PRESET AND CLEAR

When J = K = 1, Q = Q'

- A J-K FF that responds to a NGT on its clock input and has active-LOW asynchronous inputs



Point	Operation
a	Synchronous toggle on NGT of CLK
b	Asynchronous set on $\overline{PRE}$ = 0
c	Synchronous toggle
d	Synchronous toggle
e	Asynchronous clear on $\overline{CLR}$ = 0
f	$\overline{CLR}$ overrides the NGT of CLK
g	Synchronous toggle



5.4 触发和定时

> Triggering and Timing

In synchronous sequential logic, an external clock is required as a periodic pulses to trigger the synchronous FFs

There are 3 types of clock triggering methods

Level triggering which relies on the magnitude of the pulses

Edge triggering which relies on the transition in the waveform

Master Slave triggering which relies an extra FF to generate triggering signal

5.4.1 触发

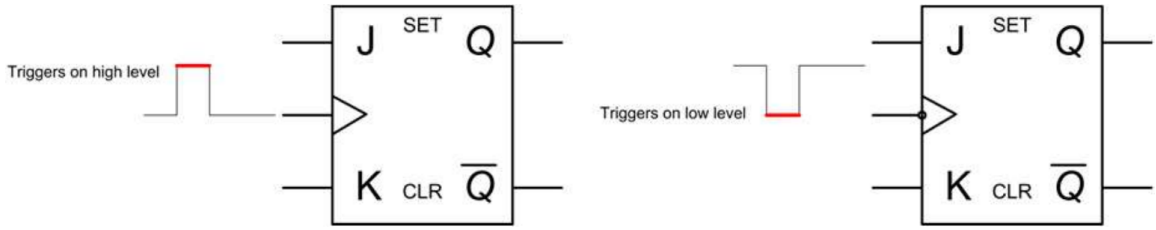
> Triggering

① 电平触发

> Level Triggering

The circuit is activated by either HIGH or LOW level of the clock signal

- HIGH level triggering
- LOW level triggering



The triggering process depends on the duration of the pulse, which may cause instability problem

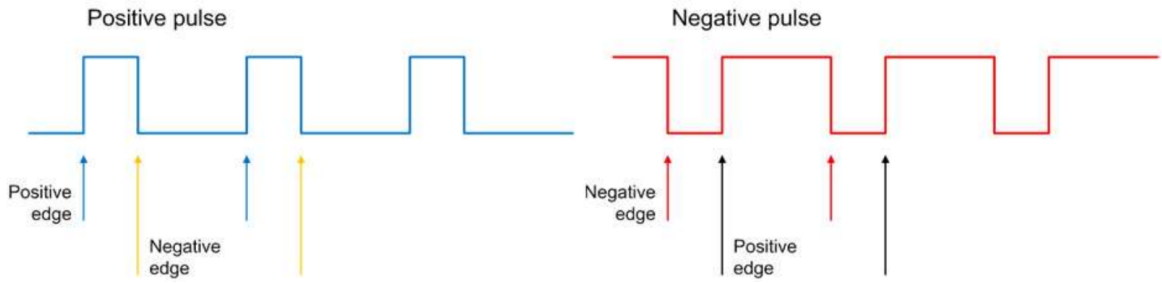
② 边沿触发

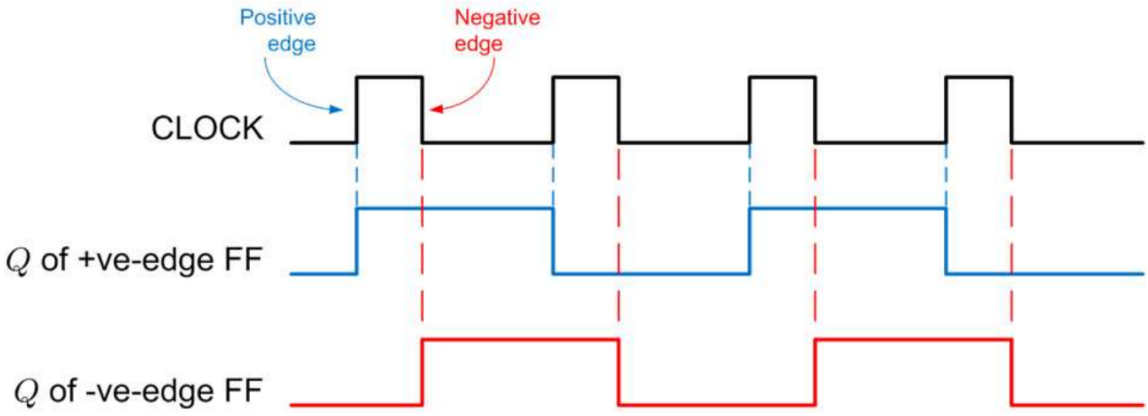
> Edge Triggering

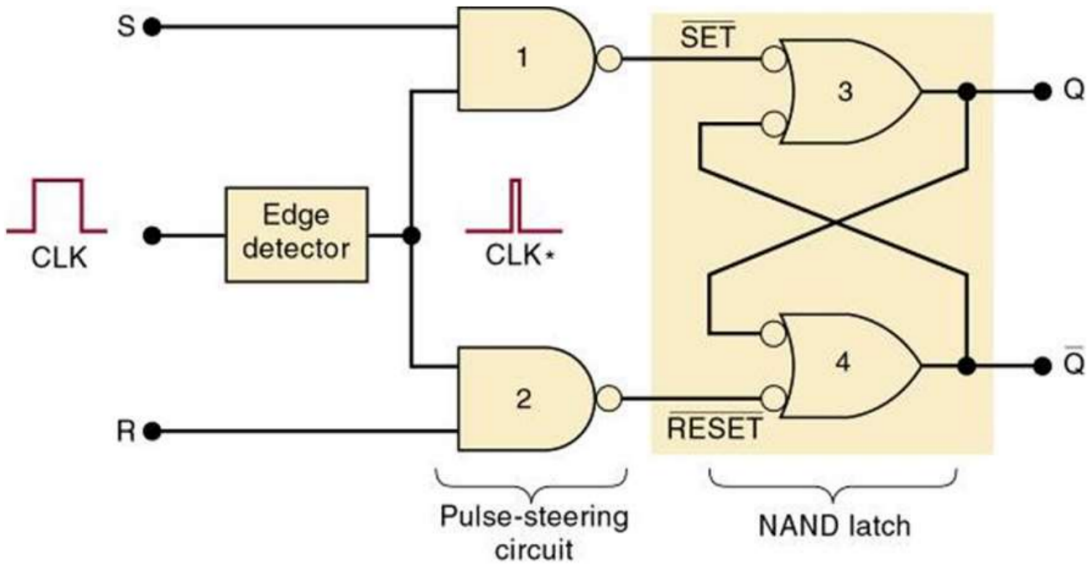
The circuit is activated by the transitions of the clock signal

Rising-edge / Positive-pulse triggering, from LOW to HIGH

Falling-edge / Negative-pulse triggering, from HIGH to LOW





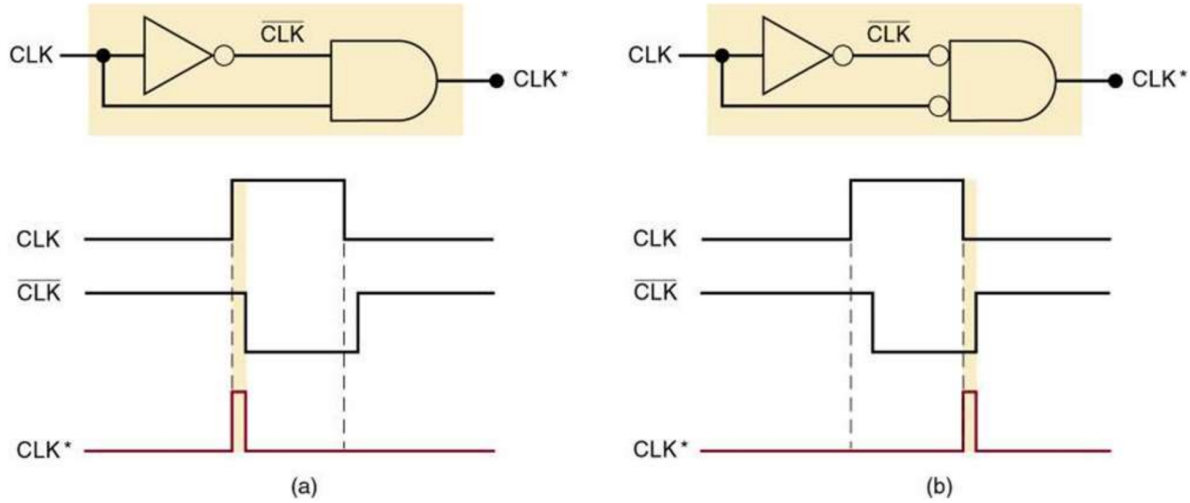


RS Latch 本身是电平敏感，通过一个边沿检测器将时钟的上升沿或下降沿转换成一个短而窄的脉冲信号，从而实现 RS-FF 的边沿敏感.

边沿检测器

> Edge Detector

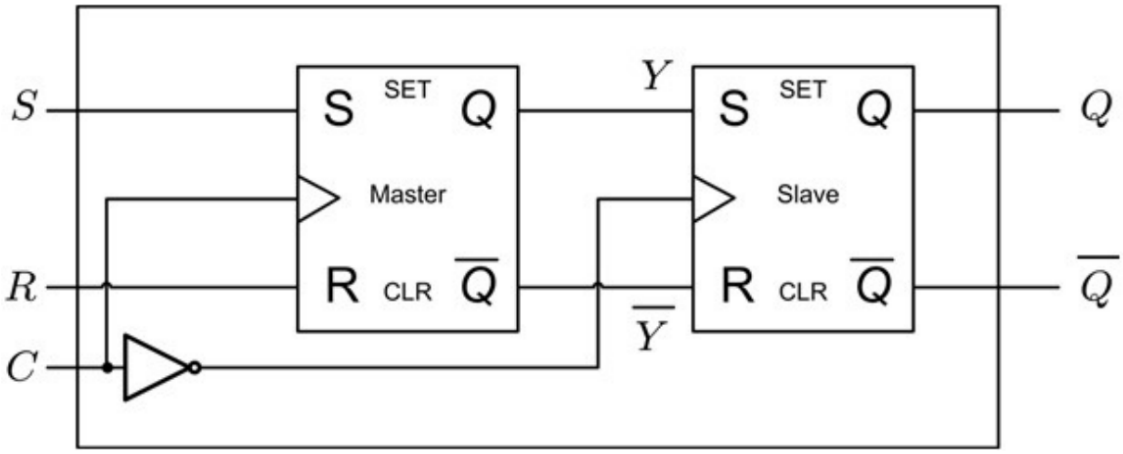
通过 RC 网络或门级组合实现正/负脉冲产生器 (PGT/NGT)，是多数现成触发器内部结构的原理.



③ 主从触发

> Master-Slave Triggering

将两级 RS-FF 级联：Master 用 CLK，Slave 用 $\overline{CLK}$.



If C is LOW, the master is DISABLED (inputs S and R have no effect to the circuit) while the slave is ENABLED (the slave clock is HIGH), and producing the slave output Q = Y

主锁存器 Master 是"禁用"的（不响应输入：S 和 R）
从锁存器 Slave 是"启用"的（因为它的时钟是 C 的反相）
Slave 输出 Q = Y（输出的是 Master 上一次的结果）

If C becomes HIGH, the master is ENABLED and the information of the master inputs S and R is transmitted to the circuit, BUT the slave being DISABLED will be isolated from an impact from these inputs – slave outputs remain intact

Master 被启用，开始采样输入 S/R
Slave 被禁用，不受 Master 当前变化的影响
Slave 保持上一个状态（锁住输出）

Note: there are delays the change in the master clock and the slave outputs –avoiding glitches (unwanted logic pulses resulting from multiple logic components operating at different speeds)

Master-Slave Flip-Flop 实际是在 CLK 的"下降沿"更新输出，或者说：它是"上升沿采样，下降沿输出".

5.4.2 定时

一些参数介绍. 待完成.

5.5 有限状态机

> Finite State Machine, FSM

FSM is a mathematical model of computation used in many digital systems.

FSM controls the behavior of systems and dataflow paths.

The machine is in only one state at a time.

It can change from one state to another by a triggering event or condition, this is called a transition

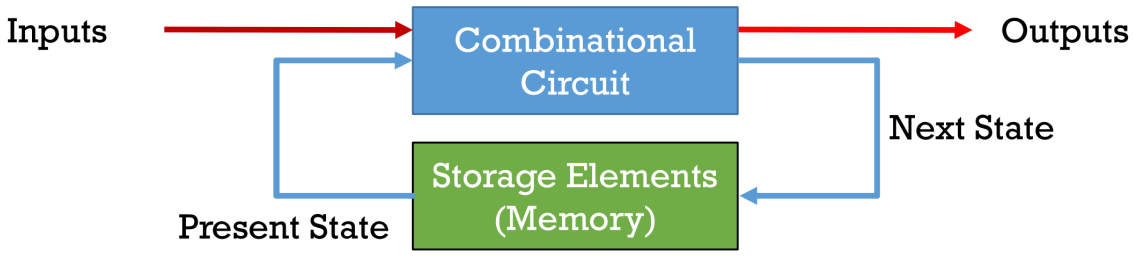
There are two types of FSMs

- Mealy FSM
- Moore FSM

5.5.1 米利型有限状态机

> Mealy Finite State Machine, Mealy FSM

Mealy FSM 的输出依赖**当前状态**和**当前输入**.



① 基本概念

一个 Mealy FSM 由以下几个部分组成:

- 状态集合（States）：系统可能处于的所有状态.
- 输入集合（Inputs）：系统接收的外部输入.
- 输出集合（Outputs）：系统产生的输出.
- 状态转移函数（Transition Function）：当前状态和输入决定下一个状态.
- 输出函数（Output Function）：当前状态和输入决定输出.
- 初始状态（Initial State）：系统开始运行时所处的状态.

② 特点

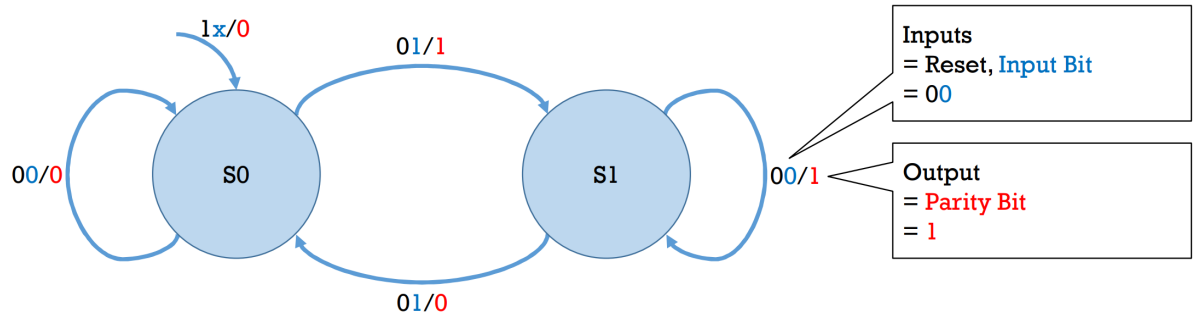
节省状态数，响应速度快.

③ 例子：奇偶校验器

> Mealy FSM Example (Parity Checker)

以偶校验（Even Parity）为例:

> 见 1.6.3 奇偶校验 ② 偶校验 .



S0：当前 1 的个数是偶数.

S1：当前 1 的个数是奇数.

这是一个用于检查并生成偶校验位的 Mealy 状态机. 它根据当前状态（1 的数量是奇数还是偶数）和输入位，输出一个校验位，使得总的 1 的数量为偶数.

箭头上标：输入/输出

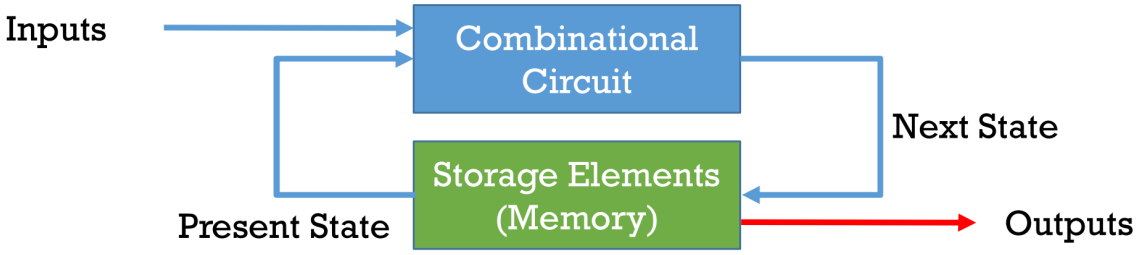
输入（两位）：ResetBit InputBit

输出（一位）：ParityBit

5.5.2 摩尔型有限状态机

> Moore Finite State Machine, Moore FSM

Moore FSM 的输出仅依赖于**当前状态**，而不依赖输入.



① 基本概念

一个 Moore FSM 由以下几个部分组成：

- 状态集合（States）：系统可能处于的所有状态.
- 输入集合（Inputs）：系统接收的外部输入.
- 输出集合（Outputs）：系统产生的输出.
- 状态转移函数（Transition Function）：当前状态和输入决定下一个状态.
- 输出函数（Output Function）：当前状态决定输出.
- 初始状态（Initial State）：系统开始运行时所处的状态.

注意，Moore FSM 的输出仅由当前状态决定，但是下一个状态仍然由当前状态和输入共同决定.

② 特点

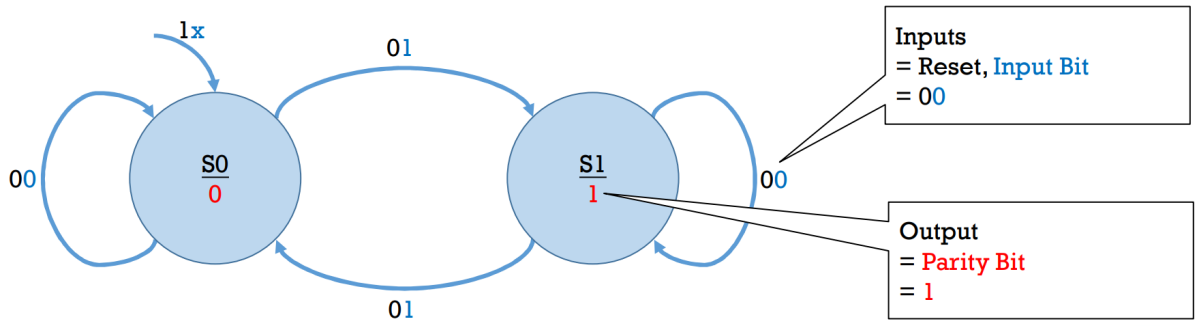
需要更多状态表达相同功能，响应速度较慢. 但设计更清晰，适合硬件实现.

③ 例子：奇偶校验器

> Moore FSM Example (Parity Checker)

以偶校验（Even Parity）为例：

> 见 `1.6.3 奇偶校验 ② 偶校验`.



5.5.3 对比

Mealy 型 FSM：每次输入变化时，**立即输出响应**.

Moore 型 FSM：输出在进入某状态后才改变，**与输入无直接关系**.

5.6 时序电路设计

> Sequential Circuit Design

从 FSM 到逻辑电路，实现步骤如下：

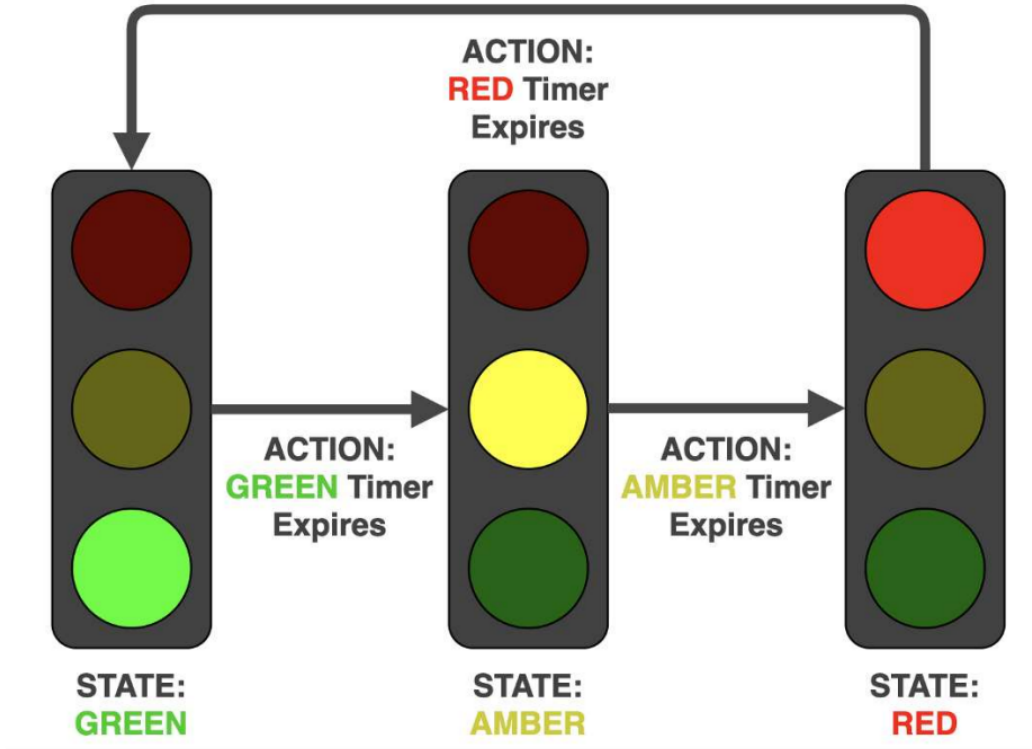
- 画出状态图（State Diagram）
- 写出状态表（State Table）/ 真值表（Truth Table）
- 为每个比特选择适当的触发器（Flip-Flop）
- 添加触发器输入列（Flip-Flop inputs）

- 求解触发器输入的逻辑表达式
- 求解输出的逻辑表达式
- 实现电路

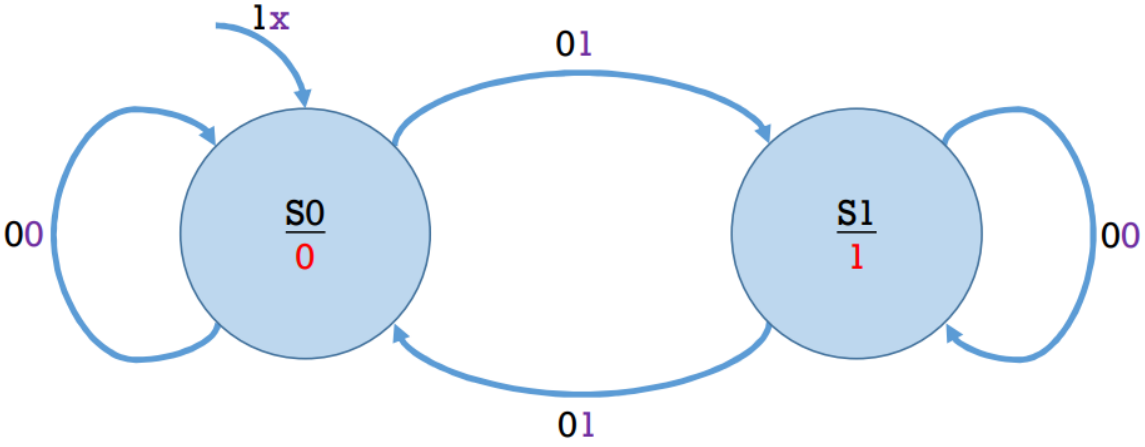
5.6.1 画状态图

> Design a State Diagram

交通信号灯状态图



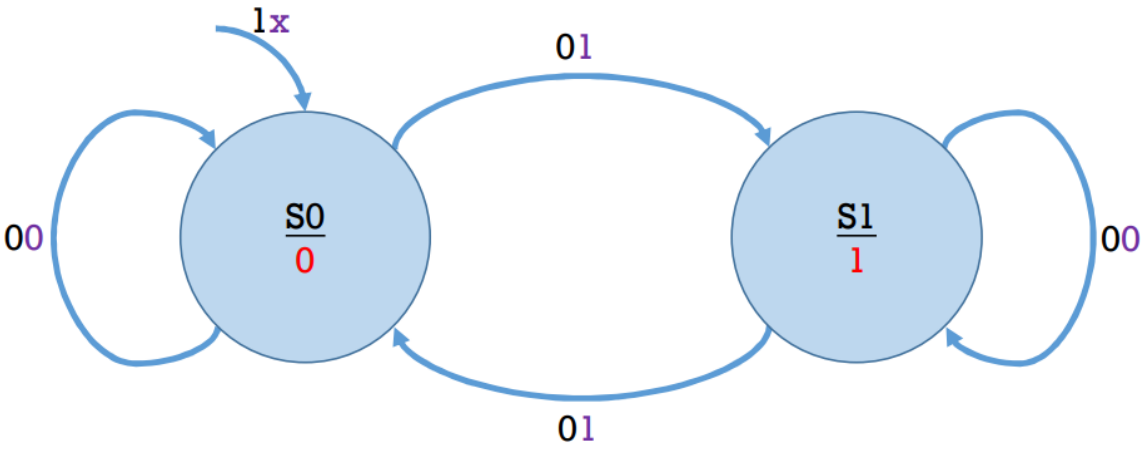
State diagram of a parity checker in Moore FSM



5.6.2 写状态表 / 真值表

> State Table / Truth Table

以 State diagram of a parity checker in Moore FSM 为例，



State Table / Truth Table 如下：

Replace the states by bits

Present State	Input	Next State	Output
Q	X	Q+	Z
S0	0	S0	0
S0	1	S1	0
S1	0	S1	1
S1	1	S0	1

Present State	Input	Next State	Output
Q	X	Q+	Z
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

> 真值表就是把状态表中的状态用二进制数代替. 如果有 n 个不同的状态数，则真值表中状态对应的二进制数最少为 $\lceil \log_2 n \rceil$ 位.

5.6.3 选触发器

> FF Selection

Truth Tables of FFs

RS-FF		
S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	?

D-FF	
D	Q
0	0
1	1

JK-FF		
J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	Q'

T-FF	
T	Q
0	Q
1	Q'

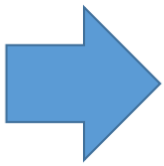
From Truth Table to **Excitation Tables**

> 为了从当前状态转换到下一状态，触发器的输入（即激励）应该是什么.

The excitation table helps to answer the question that "In order to get a particular next state output Q+, what should be the input?"

Use RS-FF as an example, where X="Don't Care" :

RS-FF		
S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	?



RS-FF			
Q	Q+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Excitation Tables of FFs

RS-FF			
Q	Q+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

D-FF		
Q	Q+	D
0	0	0
0	1	1
1	0	0
1	1	1

JK-FF			
Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

T-FF		
Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

5.6.4 添加触发器输入列

> Add FF Inputs

以 State diagram of a parity checker in Moore FSM 为例，假设采用 D-FF

Present State	Input	Next State	FF Input	Output
Q	X	Q+	D	Z
0	0	0	0	0
0	1	1	1	0
1	0	1	1	1
1	1	0	0	1

5.6.5 触发器输入表达式

> FF Inputs' Equations

Solve the equations for Flip-Flop Inputs

Present State	Input	Next State	FF Input	Output
Q	X	Q+	D	Z
0	0	0	0	0
0	1	1	1	0
1	0	1	1	1
1	1	0	0	1

$D = m(1,2) = Q \oplus X$

5.6.6 输出表达式

> Outputs' Equations

Solve the equations for the outputs

Present State	Input	Next State	FF Input	Output
Q	X	Q+	D	Z
0	0	0	0	0
0	1	1	1	0
1	0	1	1	1
1	1	0	0	1

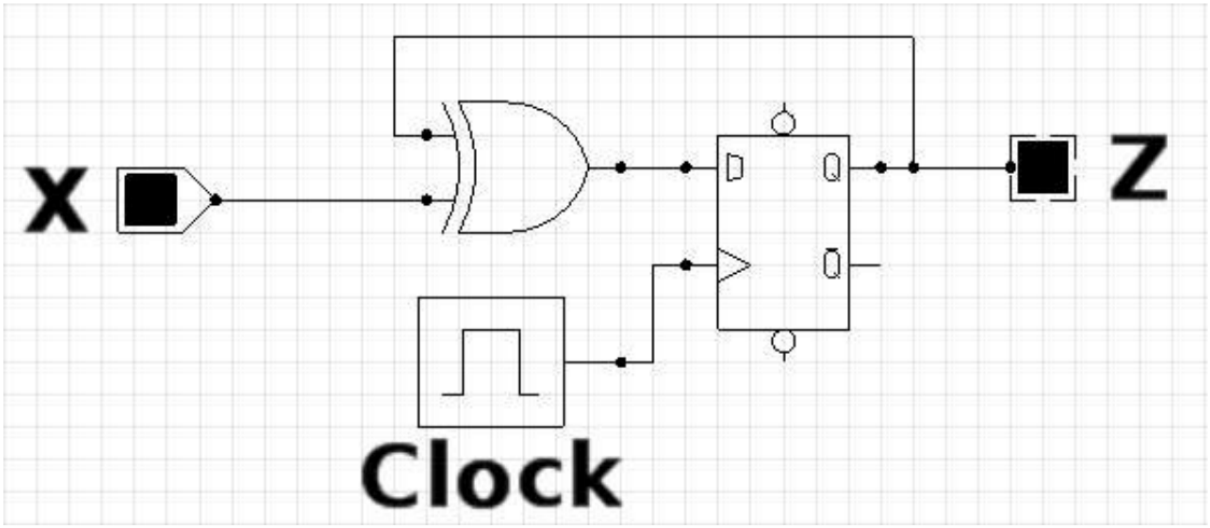
$Z = m(2,3) = Q$

5.6.7 实现电路

> Implement the circuit

$D = m(1,2) = Q \oplus X$

$Z = m(2,3) = Q$



Lec 6 寄存器与计数器

> Registers and Counters

Registers

Counters

- Asynchronous counter
- Synchronous counter

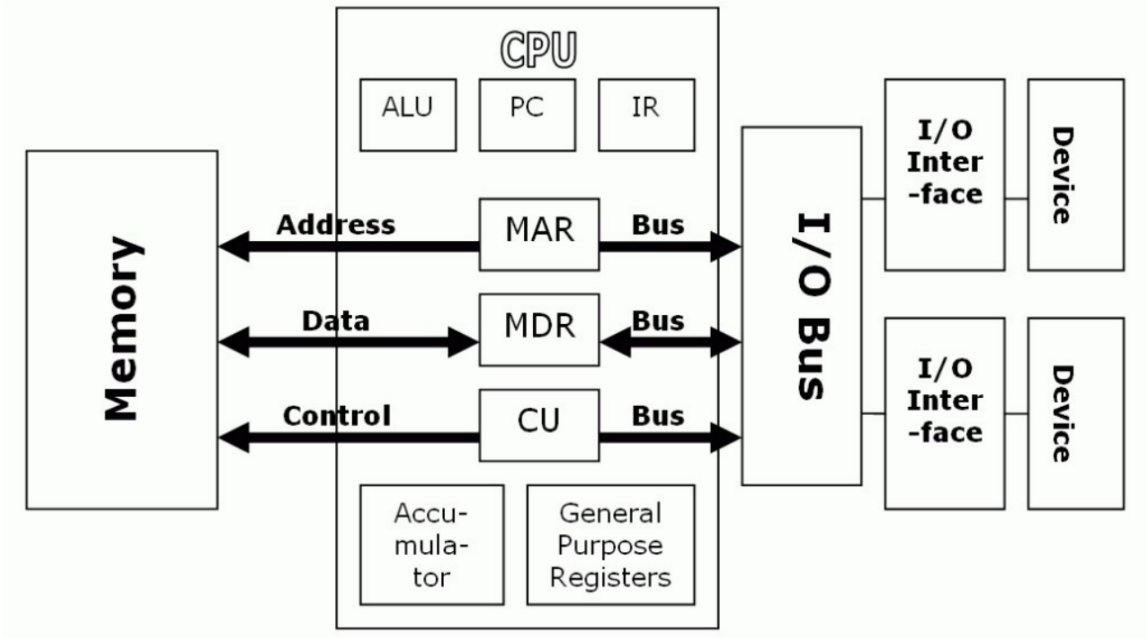
6.1 寄存器

> Registers

寄存器 $=$ 一组触发器.

- 每个触发器存 1 bit.
- n 位寄存器 $=n$ 个触发器，可存 n 位二进制信息.

此外，相比单纯的 FF，寄存器内部常集成组合逻辑门（Combinational Gates）以完成简单运算或控制.



6.1.1 分类

缩写	全称（英文）	中文功能说明
IR	Instruction Register	指令寄存器：存放当前正在执行的指令
SR	Status Register / Flag Register	状态寄存器：保存运算结果标志位 • Z (Zero) 结果为 0 • N (Negative) 结果为负 • V/O (Overflow) 溢出
AC	Accumulator Register	累加器：存放算术逻辑单元的运算结果
MAR	Memory Address Register	存储器地址寄存器：存放将要访问的内存地址
PC	Program Counter	程序计数器：保存下一条指令地址
Shift Register	Shift Register	移位寄存器：实现比特左/右移或串并转换

6.1.2 移位寄存器

> Shift Registers

Shifting binary information either to the right or to the left.

Arithmetic Shift：

- 左移一位：乘 2
- 右移一位：除 2

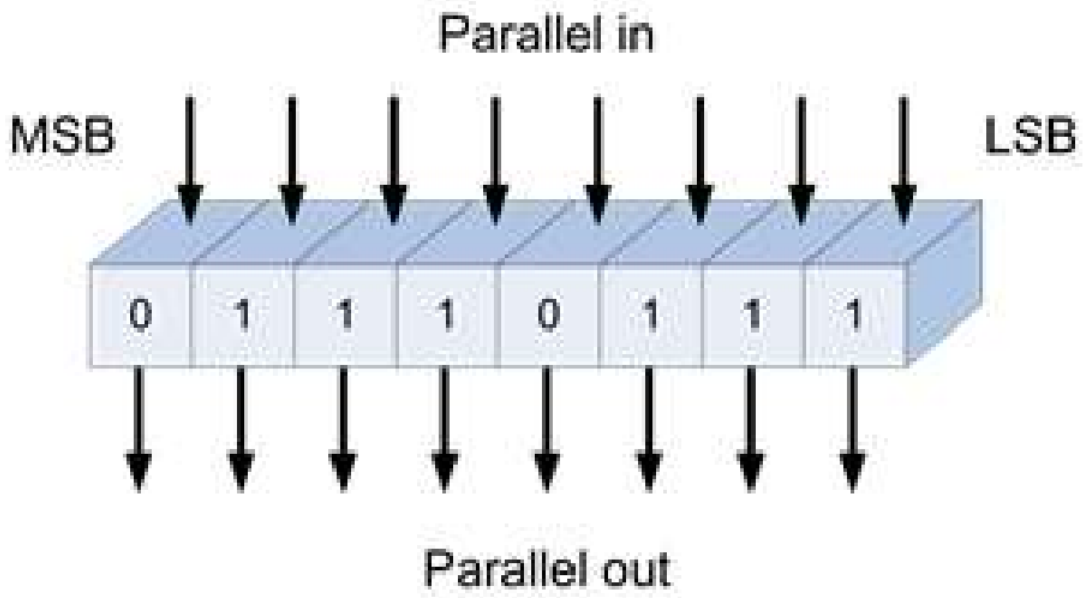
```
// a = 5(00000101)
unsigned char a = 5

// The result is 00000010
a >> 1;
```

> 注意右移可能把余数忽略掉.

移位寄存器可以实现串并转换.

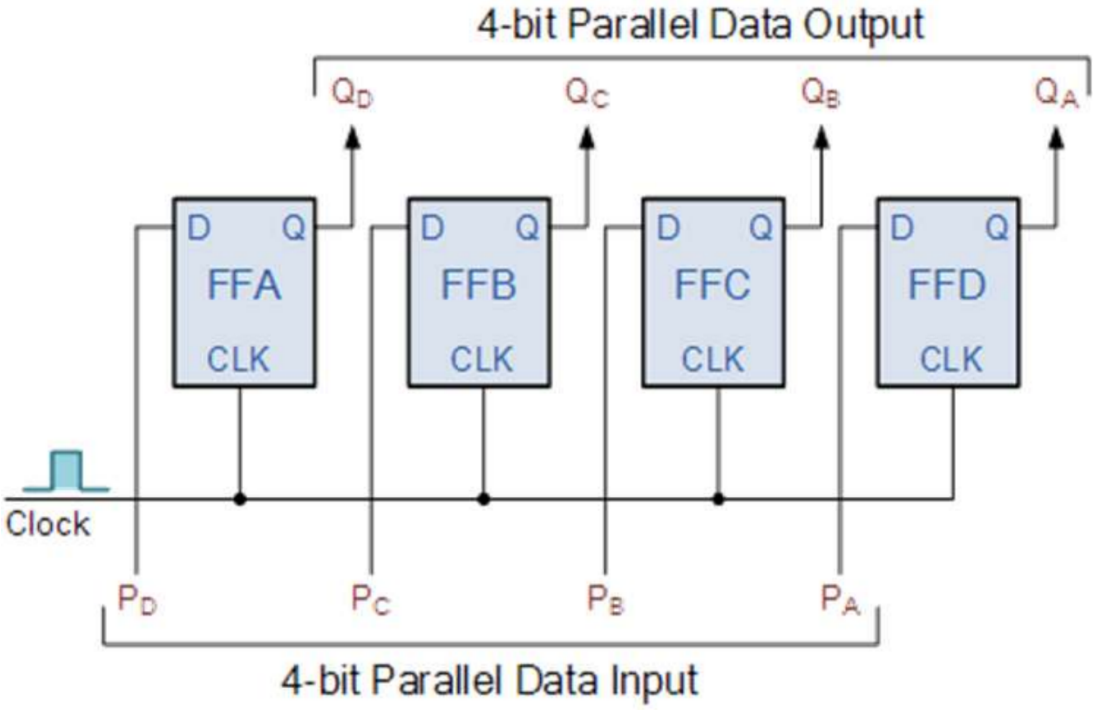
① PIPO

> Parallel-in to Parallel-out (PIPO)
>
> 并行输入，并行输出



所有位一次性写入、一次性读出；不用于移位操作，仅用于存储.

4-bit PIPO Shift Register



② SISO

> Serial-in to Serial-out (SISO)
>
> 串行输入，串行输出

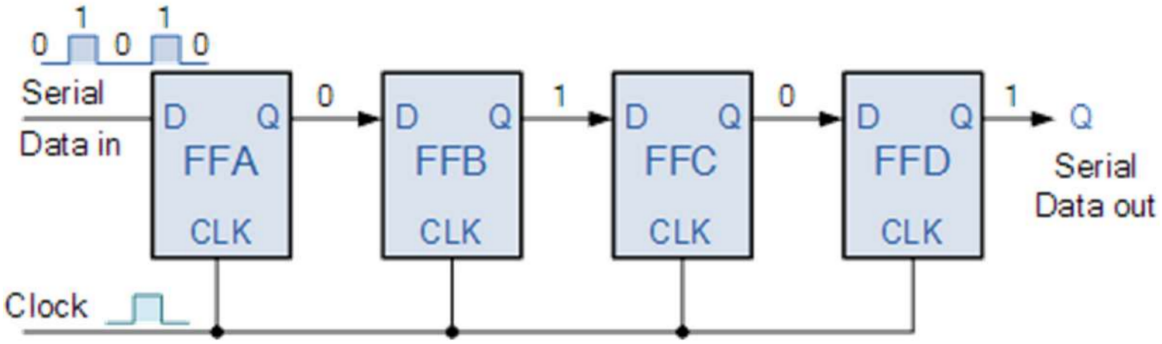
数据一位一位地输入，然后一位一位地输出.

常用于**数据延迟**或者**FIFO（队列）缓冲**.

每个时钟周期：

- 旧数据右移一位
- 新数据从左边（MSB）进入
- 最右边（LSB）的数据被推出（作为输出）.

4-bit SISO Shift Register



把 **SIPO（Serial-In Parallel-Out）移位寄存器**的**并行输出接口（多个 Q）变成只取最后一级的输出 Q**，最终实现了一个 **SISO（Serial-In Serial-Out）移位寄存器** 的功能.

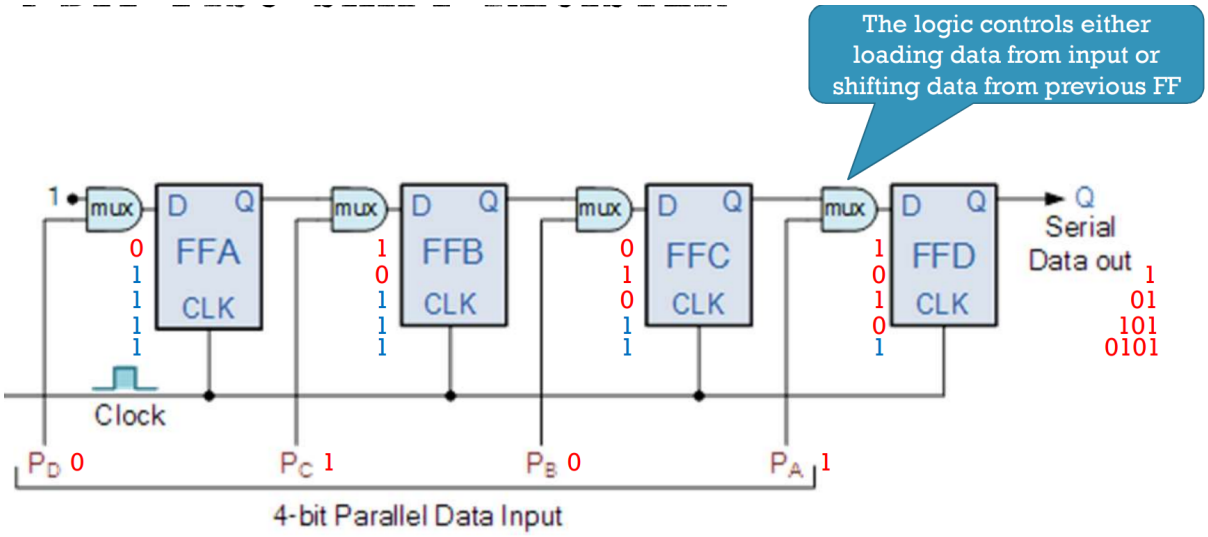
③ PISO

> Parallel-in to Serial-out (PISO)
>
> 并行输入，串行输出

一次性加载多个位，然后一位一位输出

用于将并行数据转换为串行数据（如串口通信）

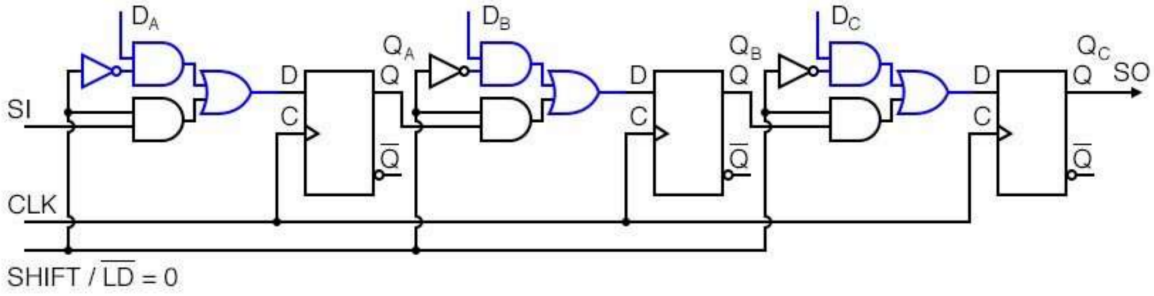
4-bit PISO Shift Register



运行分为两步，这张图没有画出 MUX 的选择控制线（左边两个输出二选一）.

逻辑是这样的：MUX 先调成 1，并行输入，然后调成 0，经过四个时钟脉冲把 4 位并行输入转换为串行输出.

3-bit PISO Shift Register 电路细节



Parallel-in/ serial-out shift register showing parallel load path

用 AND、OR、NOT 实现 MUX，用 $SHIFT / \overline{LD}$ 控制信号选择.

> 1 时移位，0 时加载数据.

注意串行移位 / 输出时需要补充最左边的 bit，因此有个 SI 输入.

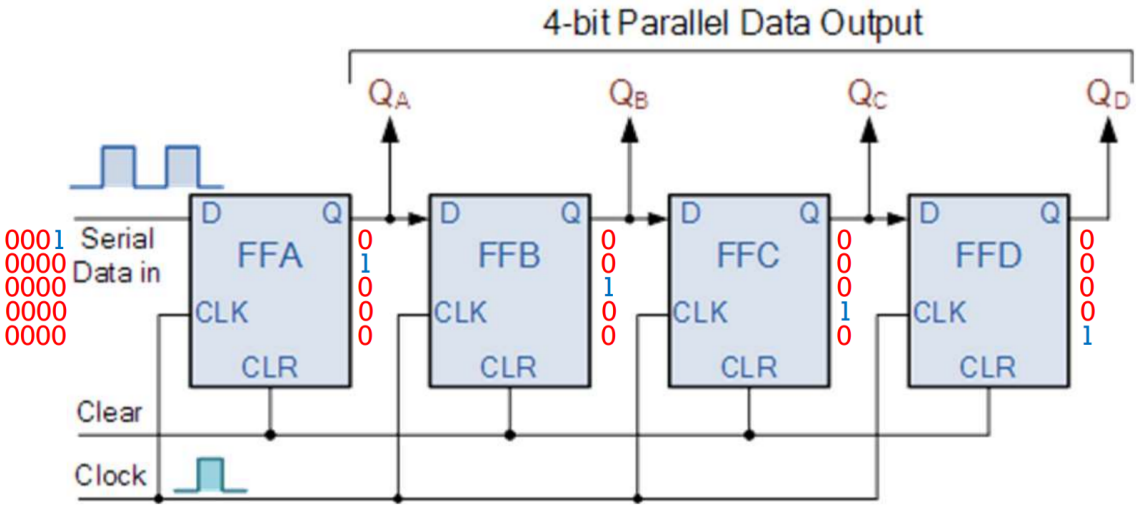
④ SIPO

> Serial-in to Parallel-out (SIPO)
>
> 串行输入，并行输出

一位一位输入，最后一次性输出多个位

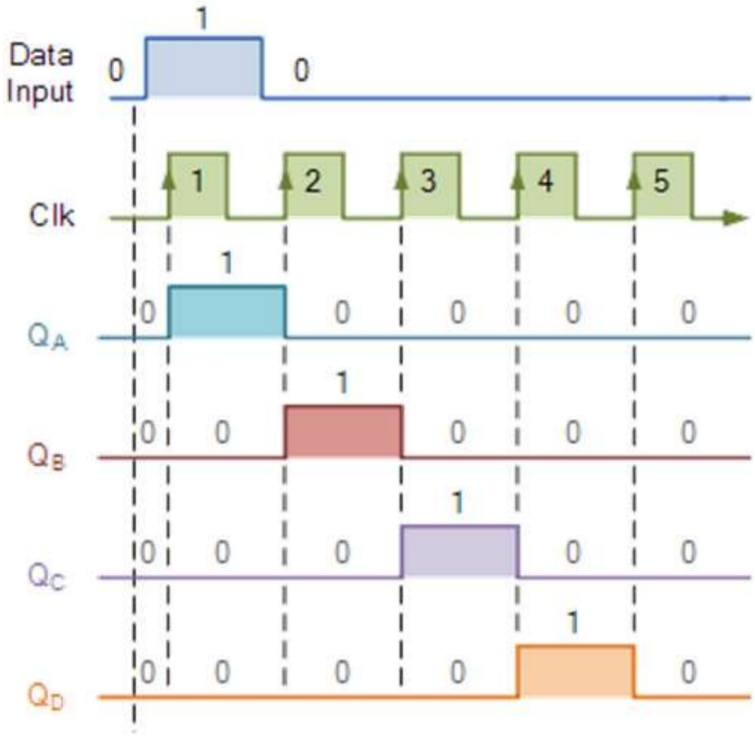
用于串行数据转并行（如接收端）

4-bit SIPO Shift Register

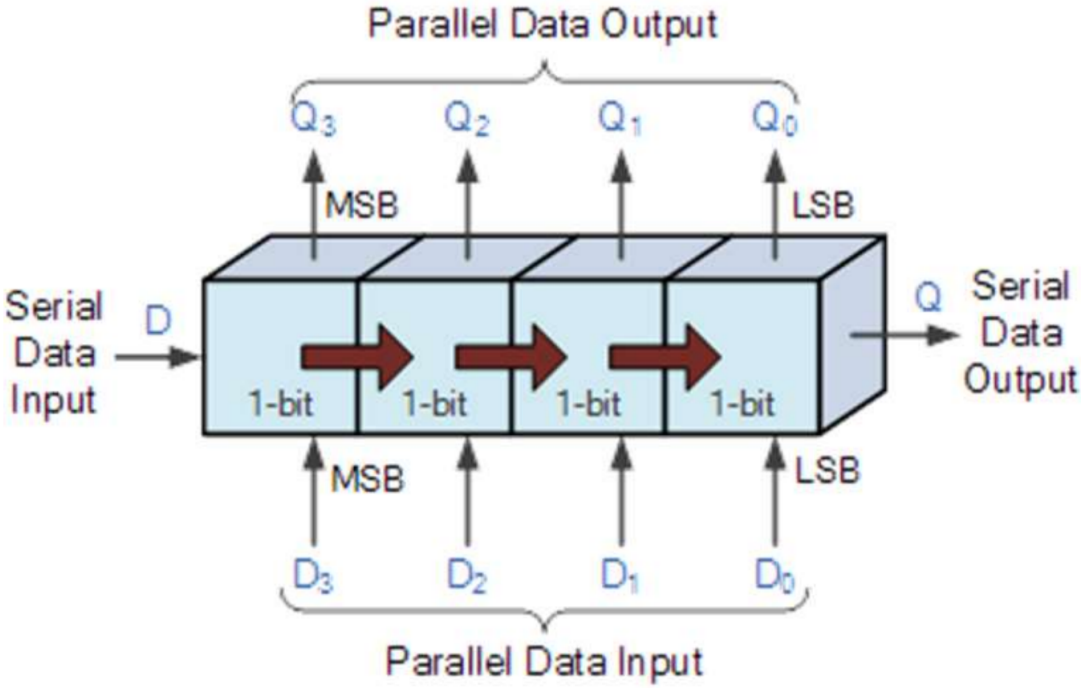


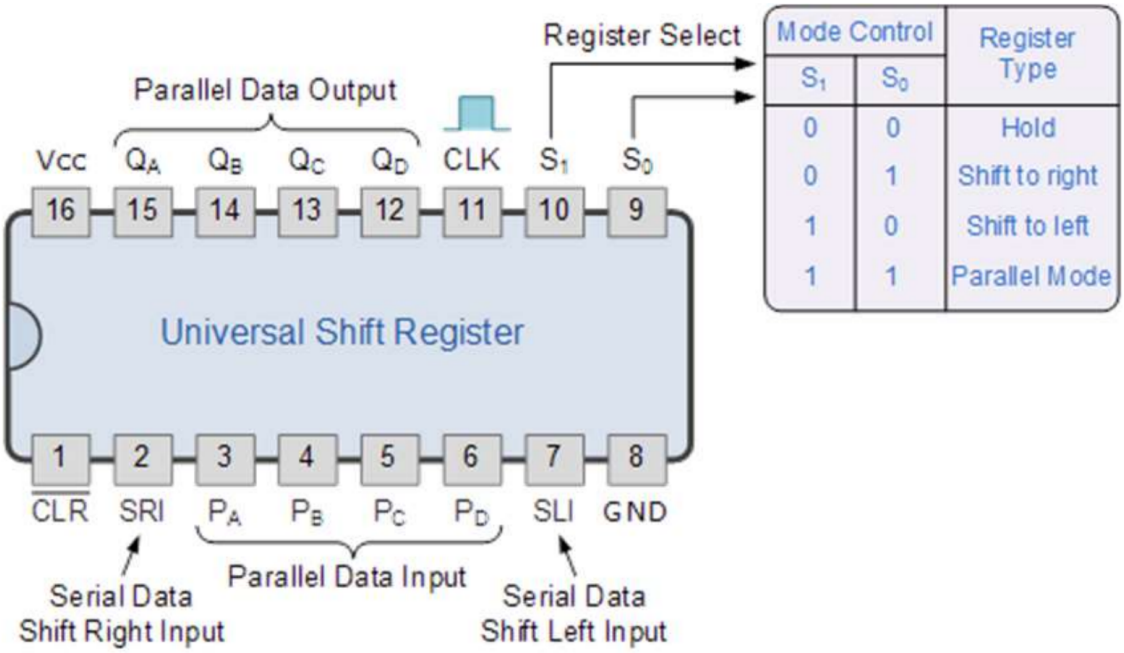
时钟周期	串行输入	QA	QB	QC	QD	状态（从左到右）
1	1	1	0	0	0	`0001`
2	0	0	1	0	0	`0010`
3	0	0	0	1	0	`0100`
4	0	0	0	0	1	`1000`
5	0	0	0	0	0	`0000`

D触发器在每一个时钟的上升沿（posedge）会将输入 `D` 的值复制到输出 `Q`，相当于所有 bit 右移一位，LSB 被原先的倒数第二位挤掉，MSB 由 Serial Data in 补充. 注意 QAQBQCQD 是作为 4 bits 同时（并行）输出的.



⑤ 通用型移位寄存器





6.2 计数器

> Counters

A counter is essentially a register that goes through a predetermined sequence of binary states

The gates in the counter are connected in such a way as to produce the prescribed sequence of states

本质：寄存器 + 特殊连线，使其经历预定的二进制状态序列.

6.2.1 MOD-8 计数器

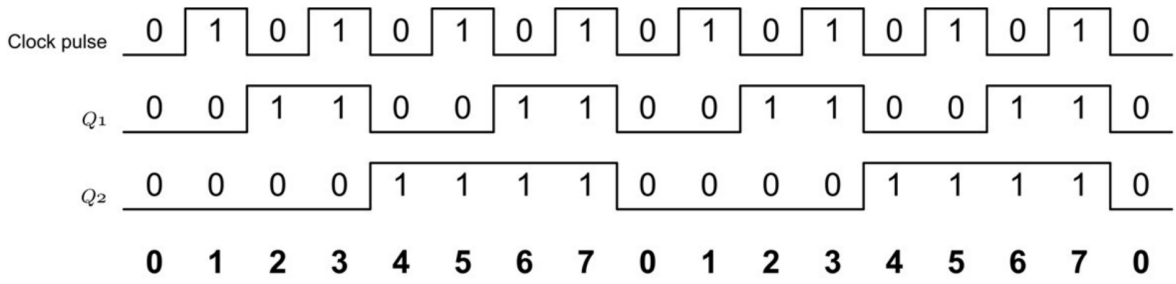
> MOD-8 Counter
>
> Counting from 0 to 7 and repeat again

DEC	COUNTER
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7

DEC	COUNTER
8	0
9	1
10	2
11	3
12	4
13	5
14	6
15	7

DEC	COUNTER
8k	0
8k+1	1
•	2
•	3
8k+n	4
•	5
•	6
8k+7	7

时序波形图



这图有点问题，应该忘记画 $Q0$ 了. 注意时钟本身一般不作为一个状态 bit.

6.2.2 应用

Timing（定时）

- From higher frequency to lower frequency
- e.g. alarm clocks, stopwatches, alerts, etc.

Sequencing（顺序控制）

- Devices ON/OFF at specific order
- e.g. digital switches and controls

Counting（计数）

- Measuring the data flow
- e.g. traffic on highway

6.2.3 异步计数器

> Ripple Couters，也叫做纹波计数器或涟波计数器.

The clock signal is NOT connected to each flip-flop directly

Ripple counters are also known as **asynchronous counter**

Instead, a flip-flop output transition serves as a source for triggering other flip-flop

特点：**只有第一个 FF 接时钟**，其余级由前级输出翻转沿触发.

优点：电路简单、门少.

缺点：翻转传播延迟逐级累积，导致高位输出延后，限制最高工作频率.

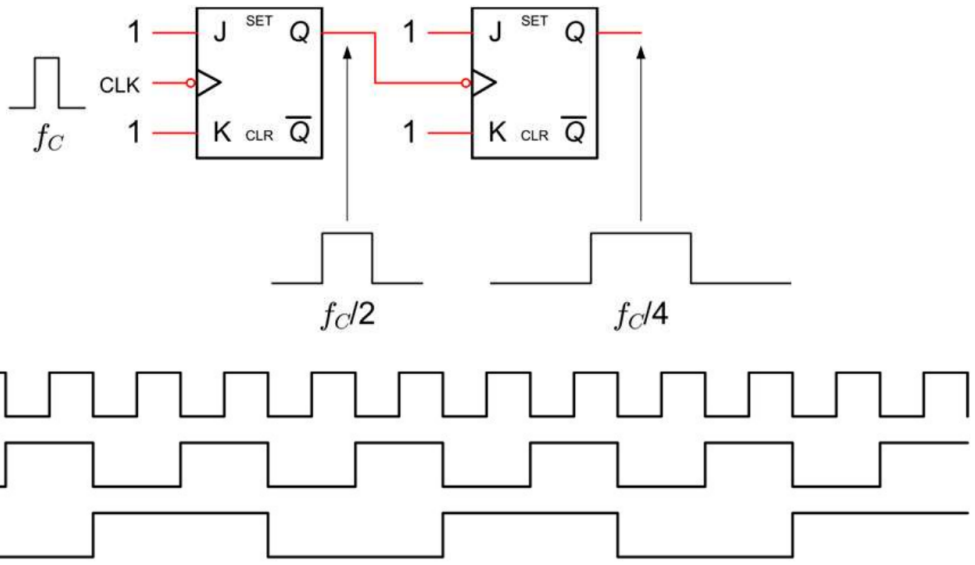
Ripple counters are not directly connected to the clock

Instead, it rely on the propagation of the outputs from the previous FFs

Time delay will be accumulated along the chain of FFs

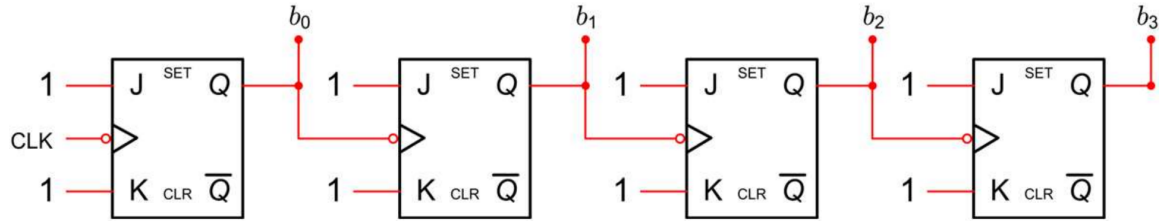
> 注意："延迟累积" **不是**指每升高一位 "频率减半"，而是指**由于时钟信号不同步带来的** "传播延迟（propagation delay）"

通过级联两个 JK-FF 实现 MOD-4 Counter

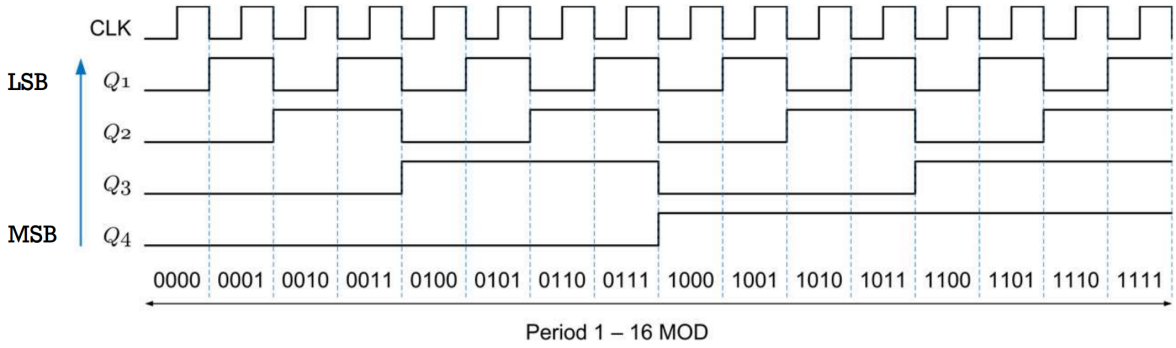


> 这里左边是低位，右边是高位.

通过级联四个 JK-FF 实现 MOD-16 Counter



时序波形图



6.2.4 N 分频计数器

> Divide-by-N Counter

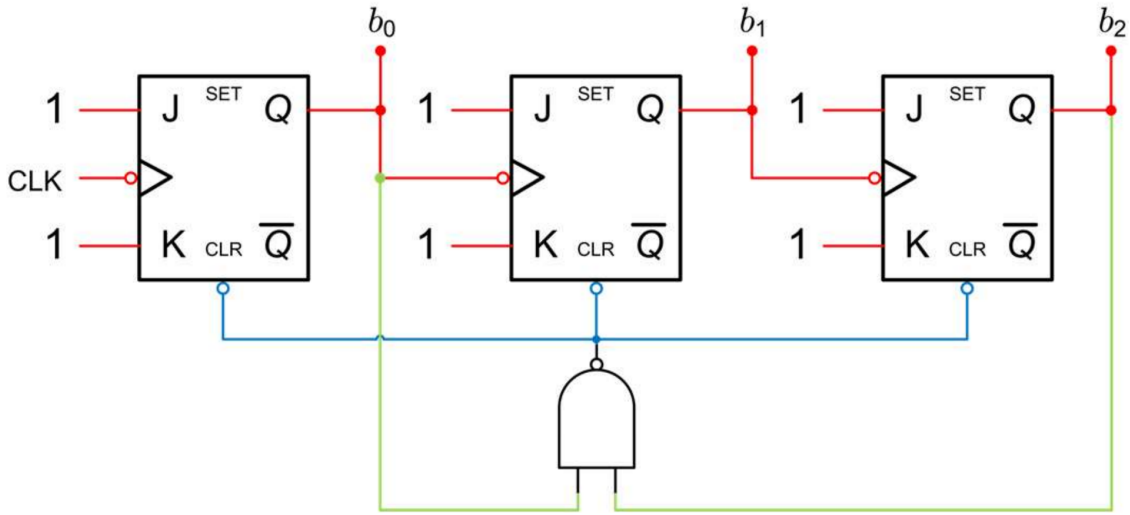
即 MOD-N 计数器，在异步计数器的基础上加一个归零.

> 这里不要求 $N = 2^n$.

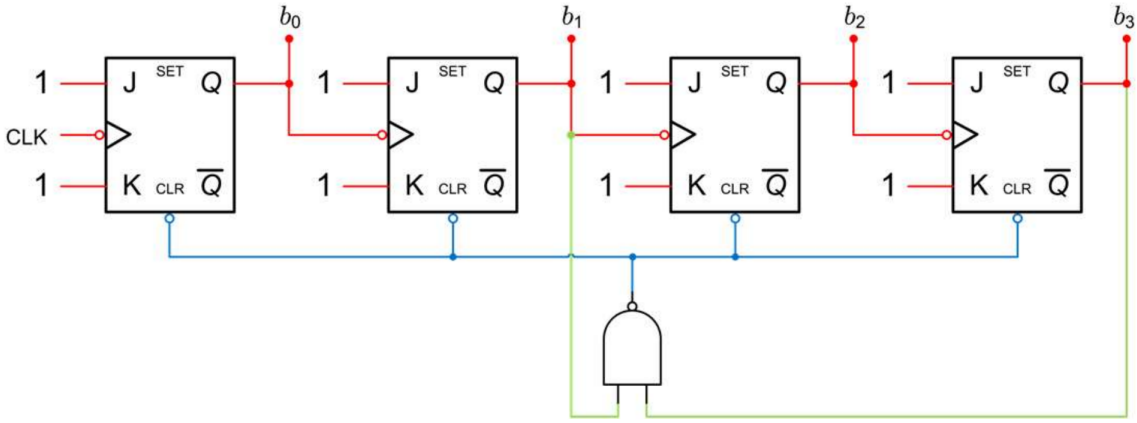
使用 $\lceil \log_2 N \rceil$ 个 FFs 表示状态.

使用 CLR，当状态达到 N 时（从 0 数到 $N-1$）自动清零回到起点.

例：Divide-by-5 Counter



例：Divide-by-10 Counter



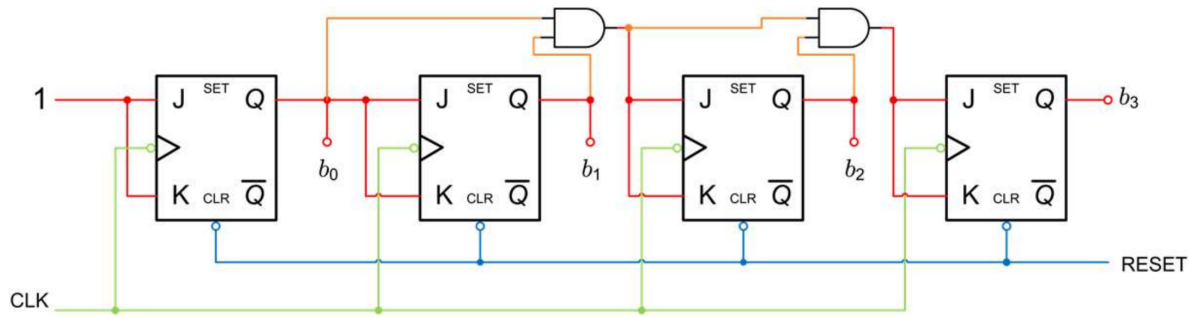
6.2.5 同步计数器

> Synchronous Counter

To avoid the accumulation of propagation delays, synchronous counter can be used

In synchronous counter, every FF is directly connected to the same clock input

例：MOD-16 Synchronous Counter



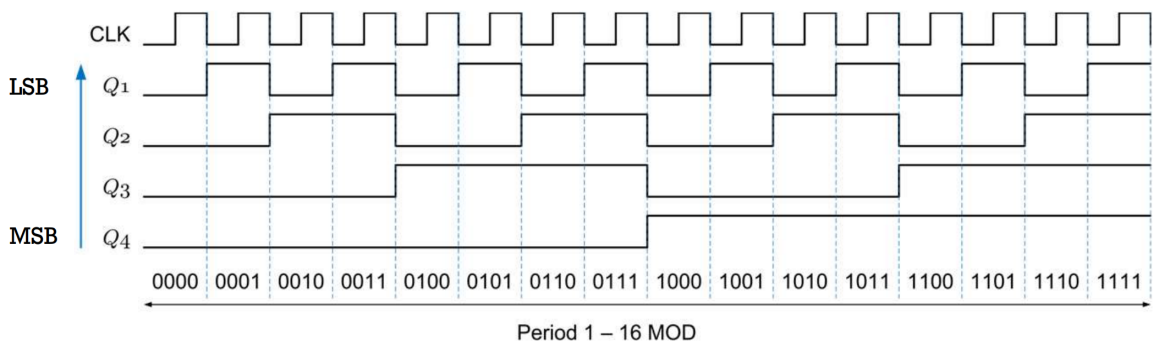
All JK inputs are identically connected, except the first pair is HIGH

All CLR inputs are connected to external RESET

The outputs of two successive FFs are ANDed to become the inputs to the next stage, but NOT connected to the CLK

> 注意，每个 JK 触发器的输入 J 和 K 都是由前面所有输出位的 AND 结果控制的，因为要前面全部为 1 才能在下一步翻转自己.

时序波形图



6.2.6 环形计数器

> Ring Counters

把移位寄存器首尾相连，且仅有一个 1 在寄存器中循环.

Similar to synchronous counters (all FFs are connected to a common clock), but their outputs are not in TRUE binary but a repetitive sequence of digital output levels – a way for generation of timing sequence

Ring Shift Counter 是一种基于移位寄存器的计数器，它的输出不是标准的二进制数，而是一个**重复的"单 1"序列**，用于**生成时序控制信号**.

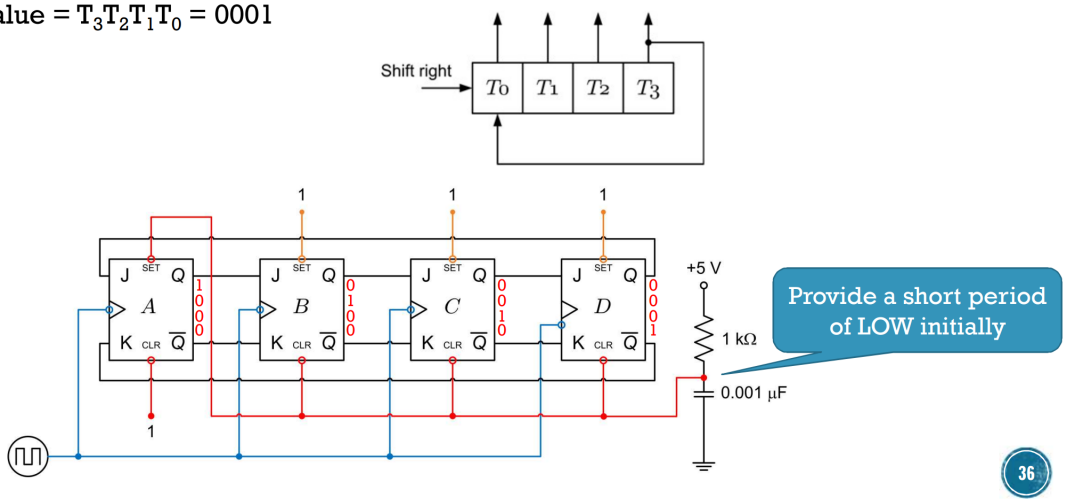
A ring shift counter is a circular shift register with

- Only one FF set at a particular time and others are cleared
- One bit is shifted from one FF to the other to produce the sequence of timing signals
- The output of the last FF is fed back to the first FF

类似一个时钟，通过唯一的 1 在环形计数器中的位置来判定当前时间.

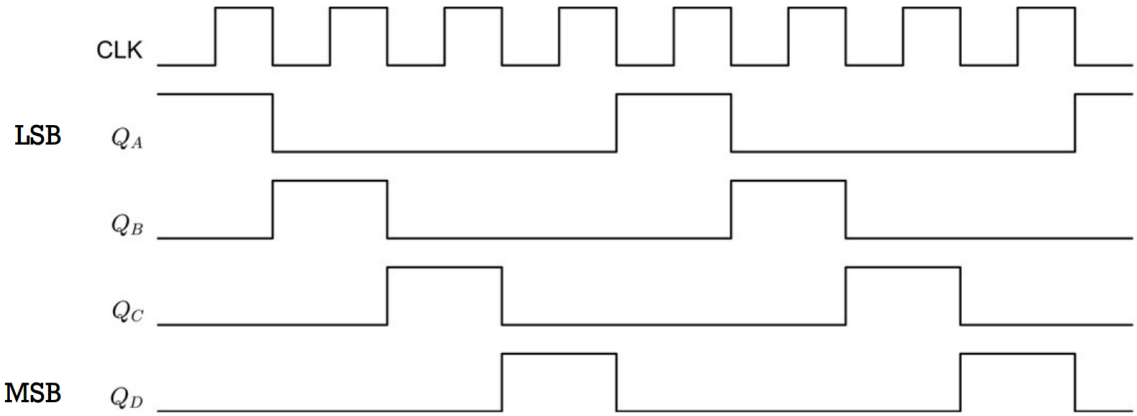
例：4-bit Ring Shift Counter

- Initial value = $T_3T_2T_1T_0$ = 0001



> 右边是上电复位，会在电路刚通电时产生一个短暂的低电平脉冲. 该脉冲触发 A 的 SET 和其他的 CLR，产生一个 1. 注意 A 的 CLR 是高电平，即不清零.
>
> 注意，JK-FF 本身是上升沿触发，但是加了个反向（很小），所以这个是下降沿触发.

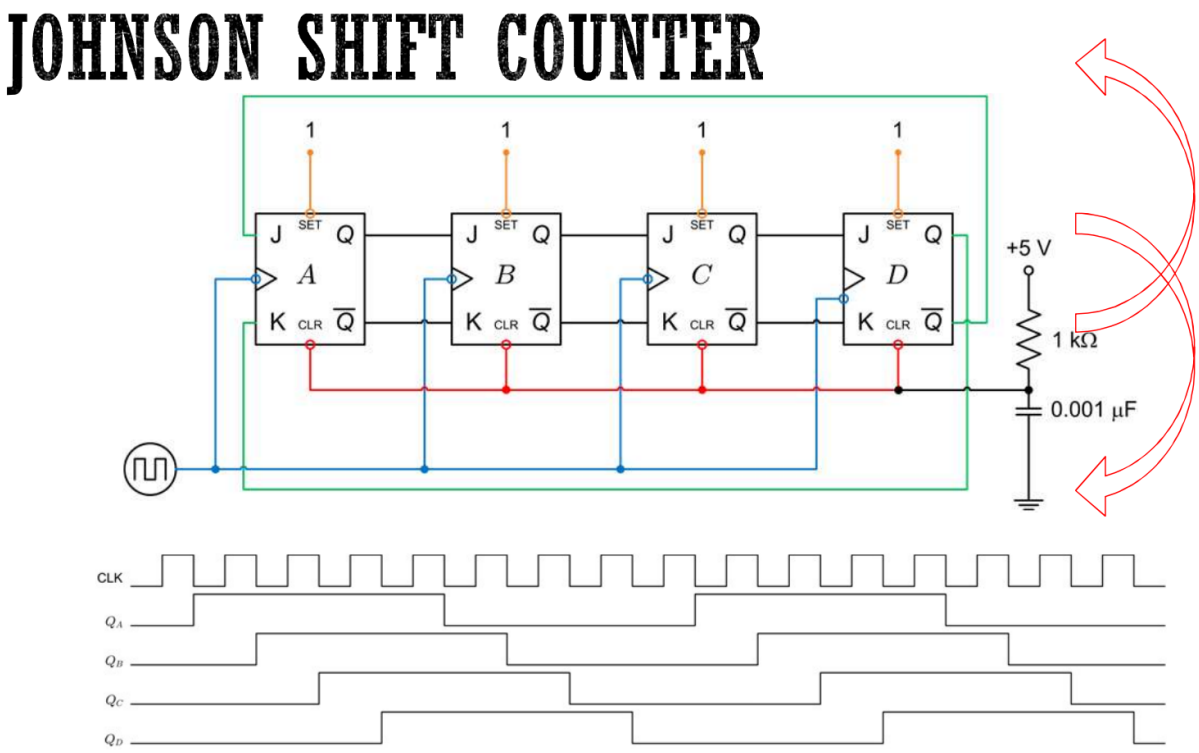
时序波形图



6.2.7 约翰逊计数器

> Johnson Shift Counter
>
> 也叫 Twisted Ring Counter 或 Switch-Tail Ring Counter

一种改进版环形计数器，不同点在于最后一个触发器的反向输出送回第一个触发器（交叉反馈）.



> 下降沿触发.

0000-1000-1100-1110-1111-0111-0011-0001-0000-1000...

Lec 7 内存与存储器

> Memory and Storage

Memory

Types of Memory

Memory Expansions

7.1 存储器

> Memory

One of the major features of any digital or computer systems is that memory capability is available

With different types of memory available in a system, more data can be processed in a faster manner

Types of memory circuits we have learnt

- Flip-flop
- Register

7.1.1 分类

> Classifications of Memory

按性质：Volatile（易失性） or non-volatile

Based on the nature of the memory

Stored values will be erased when the volatile memory is powered off

> 断电后数据丢失.
>
> 易失性如：RAM（随机存储器）
>
> 非易失性如：ROM，Flash，硬盘，SSD

按材料：Semiconductor（半导体） or magnetic（磁存储）

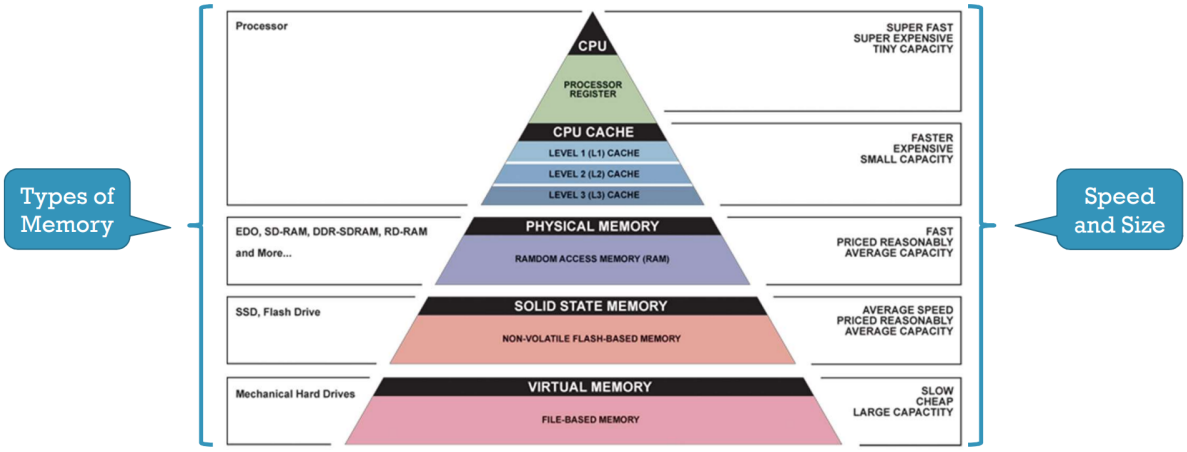
Based on the physical materials of the memory

> 半导体：RAM，ROM，Flash
>
> 磁：硬盘（HDD），磁带，磁鼓

按架构层次：Primary（主存） or secondary（辅存）

Based on the architecture of the memory system

> 主存：CPU 可直接访问. 如：RAM，Cache，寄存器
>
> 辅存：用于长期存储，访问较慢，如：硬盘，SSD，U 盘



从上到下，速度逐渐降低，容量逐渐增大，成本逐渐降低.

层级	类型	示例	特性
🔼 顶部	**CPU Register**	Processor Register	**最快**，最贵，容量极小
	CPU Cache	L1, L2, L3 Cache	**非常快**，贵，容量小
	Physical Memory	RAM（如 DDR、SDRAM）	快速，价格适中，容量中等
	Solid State Memory	SSD, Flash Drive	中速，性价比高，容量大
🔽 底部	**Virtual Memory**	HDD, 文件交换分区	最慢，最便宜，容量最大

7.1.2 RAM

> 随机存取存储器，Random Access Memory

RAM 是一种用于**临时存储数据和程序指令**的存储器，它允许**随机访问**任意位置的数据.

RAM is used for temporary storage of data and program instructions in microprocessor-based systems

Random Access means the user can access data at any location within the entire memory device randomly

Types of RAM

- Static RAM (SRAM)
- Dynamics RAM (DRAM)

特性	SRAM	DRAM
存储结构	6 个晶体管（flip-flop）	1 个晶体管 + 1 个电容器
数据保持方式	只要有电就保持	需要周期性刷新
速度	快	较慢
功耗	低（静态时）	高（因刷新）
面积/成本	大 / 贵	小 / 便宜

特性	SRAM	DRAM
应用	CPU Cache	主内存（如 DDR RAM）

SRAM 快但贵，适合做缓存；DRAM 慢但便宜，适合做主内存.

7.1.3 ROM

> 只读存储器，Read Only Memory

ROM 是一种非易失性（non-volatile）的存储器，支持随机访问（random access），常用于 BIOS、固件等领域.

Capable of random access

Non-volatile

- Do not lose their memory contents when power is removed
- e.g. BIOS (Basic Input Output System)
- e.g. Firmware in some embedded systems

Mask ROM

- Fabricated with the desired data permanently stored
- Unique mask is required in the fabrication

User-Programmable ROM (PROM)

> Avoid the high one-time cost of producing a custom mask (Mask ROM)
>
> **用户可编程 ROM**，出厂时为空，一旦写入（通过“烧录”），就无法更改.

EPROM (Erasable PROM)

> Use UV light source to erase the stored data
>
> 一种可以**擦除并重新编程**的 PROM
>
> 擦除方式：**使用紫外线（UV light）**，通过芯片顶部的透明窗口照射
>
> **可以反复擦写**，但过程慢、不方便
>
> 芯片图中左边的有透明圆窗的芯片，就是 EPROM

EEPROM or E2PROM (Electrically Erasable PROM)

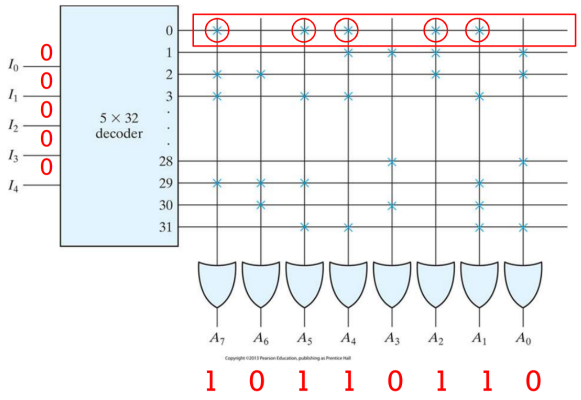
> Use high voltage to erase the chip
>
> 使用**电信号**擦除和写入数据，而不是紫外线
>
> 可以逐字节擦写（不像 EPROM 要整块擦除）
>
> 擦写过程更快、更方便
>
> 应用举例：电视机调谐器（TV Tuner）、BIOS 设置存储等
>
> 又称 **E²PROM**

- Truth Table (Partial)

Table 7.3
ROM Truth Table (Partial)

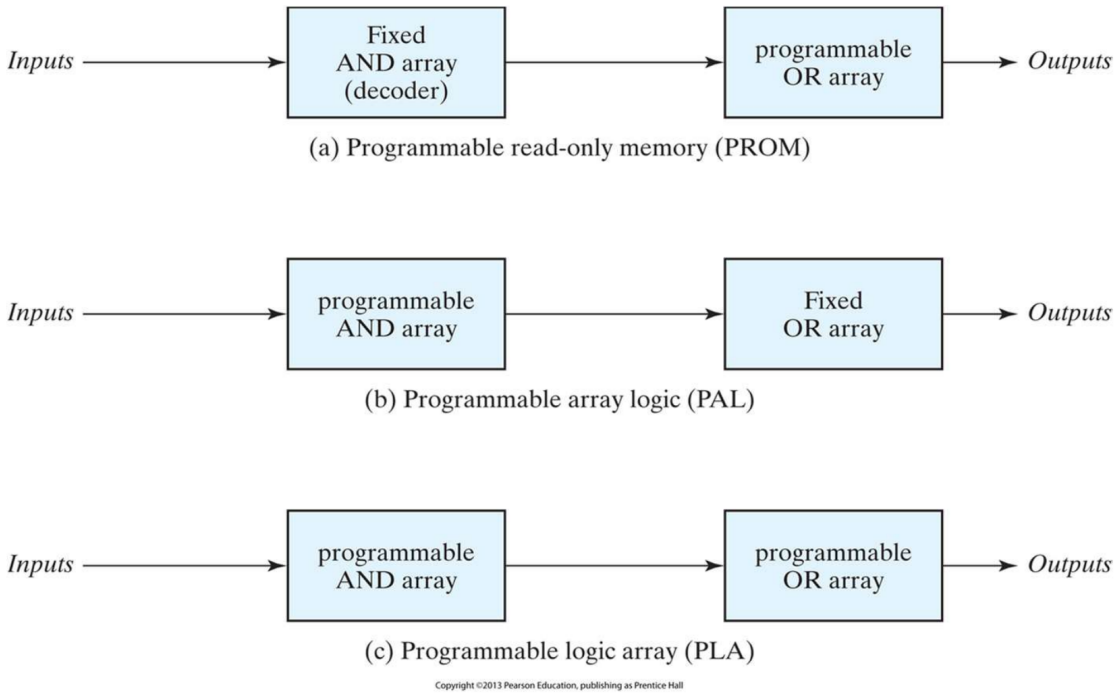
Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
⋮								⋮				
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

- Internal Logic

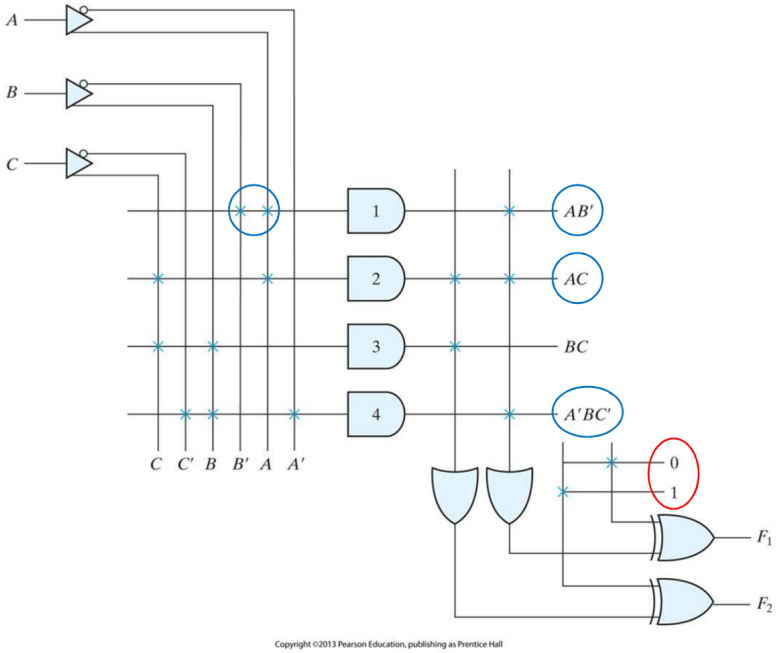


7.1.4 PLD

> 可编程逻辑器件（PLD，Programmable Logic Device）



PLA（可编程逻辑阵列）



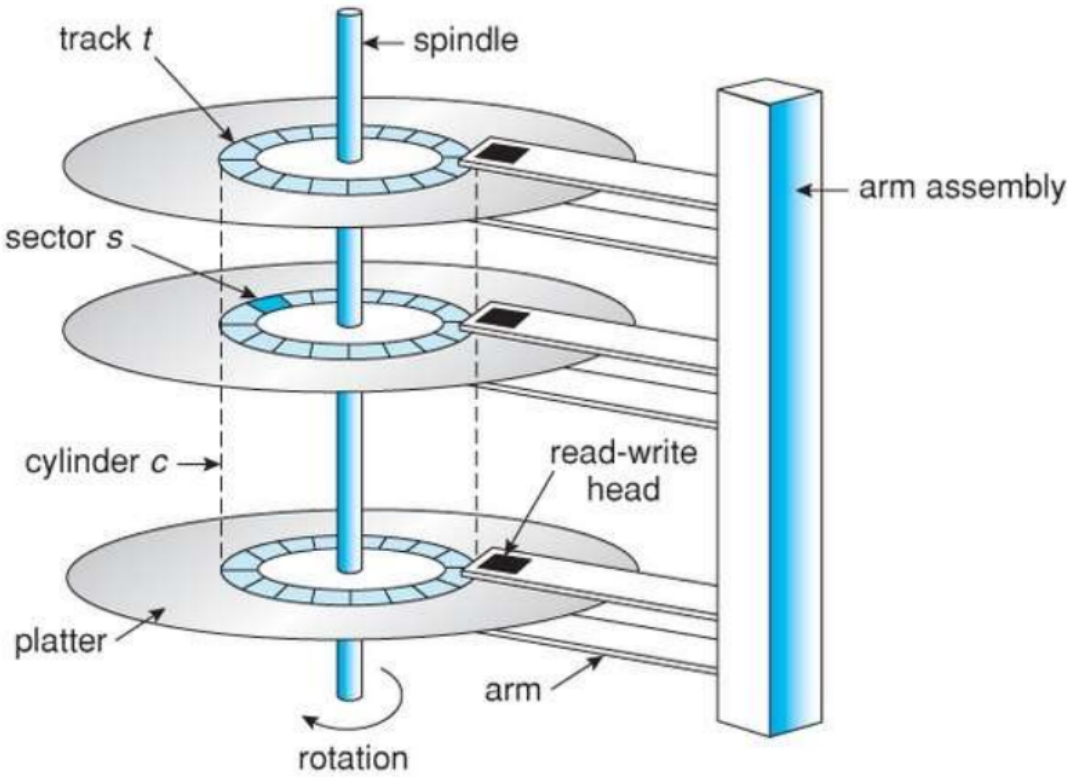
Exclusive-OR gate

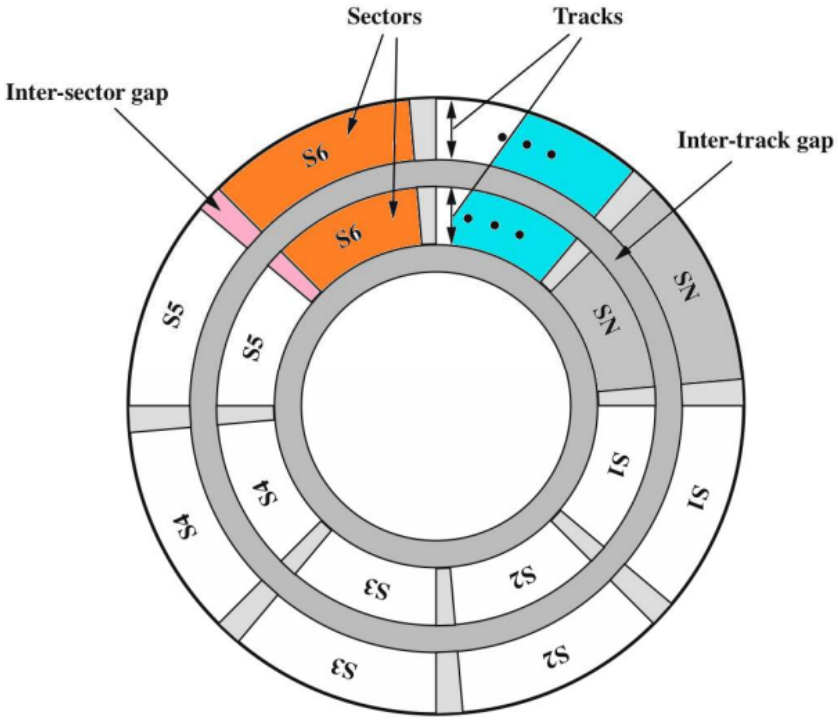
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

F1 = AB'+AC+A'BC'
F2' = AC+BC

7.1.5 磁盘

> Magnetic Disk





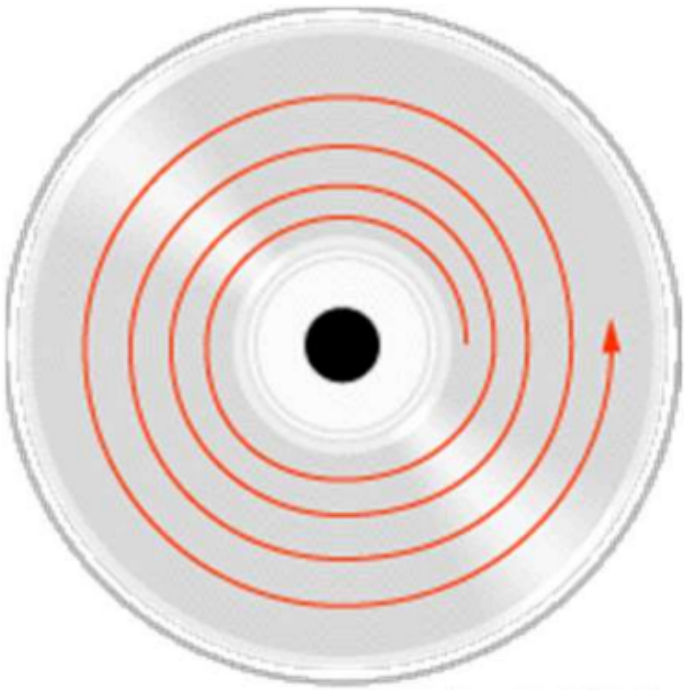
1s and 0s are represented on the magnetic medium as a tiny northsouth or south-north magnet

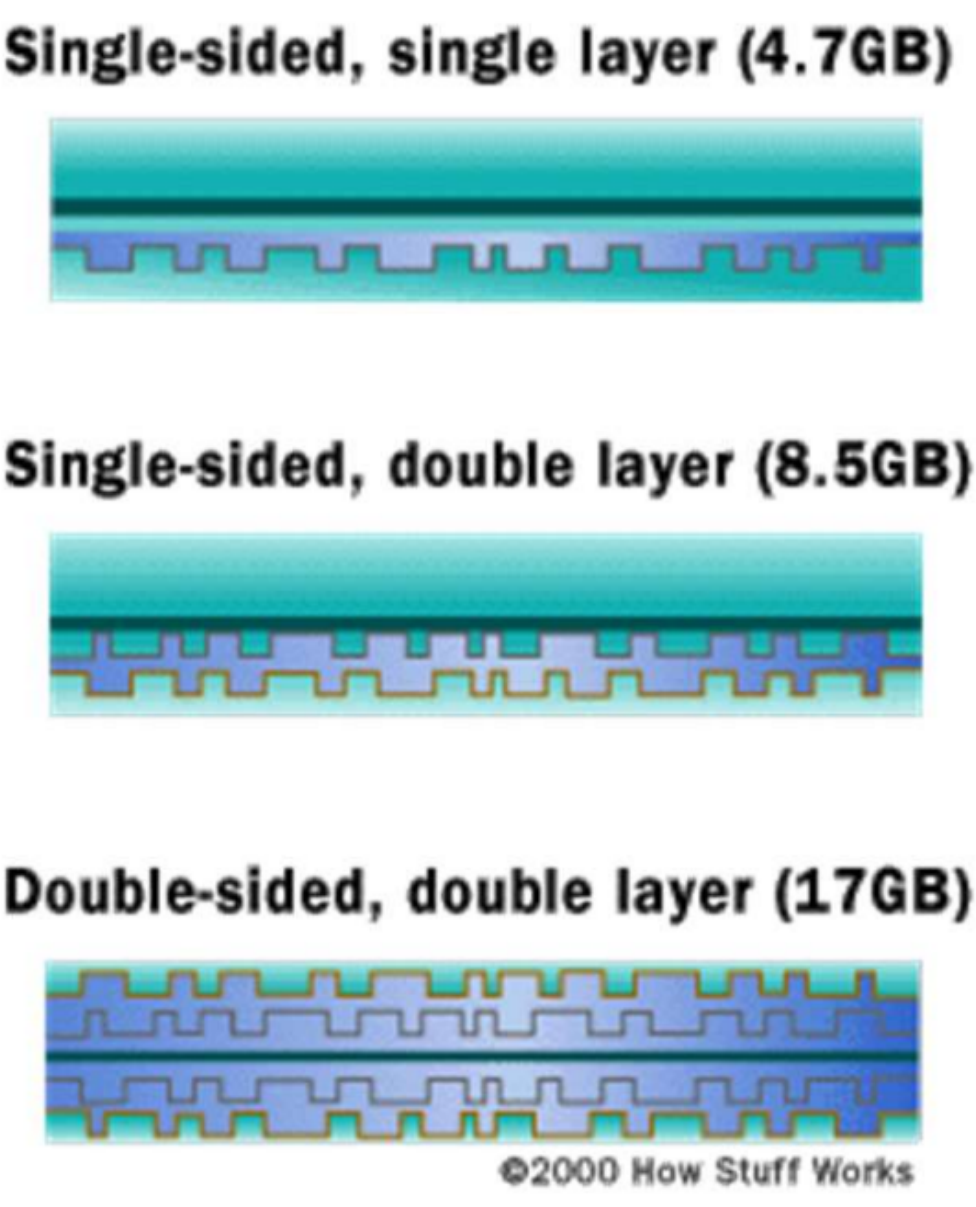
Write: Magnetize the medium into a particular orientation

Read: Detect the direction of the induced voltage when the tiny magnet passes the read/write head

7.1.6 光盘

> Optical Disk





Aluminium alloy coating on a rigid polycarbonate wafer

1s and 0s are represented on the medium as existence of an indentation (pits) or absence of the indentation (land)

Write

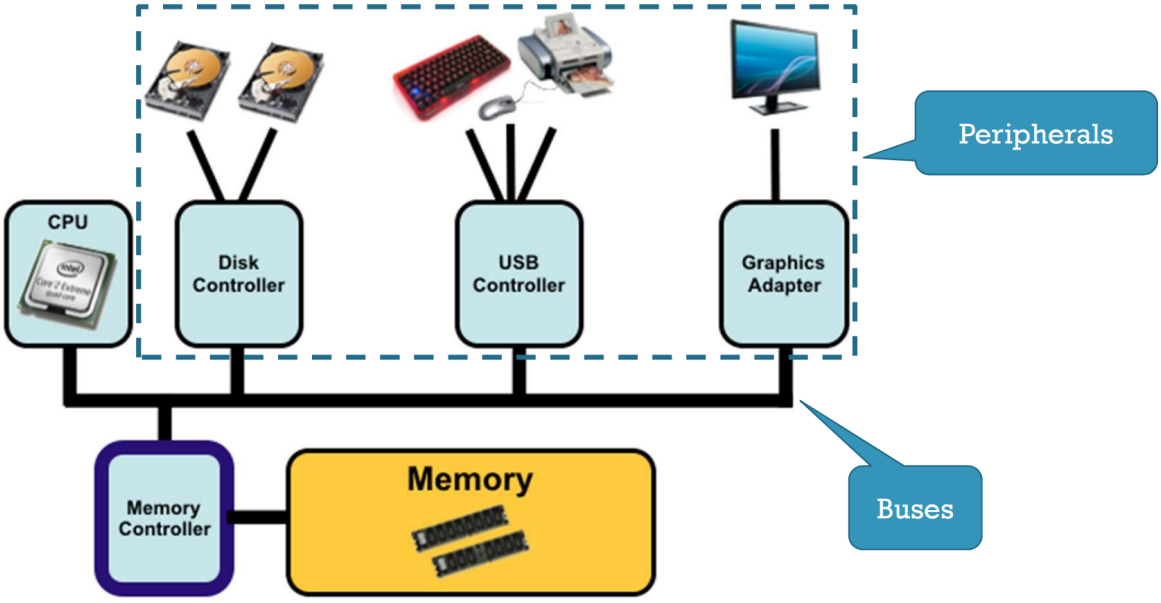
Print the required data during mass manufacturing. (e.g. Music CD)

Use Laser beam to burn the pits on blank CD (e.g. CD ROM)

> 光盘不能复位.

7.2 计算机系统结构

> Computer System Architecture



CPU（中央处理器）：计算机的“大脑”

> 负责执行指令、控制数据流动、运算等工作

Memory Controller + Memory：内存控制器 + 主存

> 内存控制器负责协调 CPU 与内存之间的数据交换，下方黄色模块是主内存（RAM），存放当前运行程序和数据

Buses：总线

> 所有模块通过 **总线（Bus）** 互联，总线是一种共享的通信通道，用于传输数据、地址和控制信号，图中粗黑线代表系统总线.

Peripherals Controllers：外设控制器

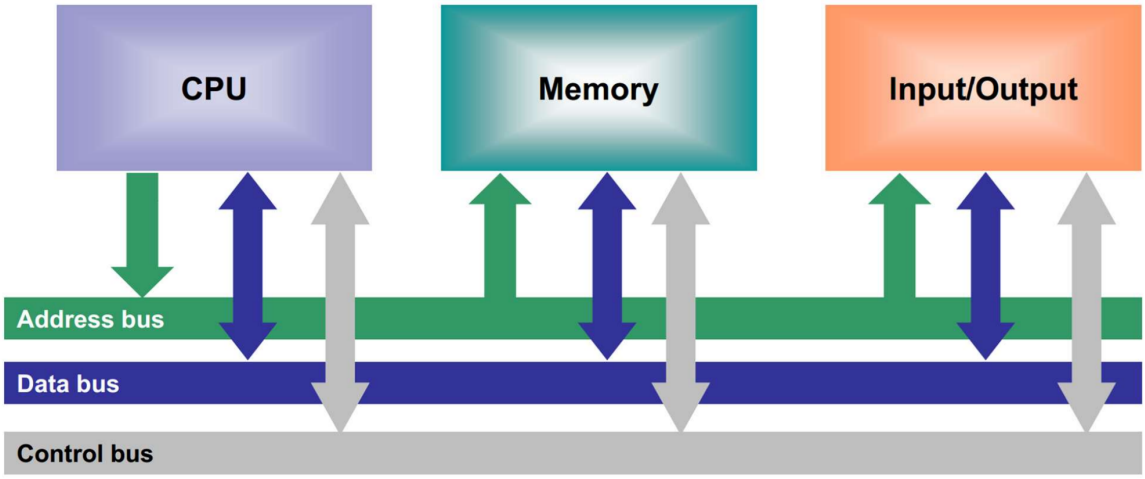
> 这些控制器连接外设，统一通过总线与 CPU 通信.

控制器	连接的外设	说明
Disk Controller	硬盘	管理磁盘数据的读写
USB Controller	鼠标、键盘、打印机等	管理 USB 接口设备
Graphics Adapter	显示器（Monitor）	管理图形输出和显示

外设控制器是硬件，驱动程序是软件.

7.3 总线

> Bus Architecture



总线名称	功能
Address Bus（地址总线）	传输地址，指出数据应从哪里读/写（单向）
Data Bus（数据总线）	实际传输数据（双向）
Control Bus（控制总线）	传输控制信号，如读/写命令、时钟等（双向）

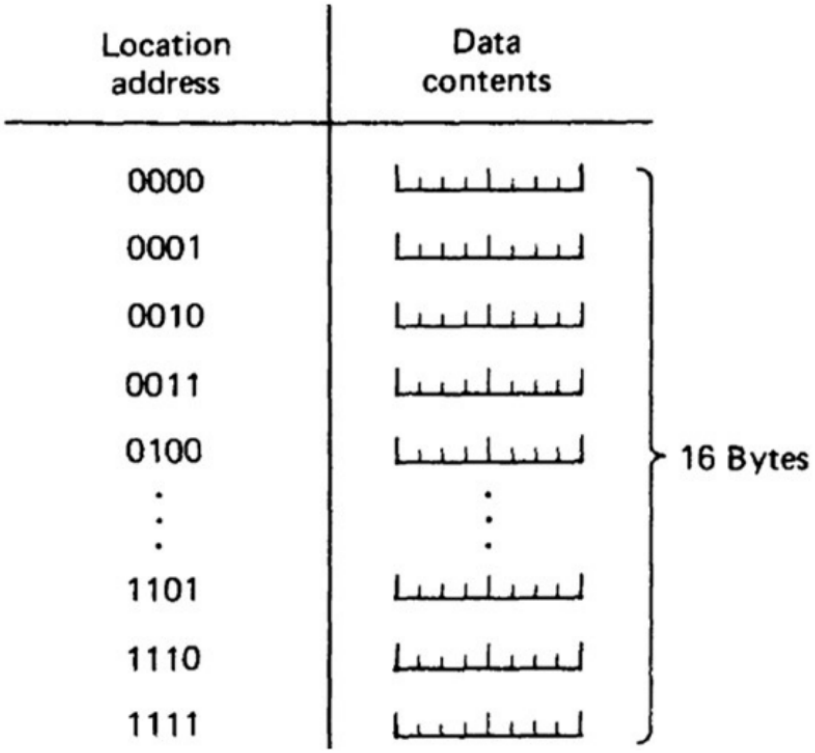
7.4 内存容量

> Memory Size

8 个开关，每个开关状态用 1 位 bit 表示，共 8 bits 表示完整状态.

每小时记录一次状态，从 7 点到 22 点，一共 16 次.

记录每天刷新，只需存储一天的数据.



需要多少内存来存储这些信息?

$$16 \times 8 \text{ bits} = 16 \text{ bytes}$$

Total capacity of the memory is the number of memory words times the data size.

整个地址空间最大容量：$2^n \times m$ bits. 其中：

- n 是 address bits，地址位数/地址线条数.
- m 是 Data bits per record，字长/每个数据的位数/数据线条数.
- 2^n：存储单元最大可用数量.

这里 16 条记录刚好用完，有的时候不一定用完，2^n 用实际记录条数代替.

7.4.1 存储单元

memory locations / memory words：存储单元，一条记录对应一个存储单元. 只需要知道地址位数就可以计算存储单元数量，不需要知道每个单元要存入多宽的数据.

The number of memory words depends on the size of the address bus

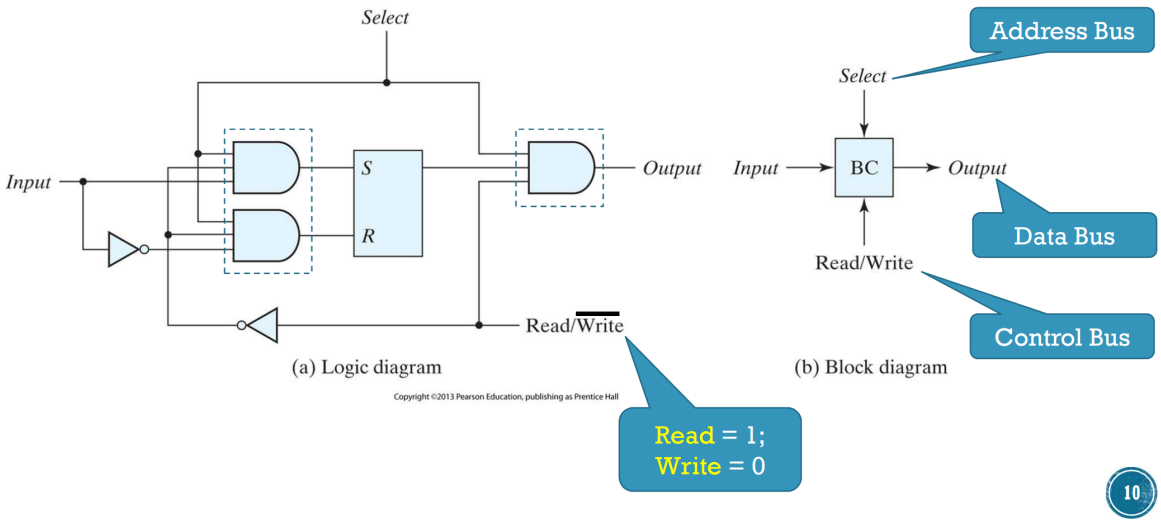
n-bit address provides 2^n memory locations

> 1K：$2^{10} = 1024$.

① 二进制存储单元

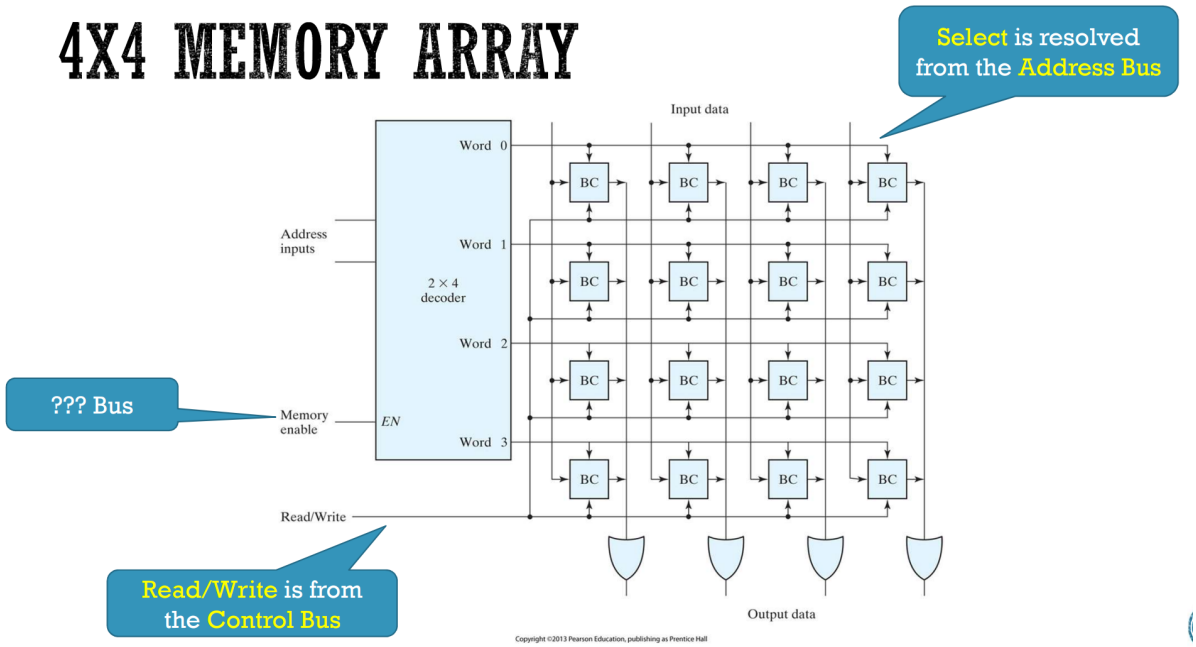
> Binary Memory Cell
>
> 存储 1 bit 的基本单元.

一个使用 RS Latch 构成的基本存储单元.



② 4×4 **内存阵列**

> 4×4 Memory Array



> EN 来自控制总线.

选中一行，一次读/写 4 bits.

注意，这里只有 4 个 memory locations，因为每行的 4 个 bits 不能分开读写，属于同一个单元里的数据. 每个地址对应一个 4-bit memory word.

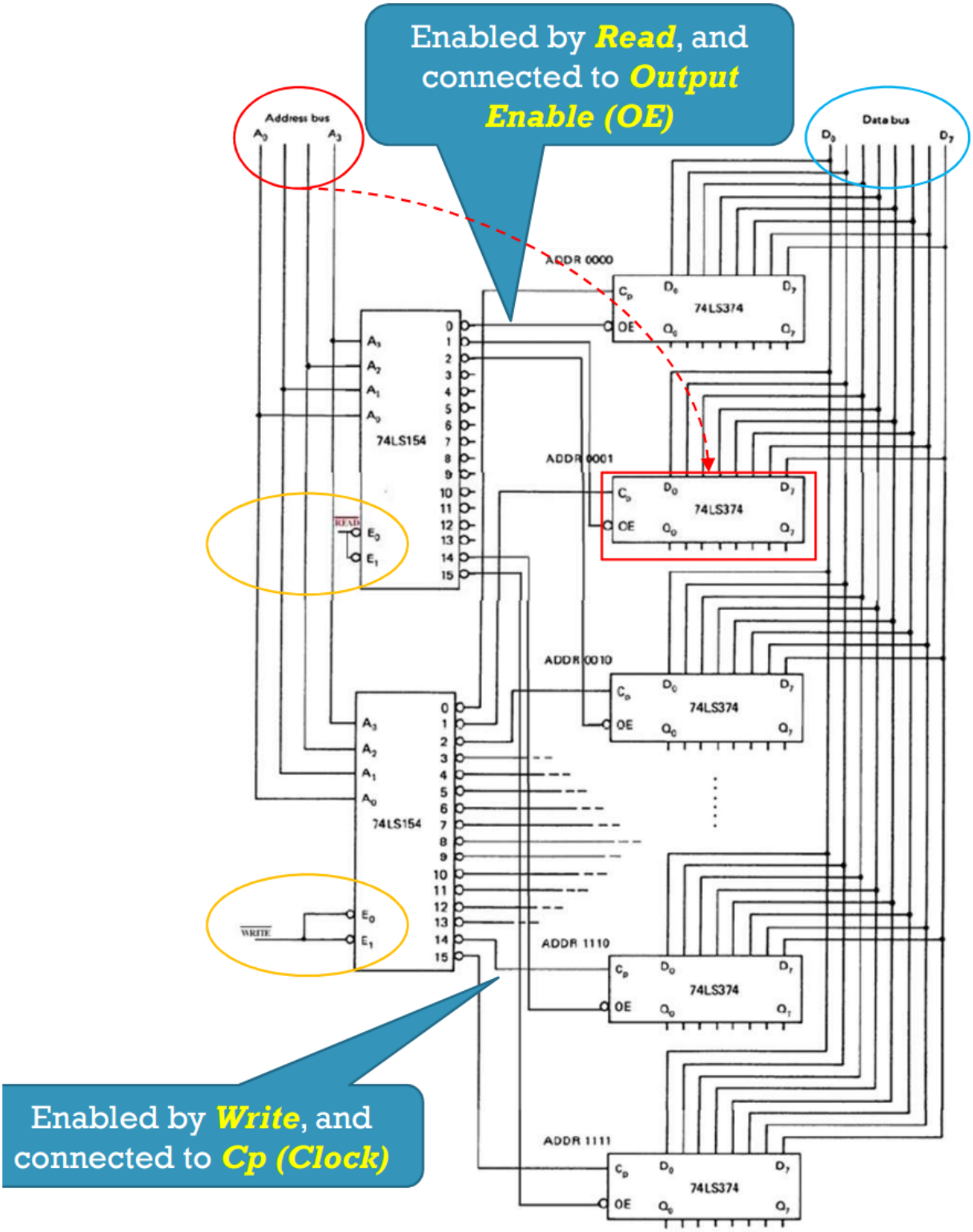
③ 16×8 **内存阵列**

74LS374 is octal D flip-flop with three-state outputs to store the data

74LS154 is a 4-Line to 16-Line Decoder/De-multiplexer to select the appropriate memory location for input/output

For READ, the Output Enable (OE) pin is LOW to get the value stored in the D flip-flop

For WRITE, the rising edge on the clock input will latch the data into flip-flop



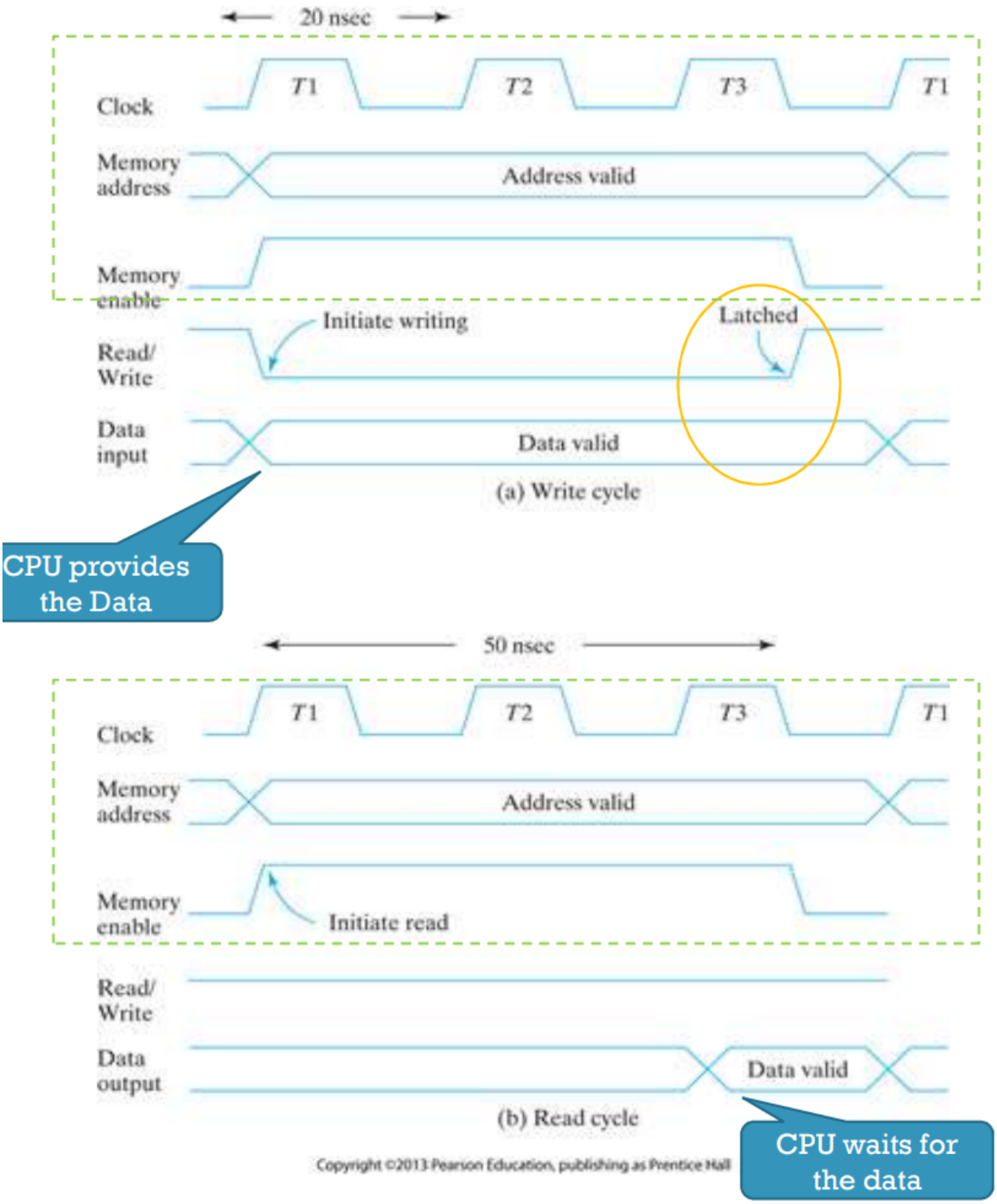
> 黄圈是控制总线.

写：接入时钟，锁存数据；读：接入 OE，输出数据.

> 注意，由于共享总线，必须保证一次只有一个芯片的输出被启用，其他都处于高阻态，否则总线冲突，烧芯片.

④ 读写周期

Timing diagrams of memory cycles



Write cycle

CPU provides the address & control signals, and data

Initiate the writing

Latch before the data expired

Read cycle

CPU provides the address & control signals

Initiate the reading

Wait for the data from the data bus

7.4.2 数据位数

> Data Size

The data size depends on the size of the data bus

n-bit data bus means the size of memory words is n-bit

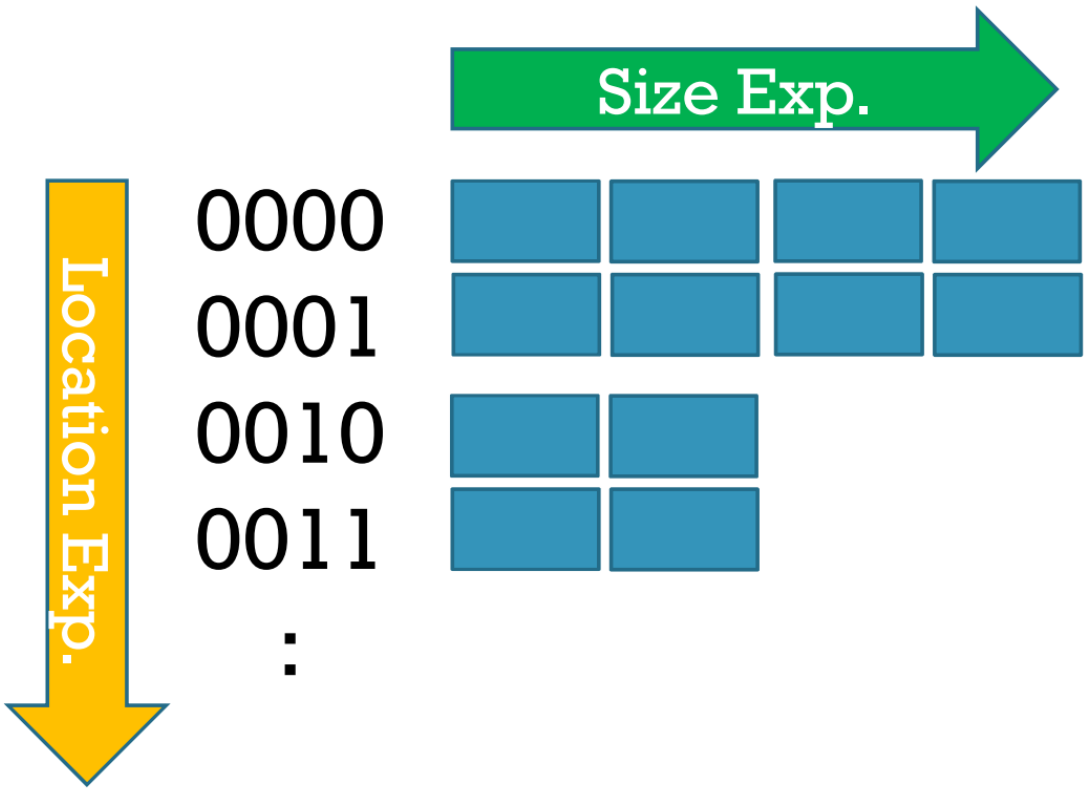
> 每个存储单元的空间.

7.5 内存扩展

> Memory Expansion

内存扩展有两种方式，扩展位宽和扩展存储单元数量.

> 如果总空间固定，扩展一个就会减小另一个，根据需要权衡.



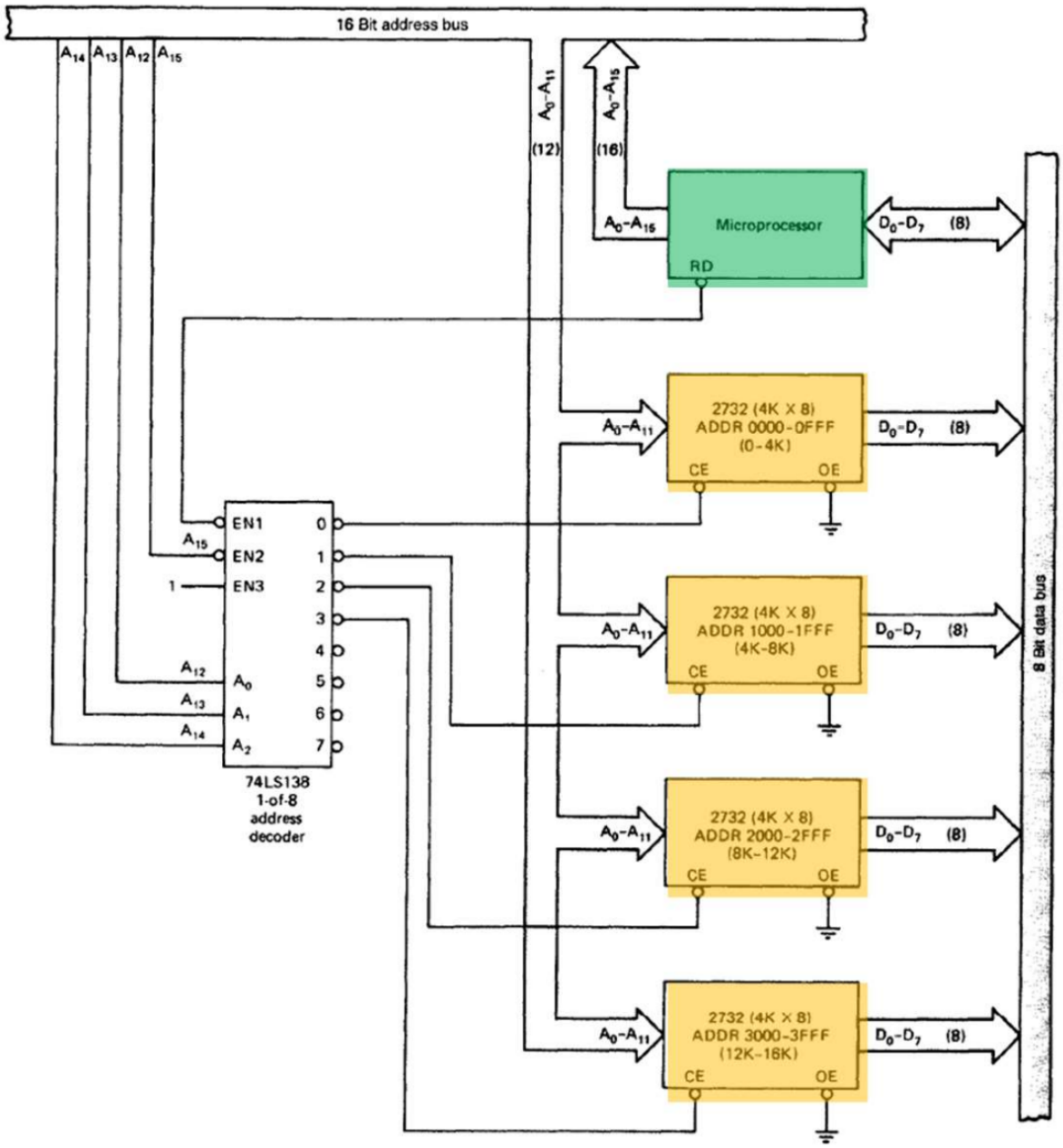
Expand the Size (e.g. 8-bit to 32-bit)

- Share all control and address signals
- Data In/Out signals are independent

> 即一组地址定位到多个芯片，每个芯片负责不同的数据位段（比如一个 8-bit 芯片负责 D0~D7）

Expand the Locations (e.g. 4K to 16K)

- Share all control signals and Data In/Out signals (i.e. share the same data bus)
- Share part of address signals (start from LSB) and the remaining address signals are used for IC selection



这张图展示了如何通过**地址线分段与译码器选择机制**，将多个内存芯片组合成一个更大的连续地址空间，实现 **地址扩展（Location Expansion）**，是一种典型的"分片式内存扩展设计".

> 看不懂.

附录

1.七段显示屏

> 7-segment
>
> 关于七段显示屏的一个简化版本（只关注数字 $1,2,3,4$），见 `Lab 2`.

真值表

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	

$$
\begin{aligned}
a &= A'B'C'D' + A'B'CD' + A'B'CD + A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'C'D \\
b &= A'B'C'D' + A'B'C'D + A'B'CD' + A'B'CD + A'BC'D' + A'BCD + AB'C'D' + AB'C'D \\
c &= A'B'C'D' + A'B'C'D + A'B'CD + A'BC'D' + A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'C'D \\
d &= A'B'C'D' + A'B'CD' + A'B'CD + A'BC'D + A'BCD' + AB'C'D' + AB'C'D \\
e &= A'B'C'D' + A'B'CD' + A'BCD' + AB'C'D' \\
f &= A'B'C'D' + A'BC'D' + A'BC'D + A'BCD' + AB'C'D' + AB'C'D \\
g &= A'B'CD' + A'B'CD + A'BC'D' + A'BC'D + A'BCD' + AB'C'D' + AB'C'D
\end{aligned}
$$

解码器：K-map 化简

> 这里 `X` 表示 Don't Care 万能格.

a

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$a = A + C + BD + (B + D)'$

b

AB\CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	X	X	X	X
10	1	1	X	X

$b = B' + (C + D)' + CD$

c

AB\CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$c = B + C' + D$

d

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	X	X	X	X
10	1	1	X	X

$d = A + B'D' + CD' + (B + C')' + BC'D$

e

AB\CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

$e = (B + D)' + (C' + D)'$

f

AB\CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

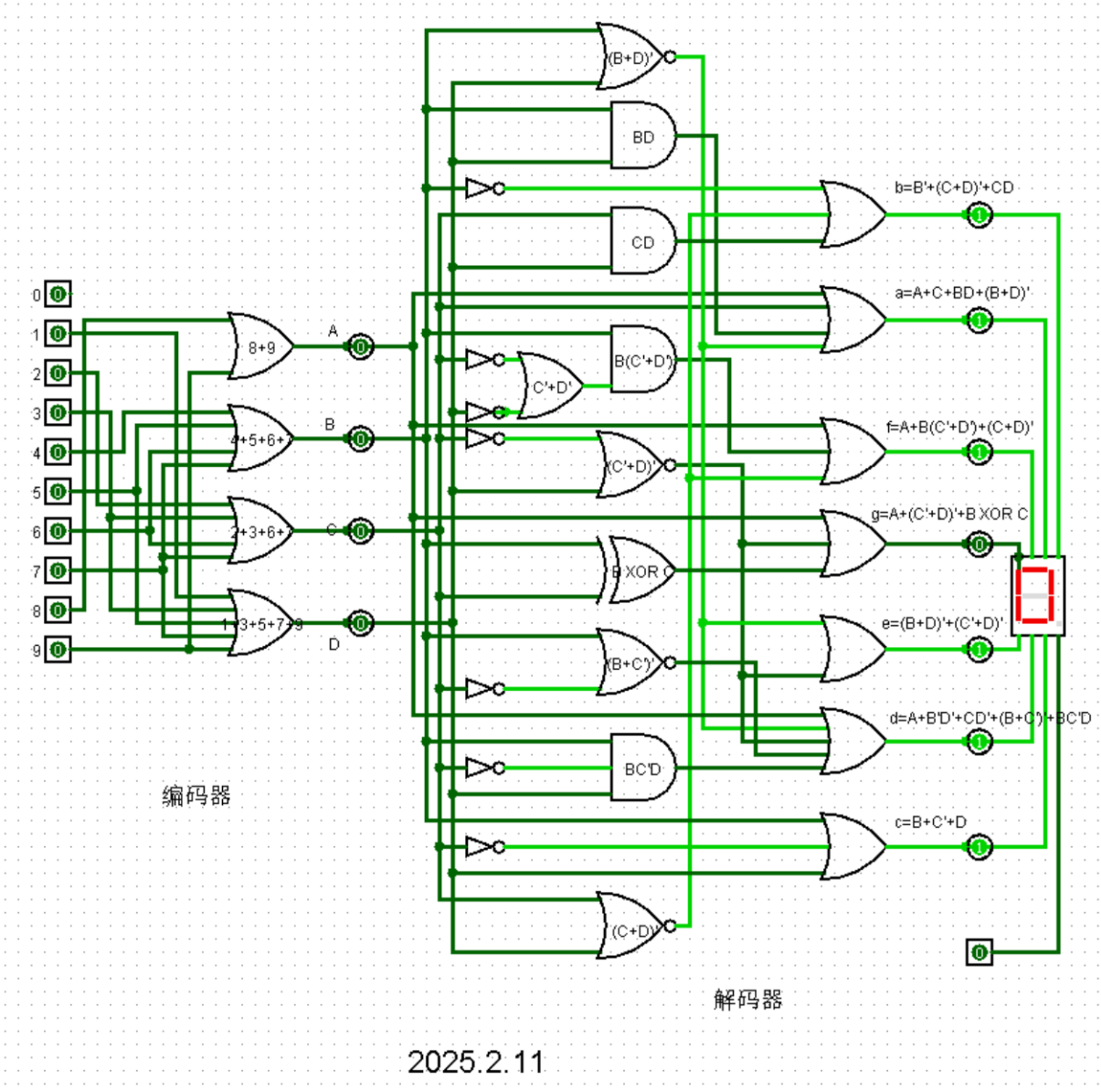
$f = A + B(C' + D') + (C + D)'$

g

AB\CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$g = A + (C' + D)' + B \oplus C$

电路图


0
1
2
3
4
5
6
7
8
9
8+9
4+5+6+
2+3+6+
1+3+5+7+9
A
B
C
D
编码器
(B+D)'
BD
CD
C'+D'
B(C'+D')
(C'+D)'
B XOR C
(B+C')'
BC'D
(C+D)'
b=B'+(C+D)'+CD
a=A+C+BD+(B+D)'
f=A+B(C'+D')+(C+D)'
g=A+(C'+D)'+B XOR C
e=(B+D)'+(C'+D)'
d=A+B'D'+CD'+(B+C')+BC'D
c=B+C'+D
解码器

2025.2.11