

第四章 处理器体系结构

4-2 ——逻辑设计

教师：夏 文

计算机科学与技术学院

哈尔滨工业大学（深圳）

本节主要内容

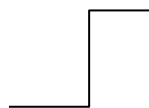
- 逻辑电路分类
 - 组合逻辑
 - 时序逻辑

- 构建CPU的硬件模块
 - ALU
 - 多路选择器
 - 比较器
 - 存储器
 - 寄存器文件
 - 读（组合逻辑）
 - 写（时序逻辑）

逻辑电路分类

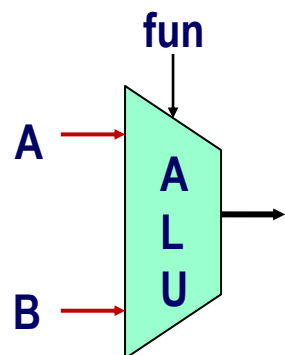
- **组合逻辑—输出仅取决于当前的输入**
 - 计算输入的布尔函数
 - 对输入的变化持续做出反应
 - 对数据做出操作并实施控制
- **时序逻辑—时钟上升沿来临时才更新输出**
 - 存储字节
 - 可寻址的内存
 - 不可寻址的寄存器
 - 时钟上升沿触发

时钟上升
沿

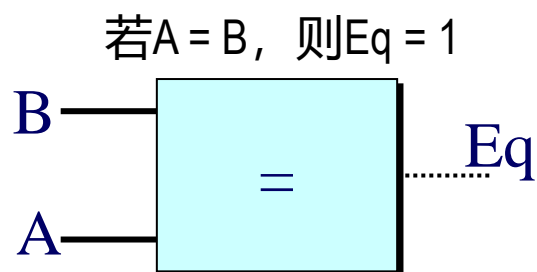


构建CPU的硬件模块

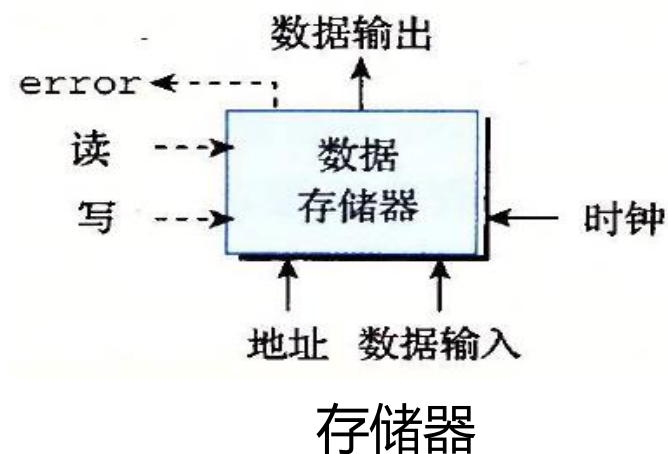
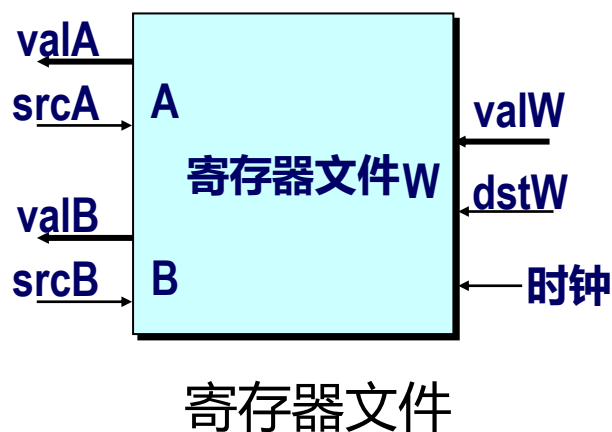
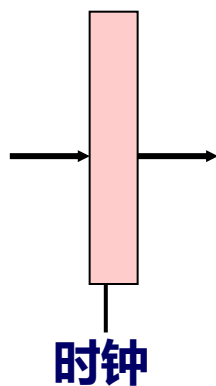
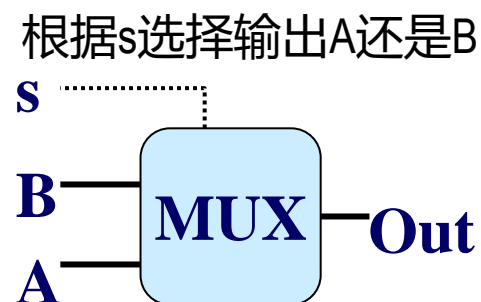
算术/逻辑单元(ALU)

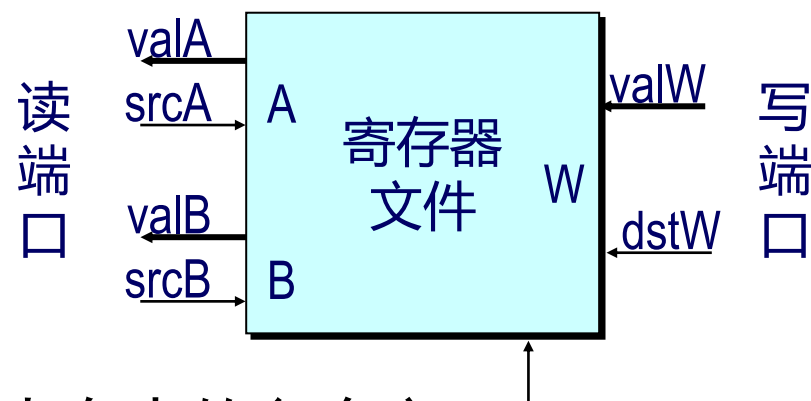
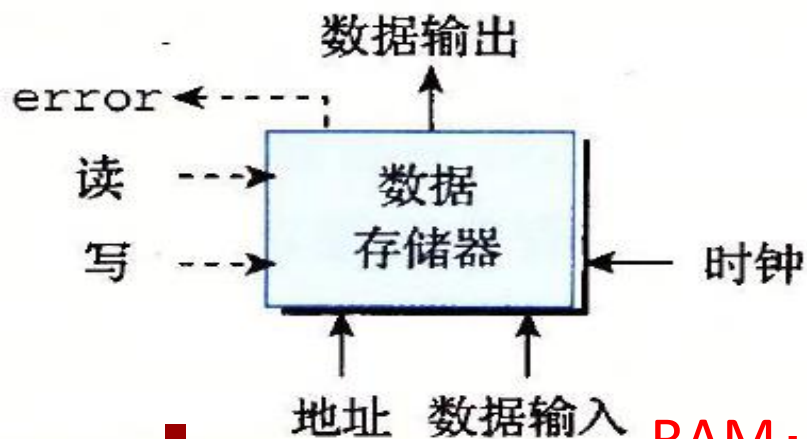


相等



多路选择 (二路选择)



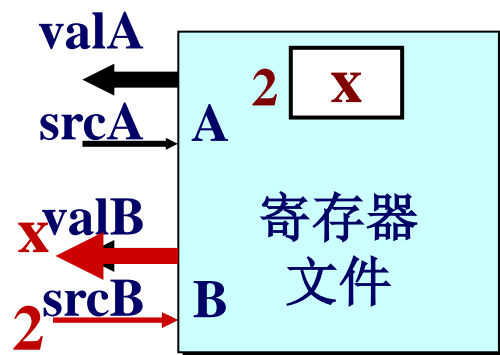


随机存取存储器

- **RAM:** 存储内存中的多个字 时钟
 - 通过输入的地址来决定读/写哪个字，写由clk控制。
- **RF: 寄存器文件**
 - **硬件寄存器:** 稳态、组合逻辑的屏障，CLK边沿触发。如：PC、CC、Stat 等
 - **程序寄存器:** 存储数据，可按ID读、写的存储单元。汇编级用 `%rax`, `%rsp`, `%r14` 等，机器级-寄存器ID标识符作为地址 (0000-1110) 15 (0xF) 表示不执行读写
- **PORTS: 多端口**
 - 在每个周期可以同时读/写多个字
 - 每个端口有单独的地址和数据输入/输出

寄存器文件时序

注意读写不同！！

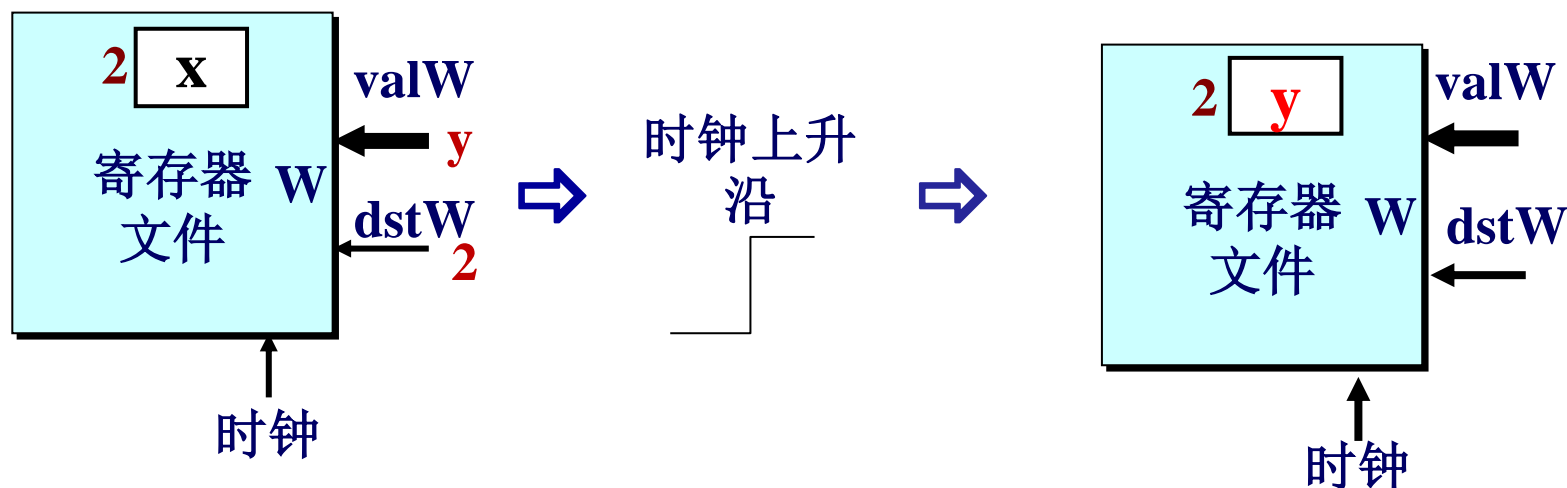


■ 读

- 类似组合逻辑
- 根据输入地址产生输出数据，
 - 有延迟


■ 写（属于时序逻辑）

- 类似寄存器
- 只在时钟上升沿更新



4-2 逻辑设计习题

1.判断对错:

在Y86-64的顺序结构实现中, 寄存器文件写时是作为组合逻辑器件看待。 ()

逻辑设计概述2

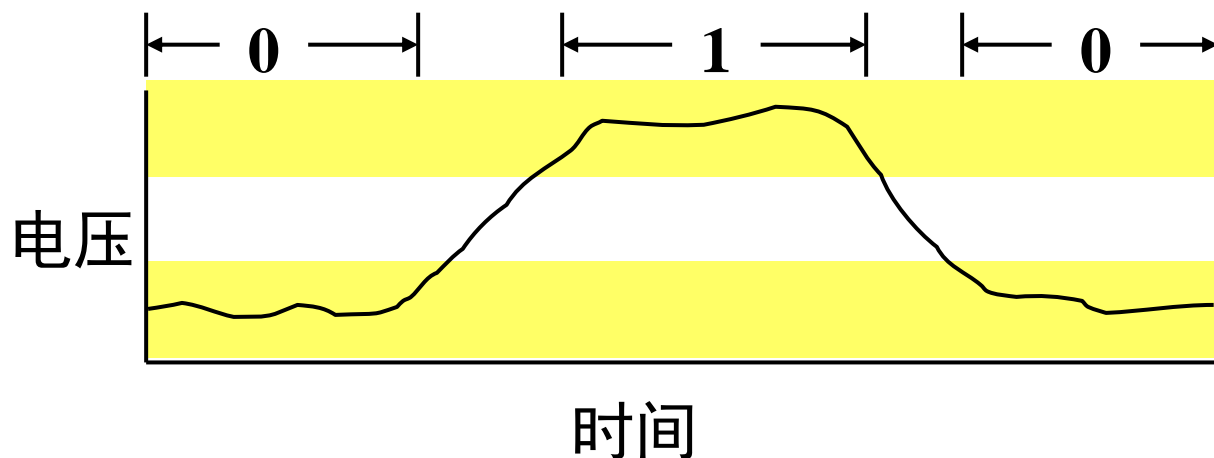
■ 基本的硬件需求

- 通讯
 - 如何将值从一个地方移动到另外一个地方
- 计算
- 存储

■ 比特是我们的朋友

- 全部用0和1表示
- 通讯
 - 电线上的低或高电压
- 计算
 - 计算布尔函数
- 存储
 - 存储信息位

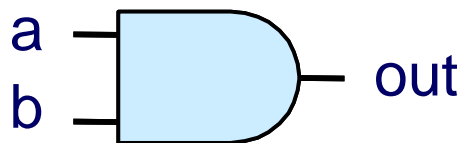
数字信号



- 用电压阈值从连续信号中提取离散值
- 最简单版本：1位信号
 - 或者是高位，或者是低位
 - 在高位和低位之间有保护范围
- 不会收到噪音或者低质量的电路因素影响
 - 可以使得电路简单、规模小、速度快

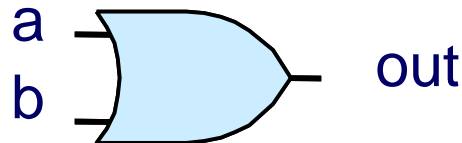
通过逻辑门进行计算

And



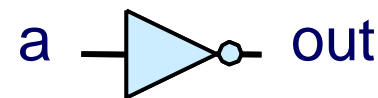
$$\text{out} = a \ \&\& \ b$$

Or



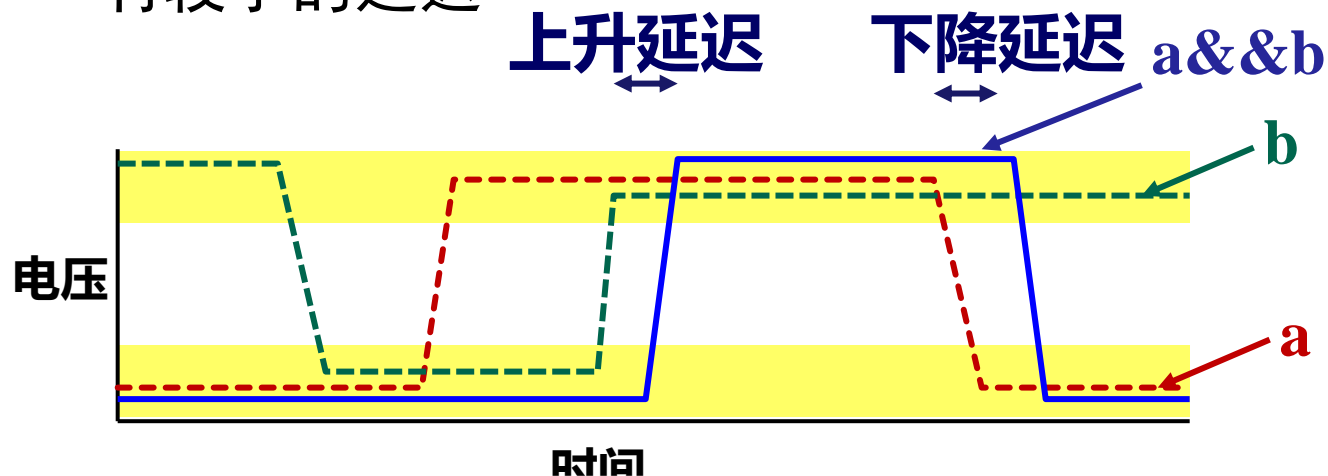
$$\text{out} = a \ || \ b$$

Not

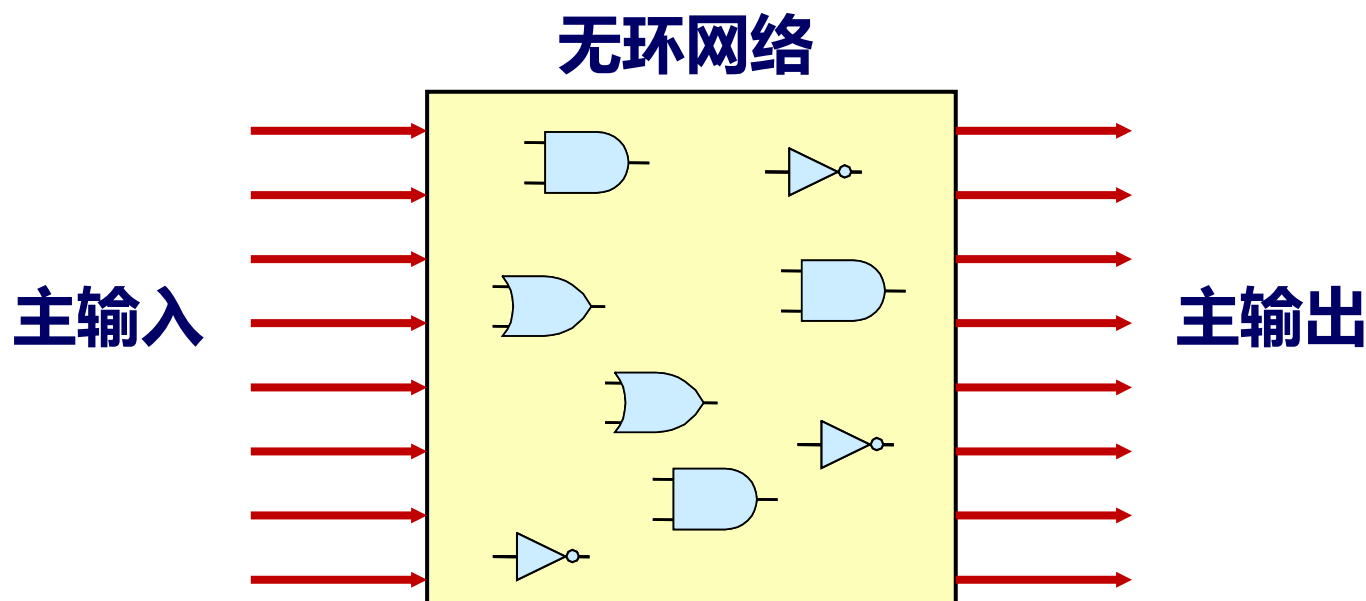


$$\text{out} = !a$$

- 输出是输入的布尔函数
- 连续响应输入的变化
 - 有较小的延迟



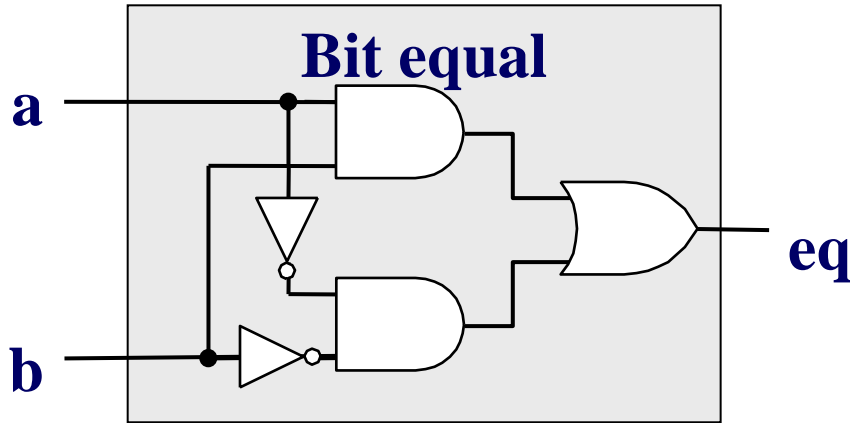
混合电路



■ 逻辑门无环网络

- 连续响应主输入的变化
- 主输出是主输入的布尔函数（有延迟）

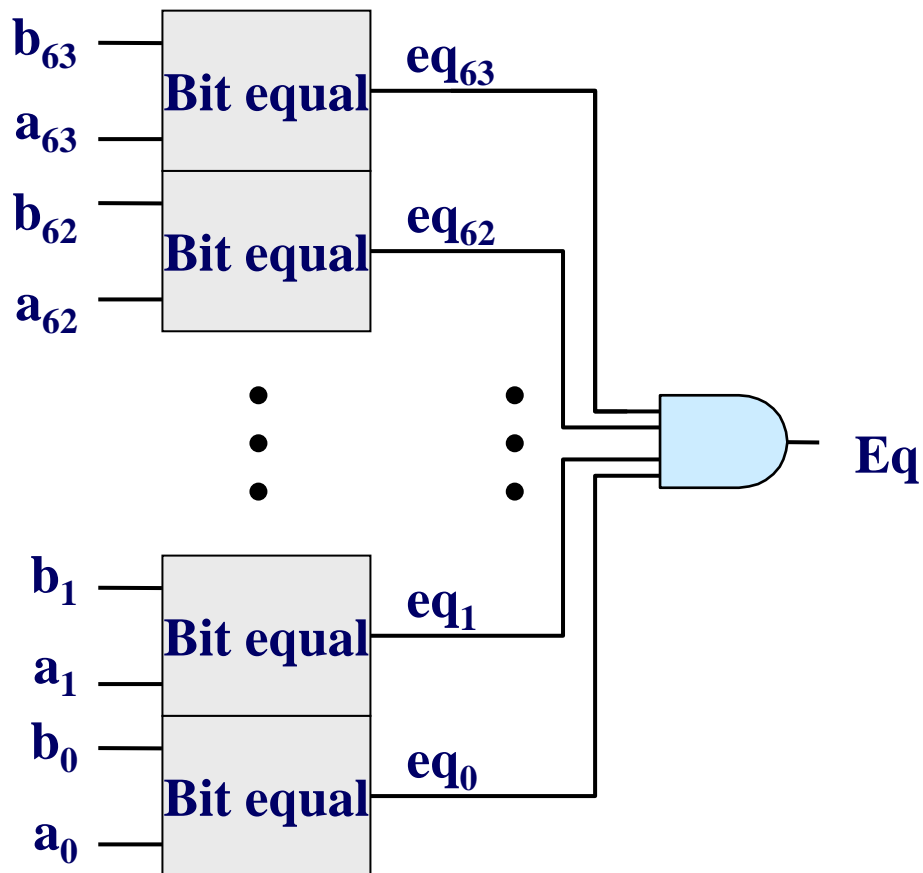
位相等



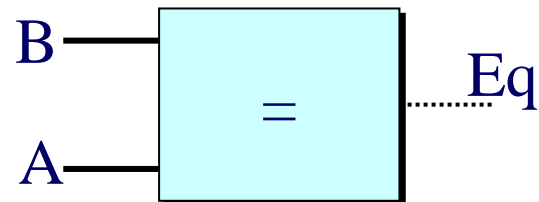
硬件控制语言(HCL)表达式
 $\text{bool eq} = (a \& \& b) \parallel (!a \& \& !b)$

- 如果a和b相等就生成1
- **硬件控制语言(HCL, Hardware Control Language)**
 - 非常简单的硬件描述语言
 - 布尔操作的语法和C语言逻辑运算相似
 - 将用HCL描述处理器的控制逻辑

字相等



字级表示

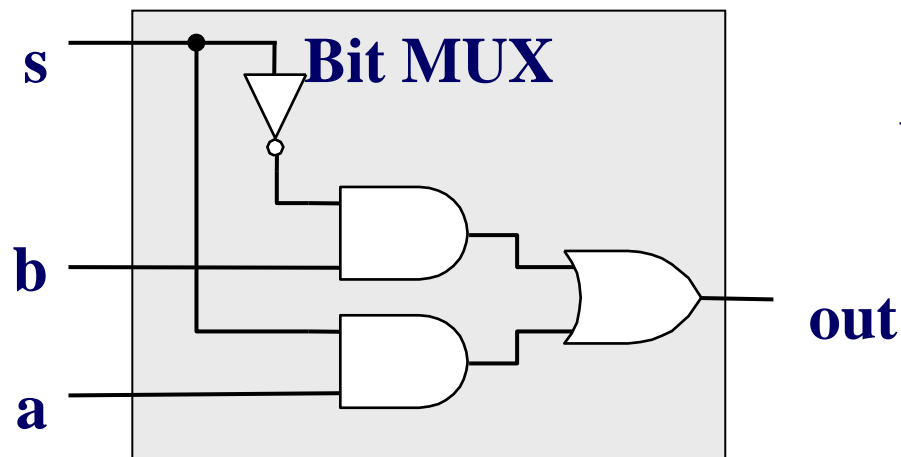


HCL表示

bool Eq = (A == B)

- 64 位 字的大小
- HCL表示
 - 相等操作
 - 产生布尔值

位多路复用器

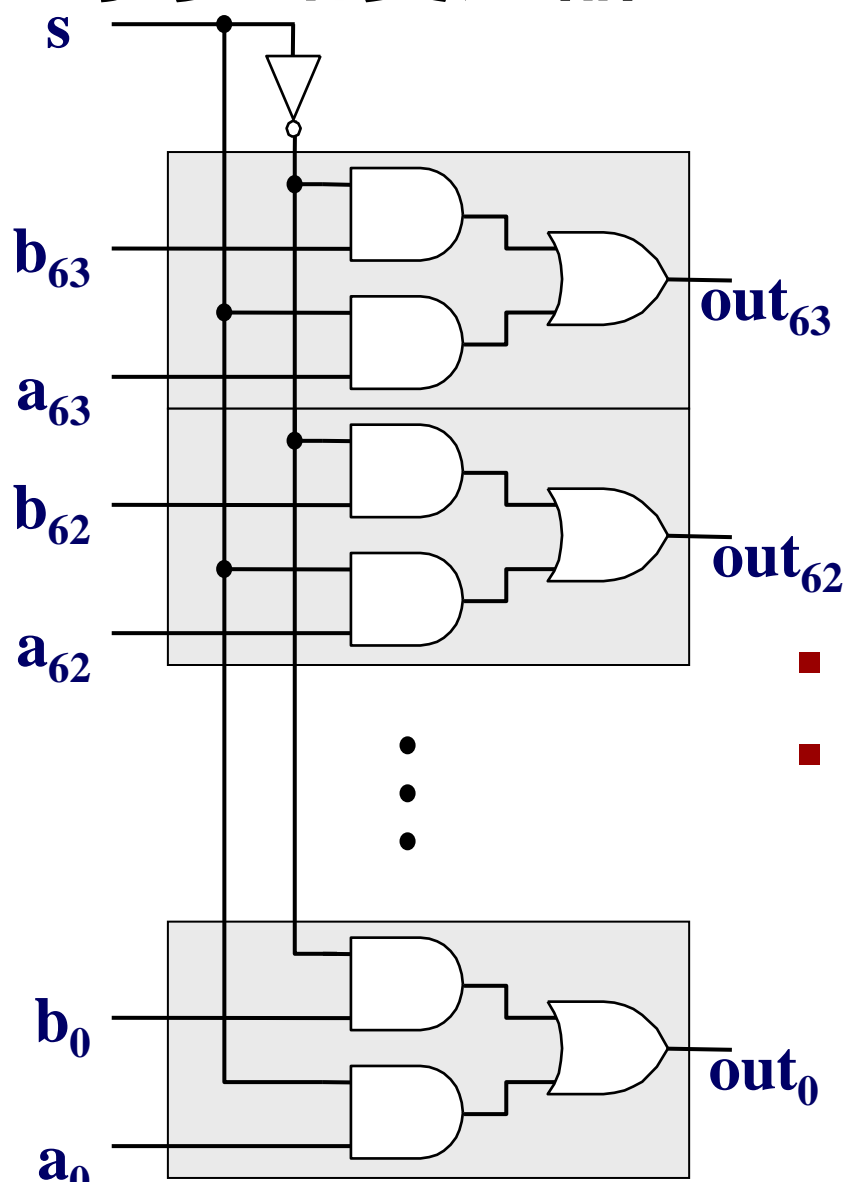


HCL表示

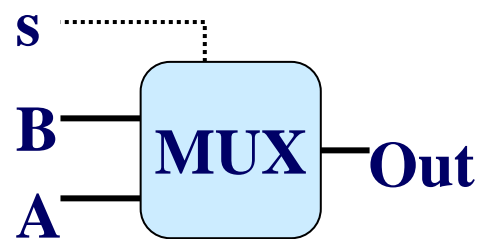
$\text{bool out} = (s \& \& a) || (!s \& \& b)$

- 控制信号 s
- 数据信号 a 和 b
- 当 $s=1$ 时输出 a ，当 $s=0$ 时输出 b

字多路复用器



字级表示



HCL表示

```
int Out = [
    s : A;
    1 : B;
];
```

- 根据控制信号 s 来选择输入字 A 或 B
- 硬件控制语言表示
 - 情况表达式(case语句)
 - 一系列二元组“布尔表达式:整数表达式”
 - 第一个求值为1的情况会被选中

硬件控制语言(HCL)字级示例

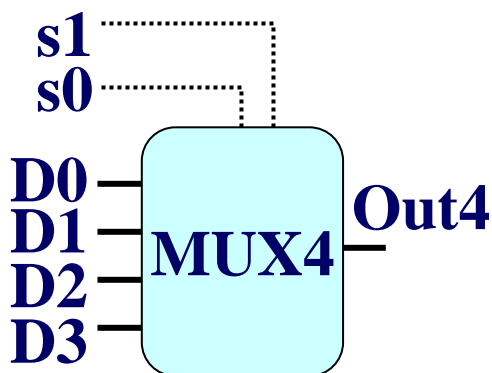
3个字的最小值



```
int Min3 = [
    A < B && A < C : A;
    B < A && B < C : B;
    1               : C;
];
```

- 找到三位输入中最小的字
- HCL情况表达式
- 最后的情况：保证有一个匹配值

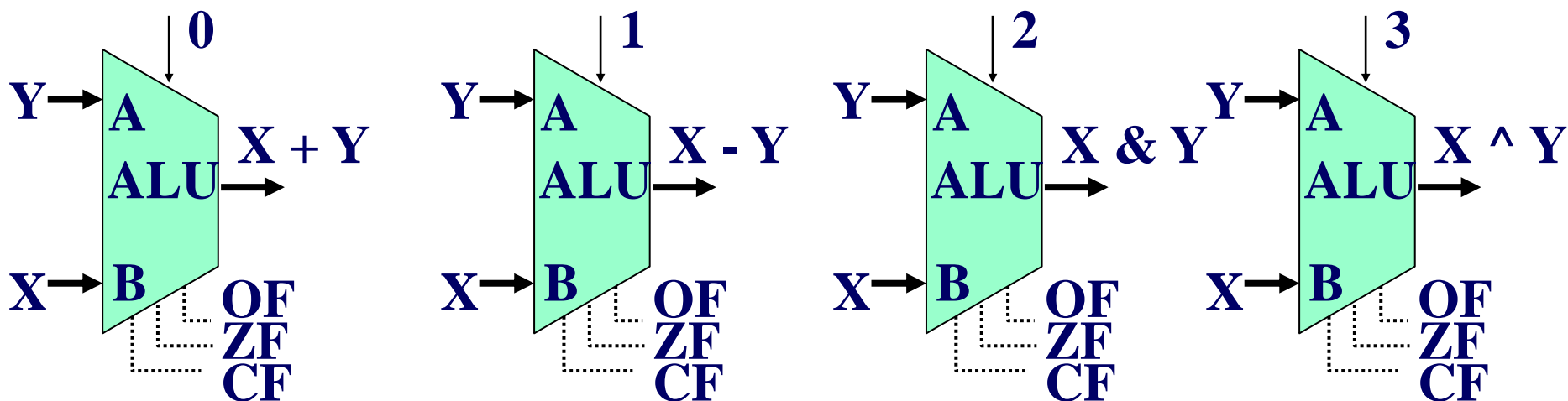
4路复用器



```
int Out4 = [
    !s1 && !s0 : D0;
    !s1       : D1;
    !s0       : D2;
    1         : D3;
];
```

- 根据两个控制位，从4个输入中选择一个（字）
- HCL情况表达式
- 通过假定顺序地匹配，简化测试

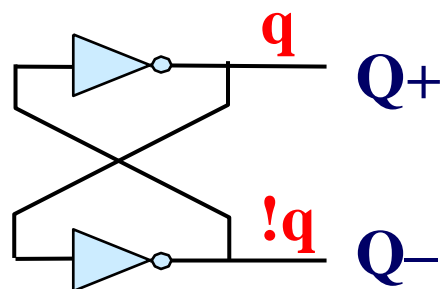
算术/逻辑单元(ALU)



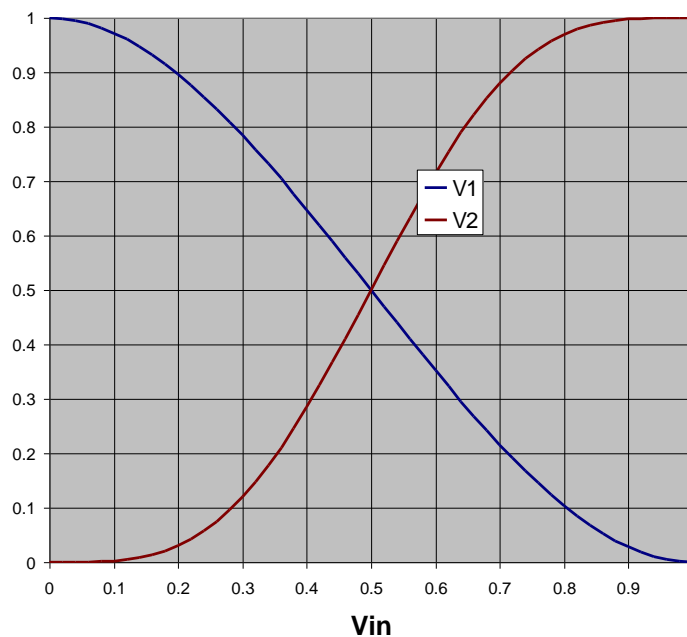
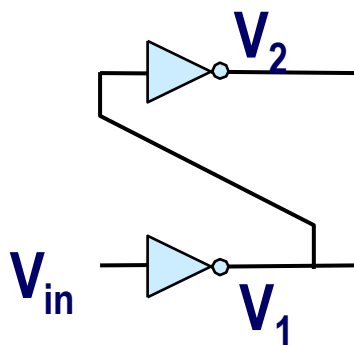
- 组合逻辑
 - 连续响应输入
- 控制信号选择计算函数
 - 对应Y86-64中4种算术/逻辑操作，控制值和操作的功能码对应
 - 也计算条件码的值

存储 1 位

双稳态元件

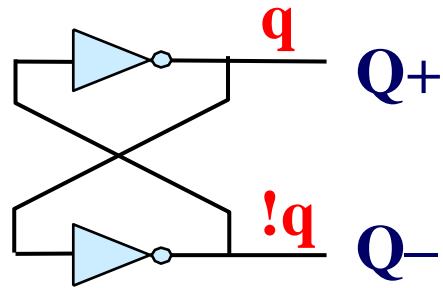


$q = 0 \text{ or } 1$

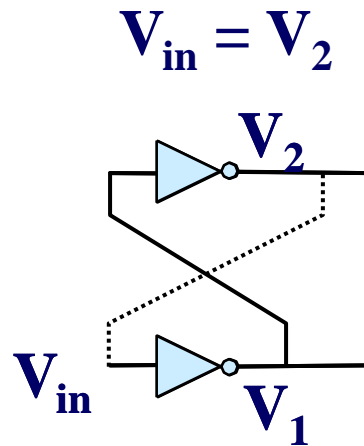


存储1位 (cont.)

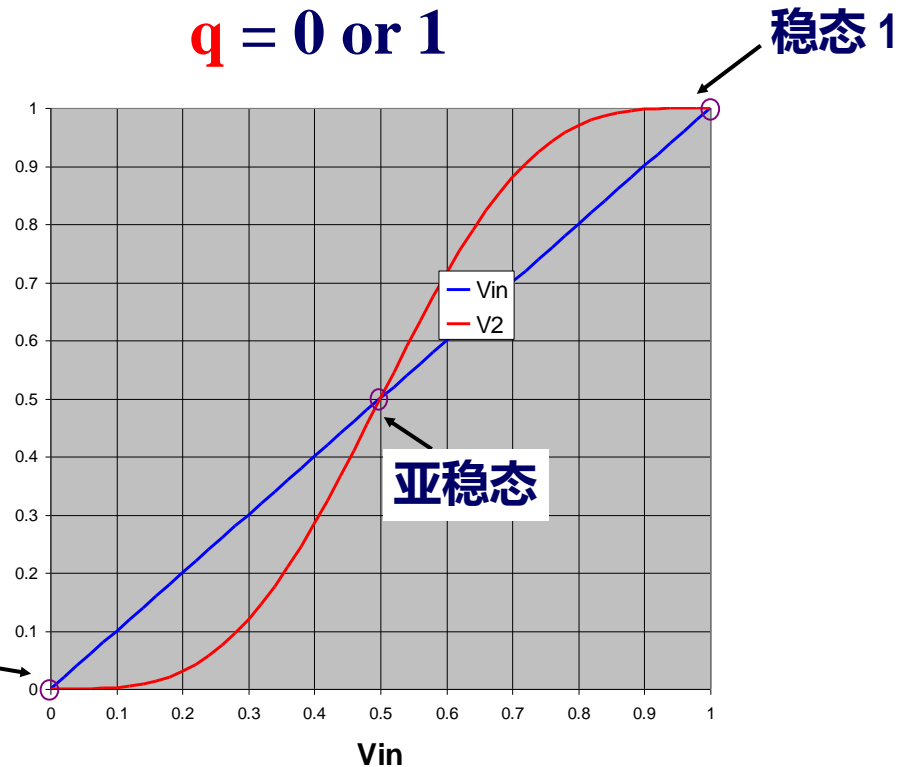
双稳态元件



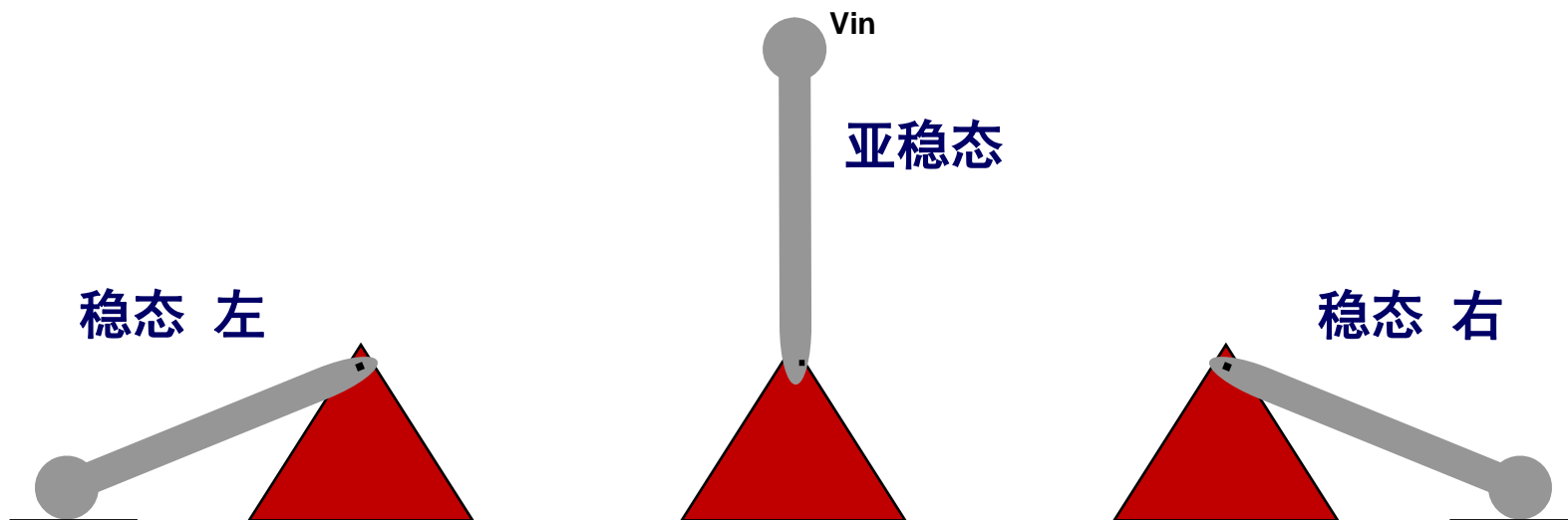
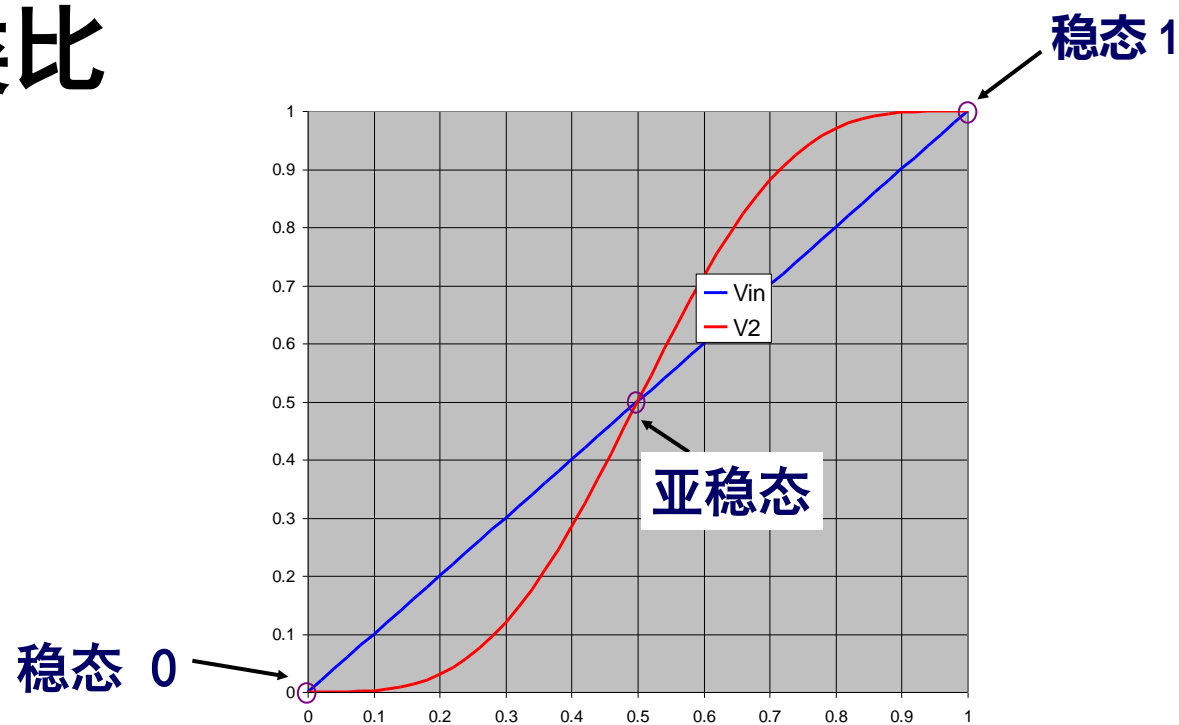
$q = 0 \text{ or } 1$



稳态 0

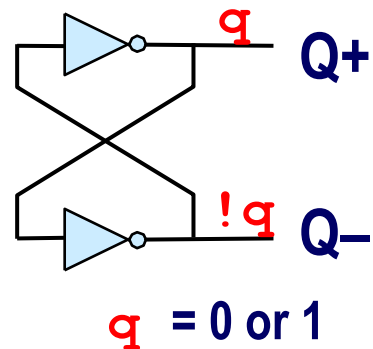


物理类比

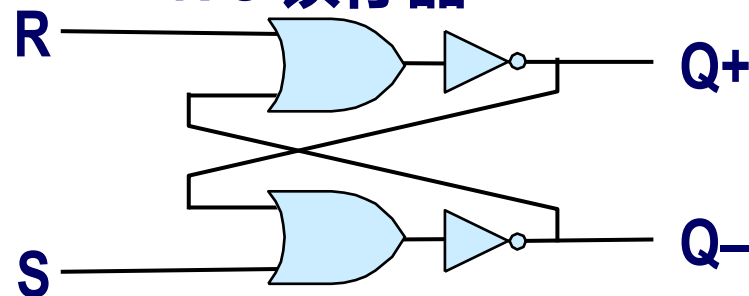


存储、访问1 位

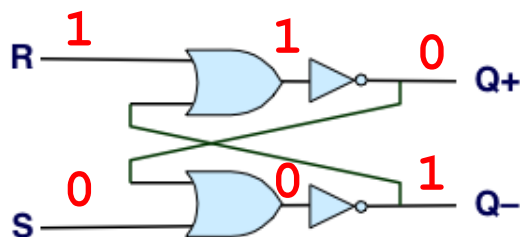
双稳态元件



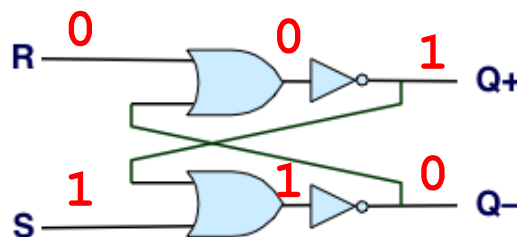
R-S 锁存器



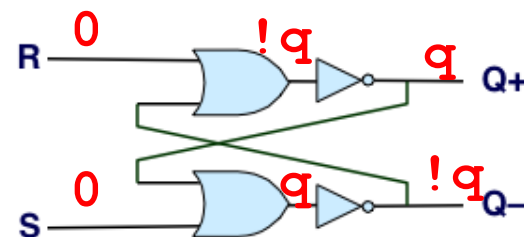
重新设置



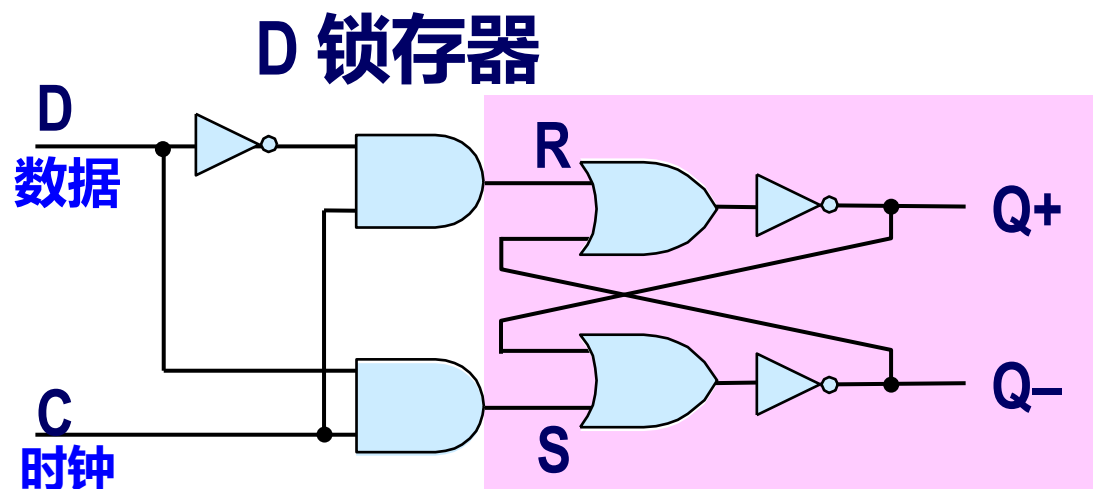
设置



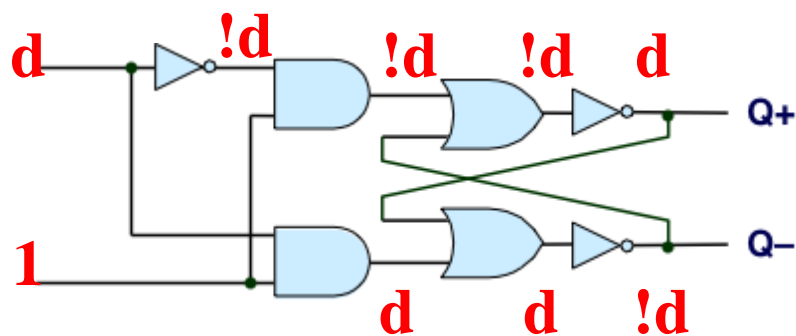
存储



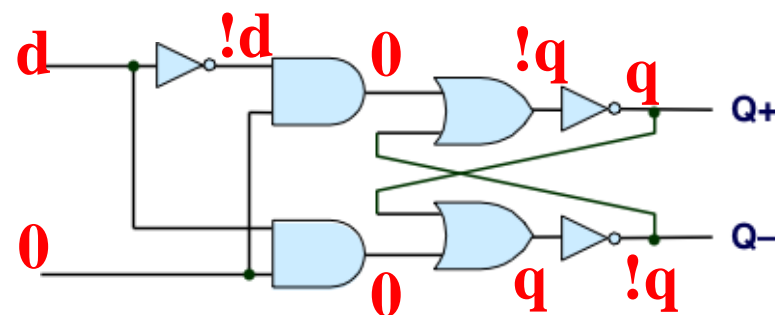
1位锁存器



锁存Latching

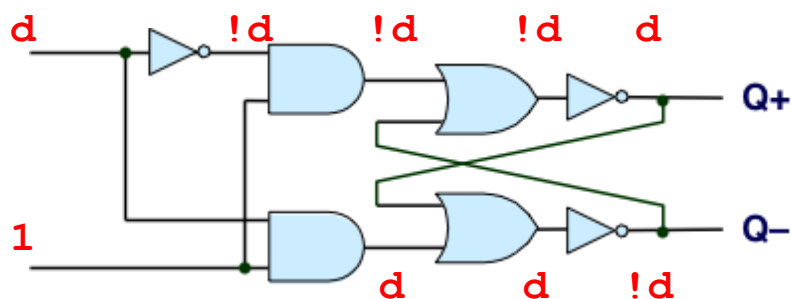


存储

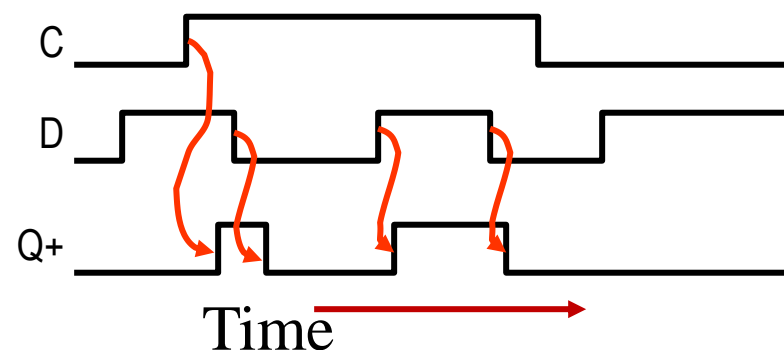


透明1位锁存器

锁存Latching

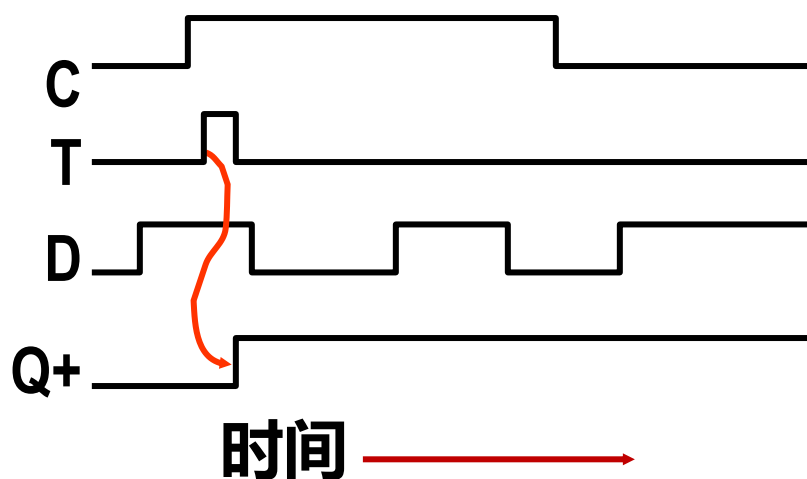
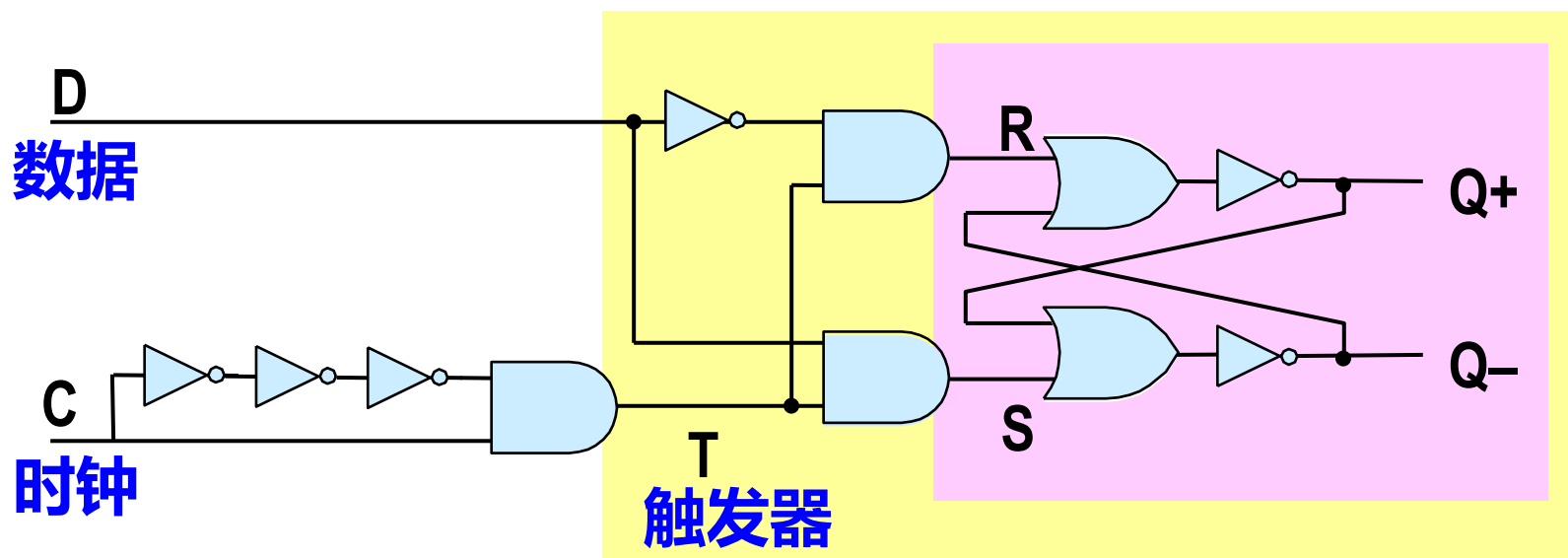


改变D



- 当在锁存模式时，D的信号会传递给Q+ 和 Q-
- 当C下降时，锁存的值取决于D

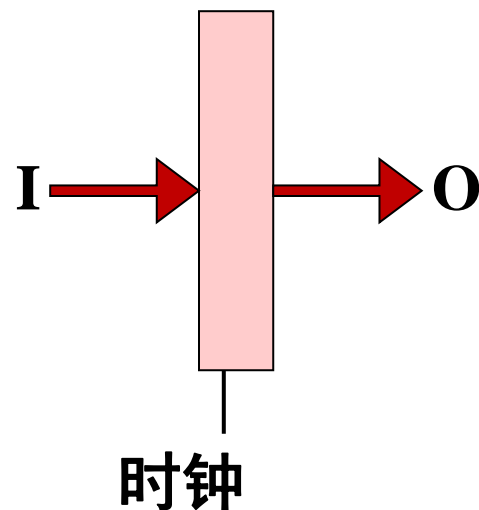
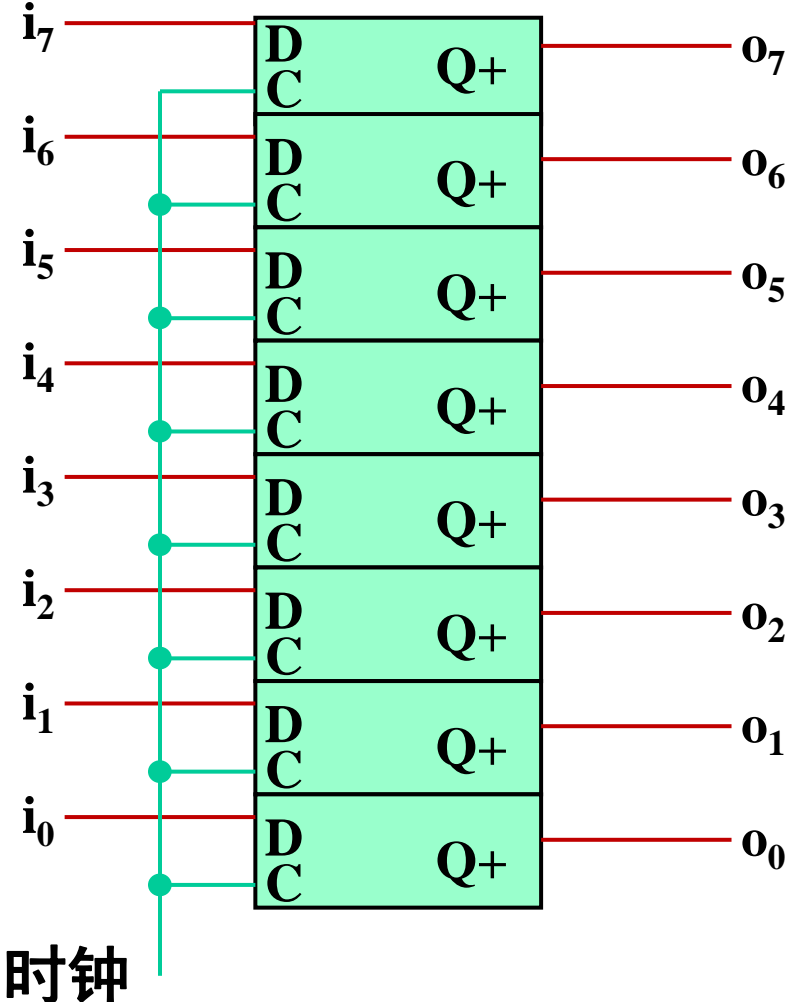
边缘触发锁存器



- 只在短暂的时间段处于锁存模式
 - 时钟上升沿 （下降沿？）
- 当时钟上升的时候，锁存的值取决于数据
- 在其他时候输出保持不变

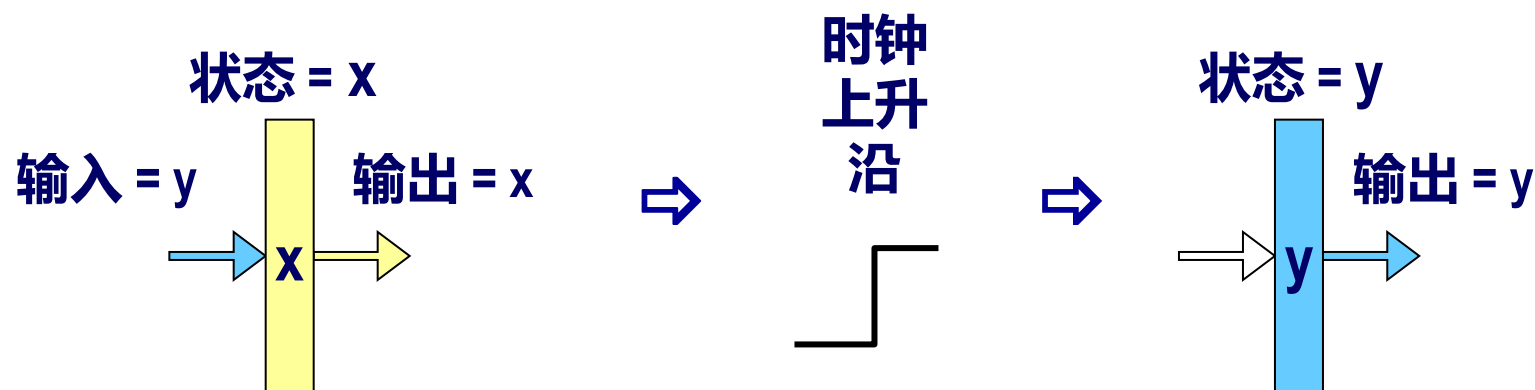
寄存器—时钟寄存器

结构



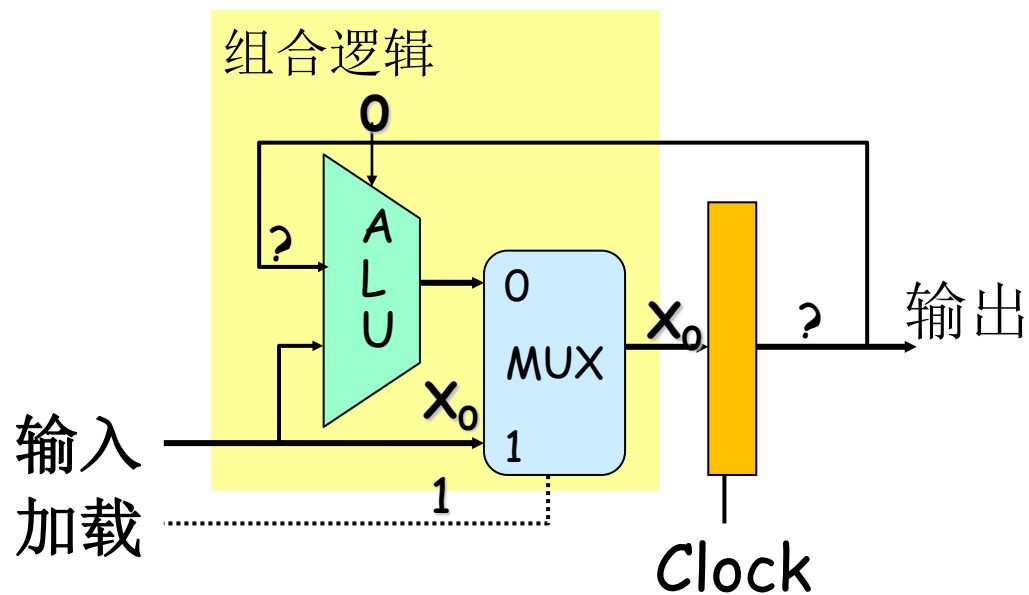
- 存储字数据
 - 和在汇编代码中看到的程序寄存器不同
- 边缘触发锁存器的集合
- 在时钟上升沿加载输入

寄存器操作

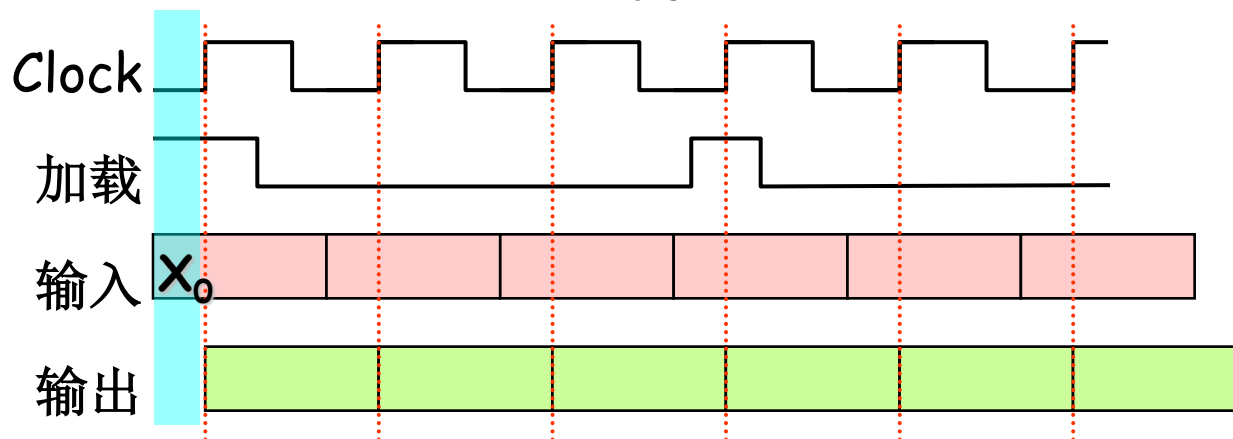


- 存储数据位
- 大多数时候作为输入和输出之间的栅栏（隔离）
- 当时钟上升的时候，加载输入

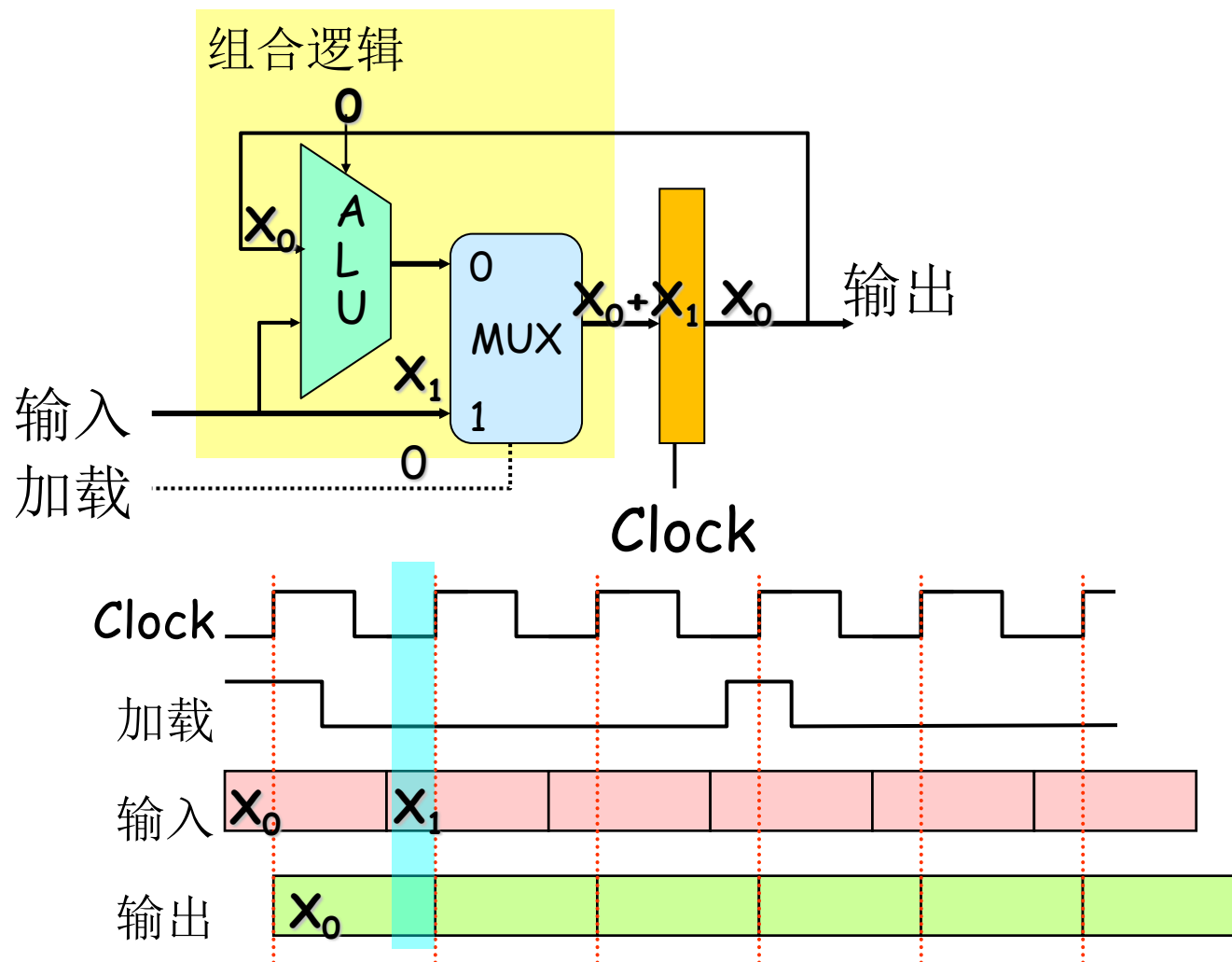
状态机示例



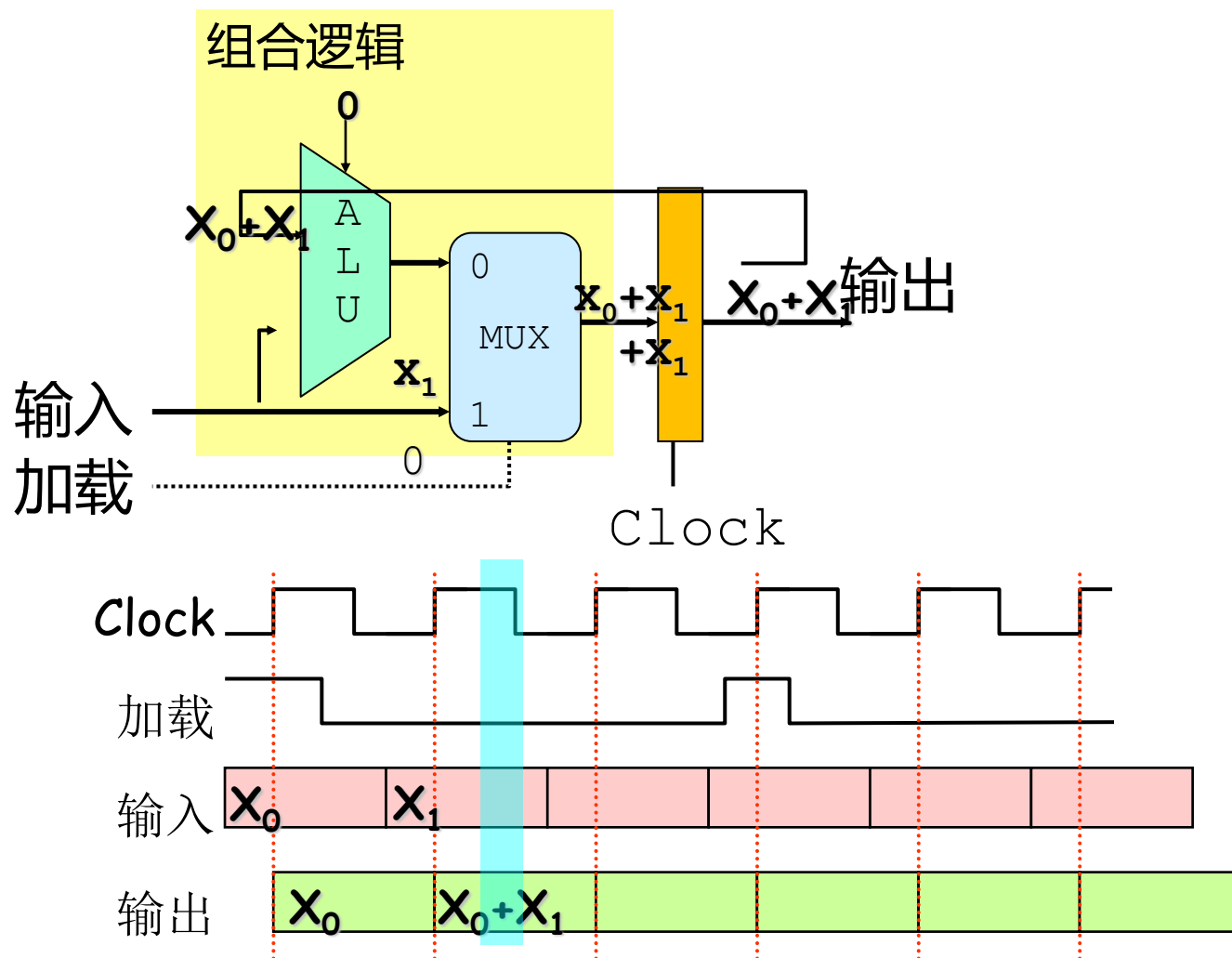
- 累加器示例
- 在每个时钟加载或累加



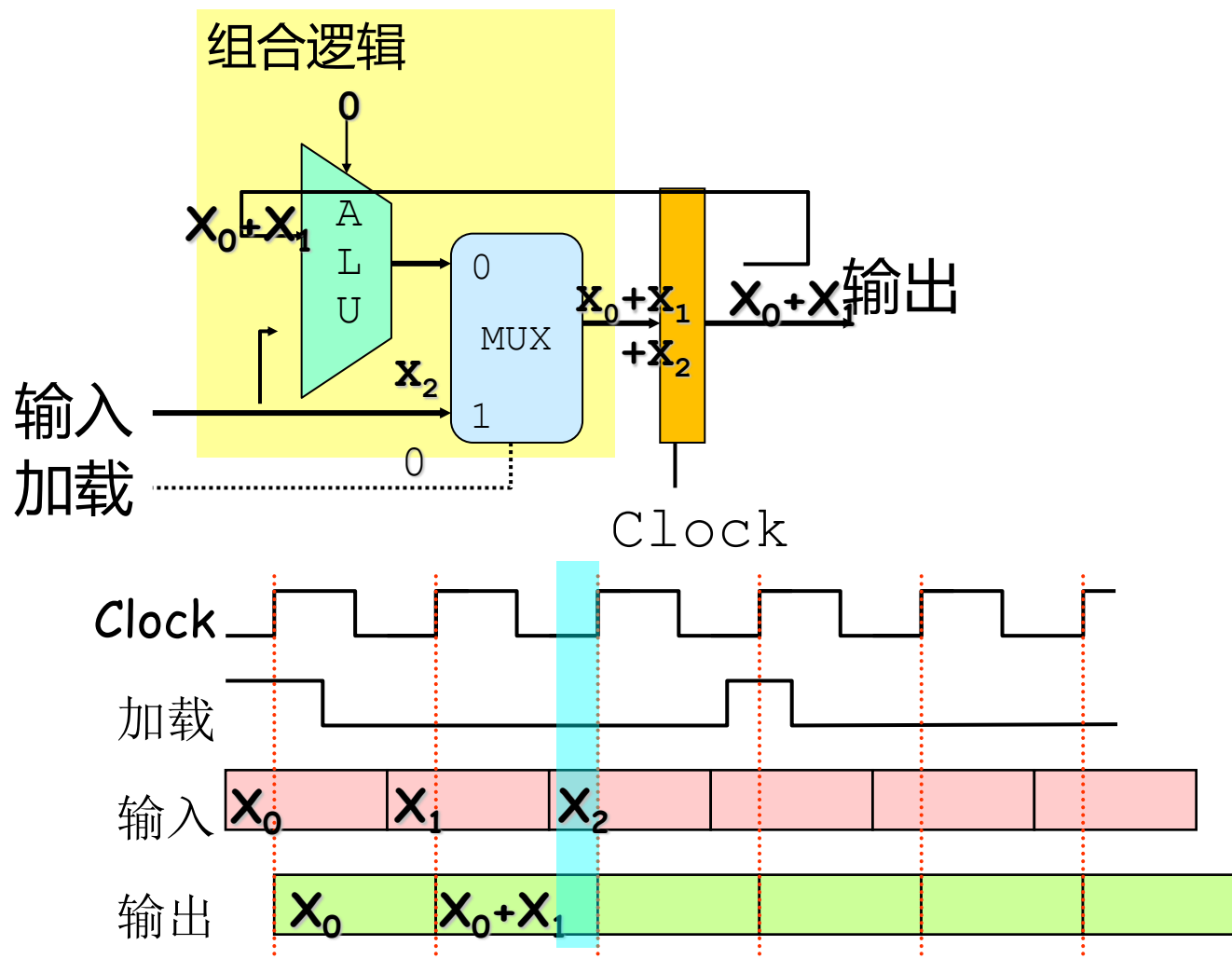
状态机示例



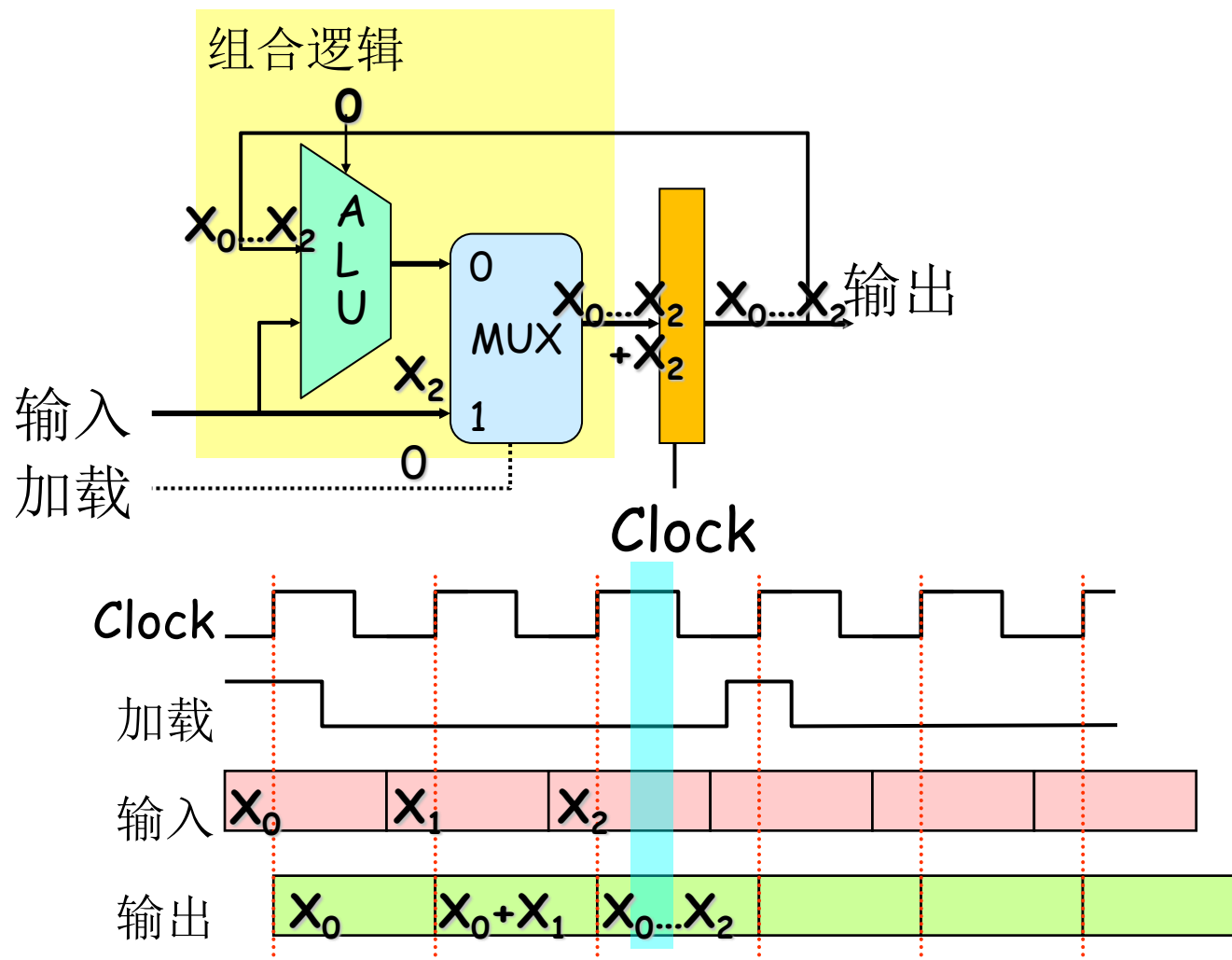
状态机示例



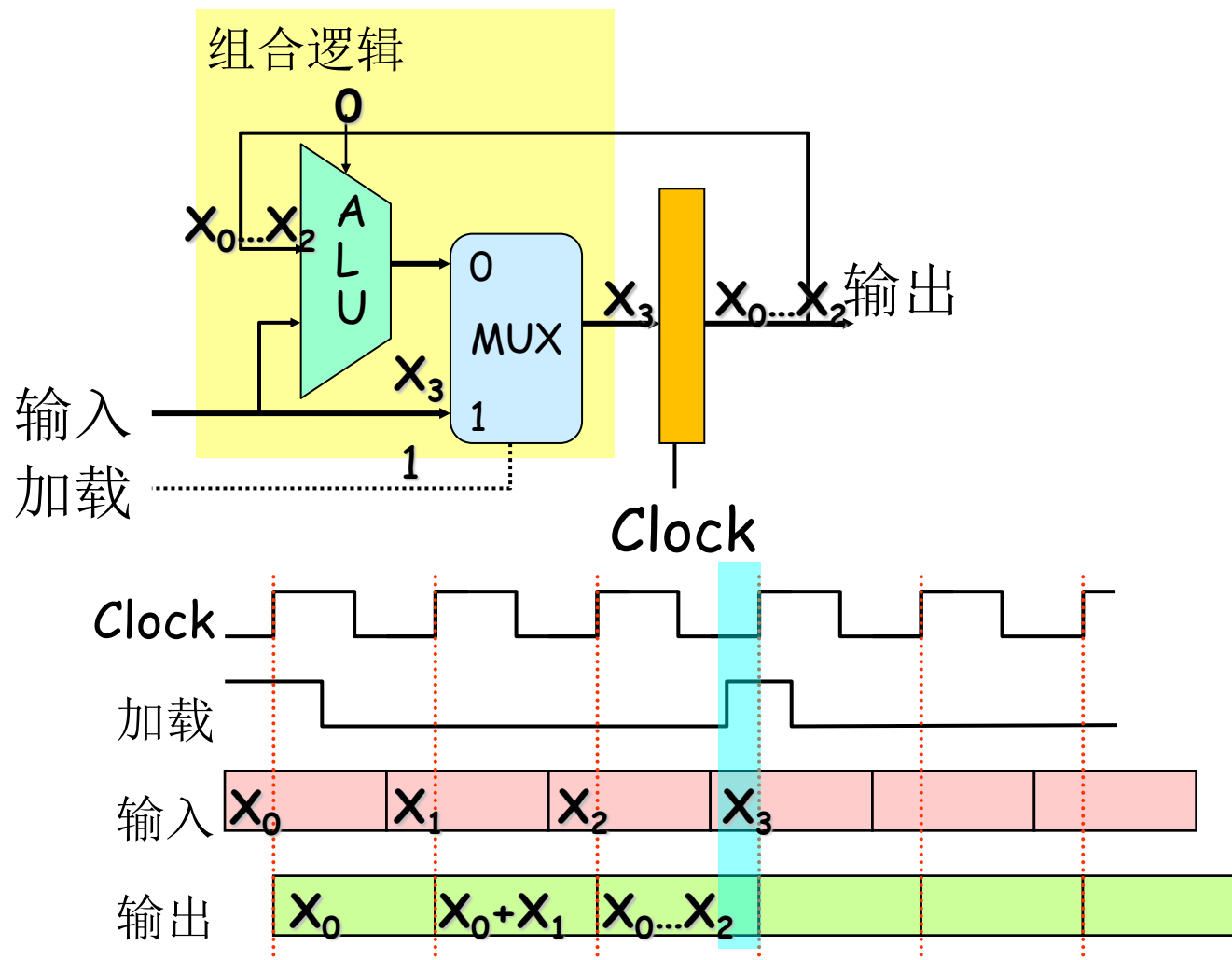
状态机示例



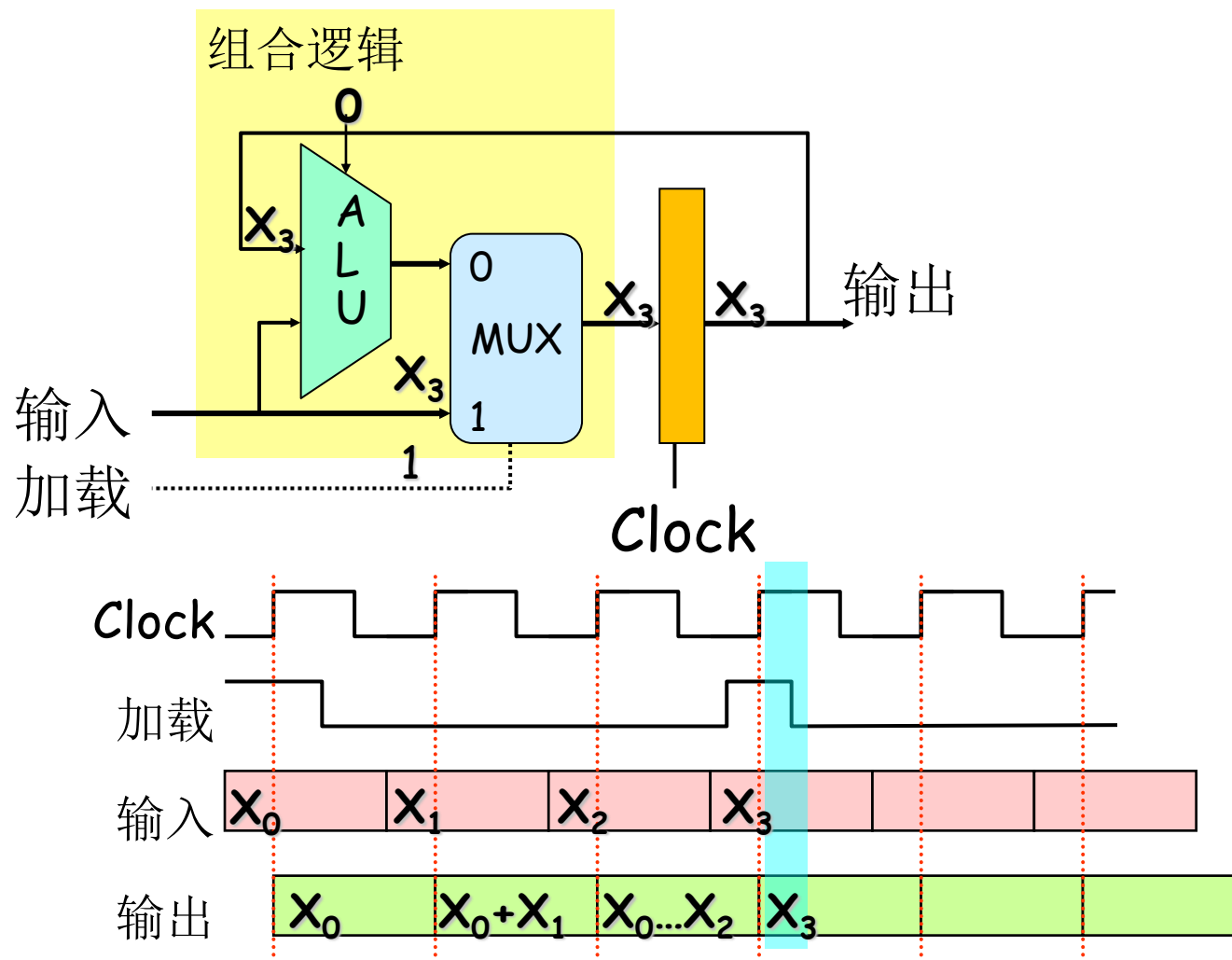
状态机示例



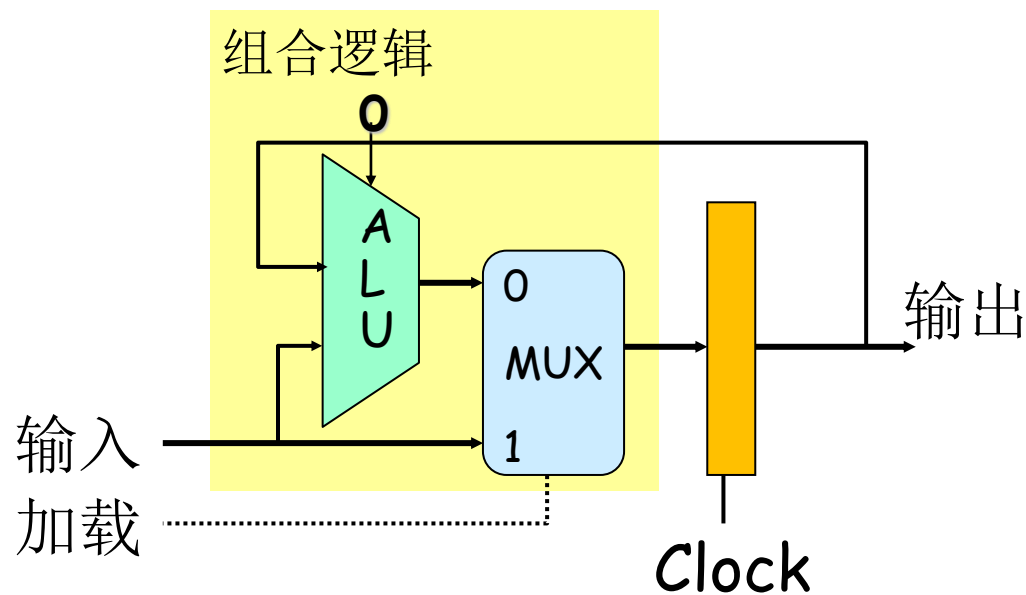
状态机示例



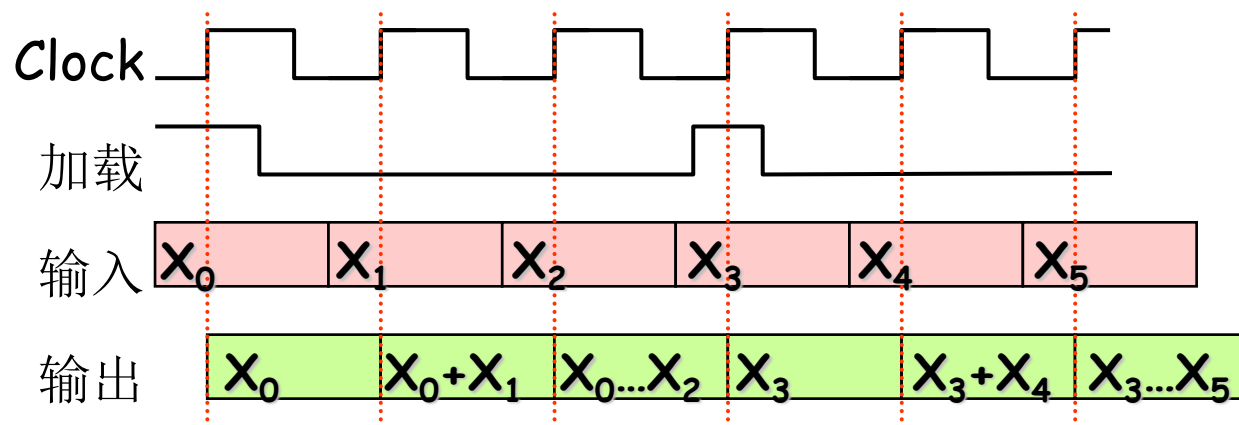
状态机示例



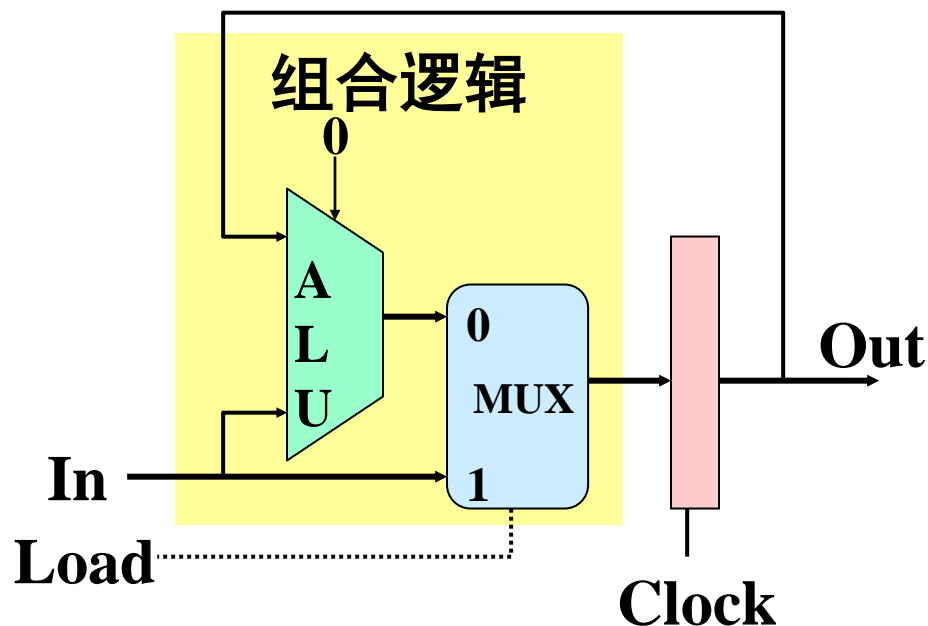
状态机示例



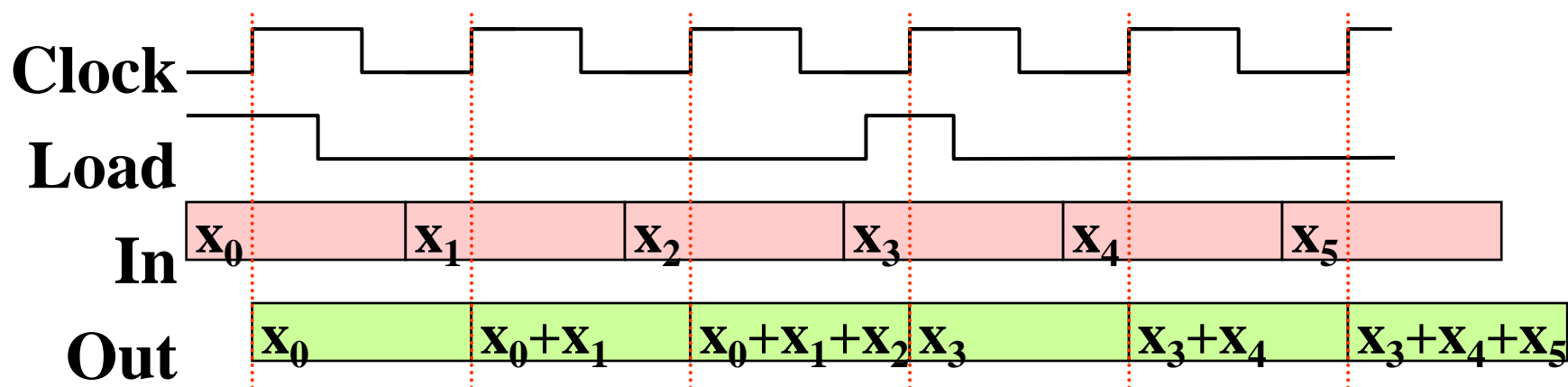
- 累加器示例
- 在每个时钟加载或累加

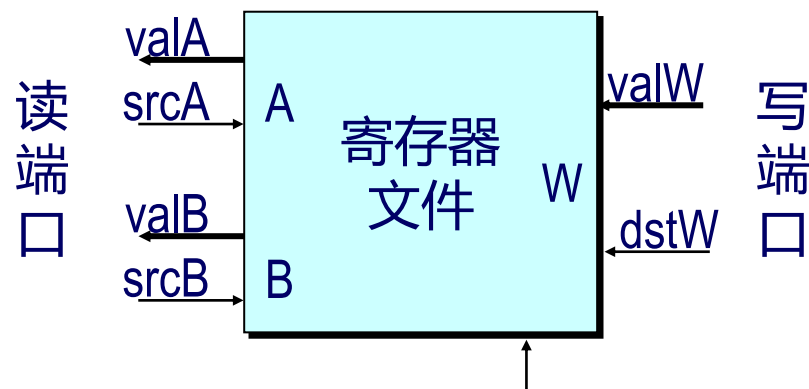
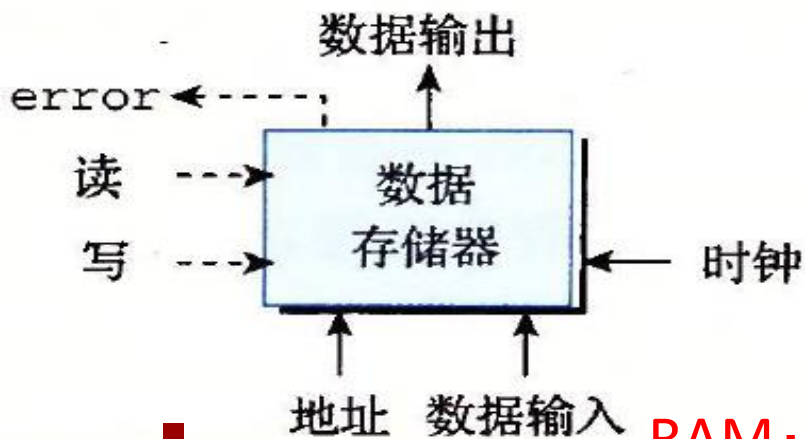


状态机示例



- 累加器电路
- 在每个时钟加载或累加





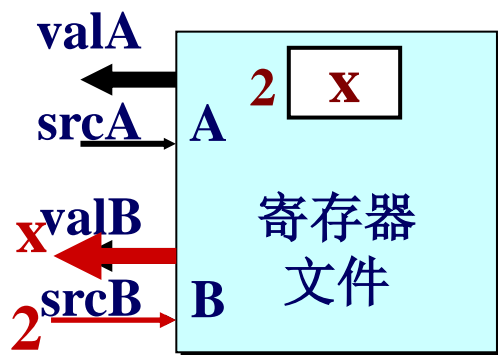
随机存取存储器

- **RAM: 存储内存中的多个字**
 - 通过输入的地址来决定读/写哪个字，写由clk控制。
- **RF: 寄存器文件**
 - **硬件寄存器**: 稳态、组合逻辑的屏障，CLK边沿触发。如: PC、CC、Stat 等
 - **程序寄存器**: 存储数据，可按ID读、写的存储单元。汇编级用 `%rax`, `%rsp`, `%r14` 等，机器级-寄存器ID标识符作为地址 (0000-1110) 15 (0xF) 表示不执行读写
- **PORTS: 多端口**
 - 在每个周期可以同时读/写多个字
 - 每个端口有单独的地址和数据输入/输出

寄存器文件时序

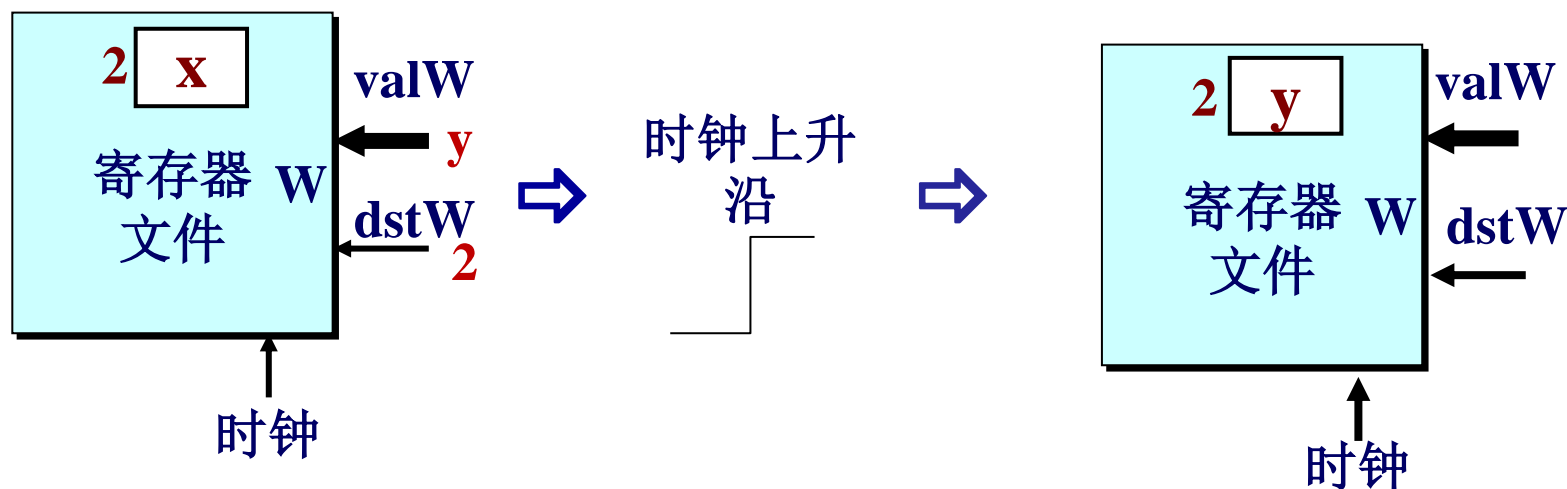
■ 读

- 类似组合逻辑
- 根据输入地址产生输出数据，
 - 有延迟



■ 写

- 类似寄存器
- 只在时钟上升沿更新



硬件控制语言(HCL)

非常简单的硬件描述语言，仅可描述硬件操作的一部分：我们将要研究和修改的部分

■ 数据类型

- bool: 布尔
 - a, b, c, ...
- int: 字
 - A, B, C, ...
 - 不指定字大小——字节, 64-位 字, ...

■ 声明

- **bool a = 布尔表达式;**
- **int A = 整数表达式;**

HCL操作

通过返回值的类型分类

■ 布尔表达式

- 逻辑操作

- $a \ \&\& \ b, a \ || \ b, !a$

- 字比较

- $A == B, A != B, A < B, A <= B, A >= B, A > B$

- 集合成员

- $A \text{ in } \{ B, C, D \}$

- 与 $A == B \ || \ A == C \ || \ A == D$ 一样

■ 字表达式

- 情况表达

- $[a : A; b : B; c : C]$

- 按顺序评估测试表达式 a, b, c, \dots

- 返回和首次成功测试对应的字表达式 A, B, C, \dots

H
C
L
程
序

```

wordsig rB      'rb'      # rB field from instruction
wordsig valC    'valc'    # Constant from instruction
wordsig valP    'valp'    # Address of following instruction
boolsig imem_error 'imem_error' # Error signal from instruction memory
boolsig instr_valid 'instr_valid' # Is fetched instruction valid?

##### Decode stage computations #####
wordsig valA    'vala'    # Value from register A port
wordsig valB    'valb'    # Value from register B port

##### Execute stage computations #####
wordsig valE    'vale'    # Value computed by ALU
boolsig Cnd     'cond'    # Branch test
|
##### Memory stage computations #####
wordsig valM    'valm'    # Value read from memory
boolsig dmem_error 'dmem_error' # Error signal from data memory

#####
# Control Signal Definitions. #
#####

##### Fetch Stage #####

# Determine instruction code
word icode = [
    imem_error: INOP;
    1: imem_icode;      # Default: get from instruction memory
];

# Determine instruction function
word ifun = [
    imem_error: FNONE;
    1: imem_ifun;      # Default: get from instruction memory
];

bool instr_valid = icode in
    { INOP, IHALT, IRRMOVQ, IIRMOVQ, IRMMOVQ, IMRMOVQ,
      IOPQ, IJXX, ICALL, IRET, IPUSHQ, IPOPQ };

# Does fetched instruction require a regid byte?
bool need_regids =
    icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
              IIRMOVQ, IRMMOVQ, IMRMOVQ };

```

总结

■ 计算

- 通过组合逻辑实现
- 计算布尔函数
- 连续地对输入变化响应

■ 存储

- 寄存器
 - 存储单字
 - 当时钟上升时加载
- 随机存取存储器
 - 存储多字
 - 可能有多个读/写端口
 - 当输入地址变化时读取字
 - 当时钟上升时写入字

Enjoy!