



北京航空航天大学计算机学院

School of Computer Science and Engineering, Beihang University

# 计算机组成-总复习 (2020级)

## 计算机组成原理课程组

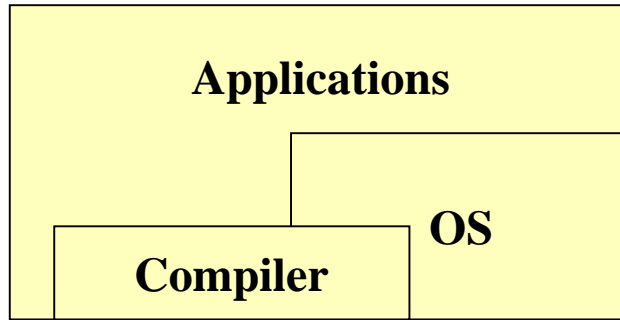
(刘旭东、肖利民、高小鹏、栾钟治、万寒)

Tel : 82316285

Mail: liuxd@buaa.edu.cn

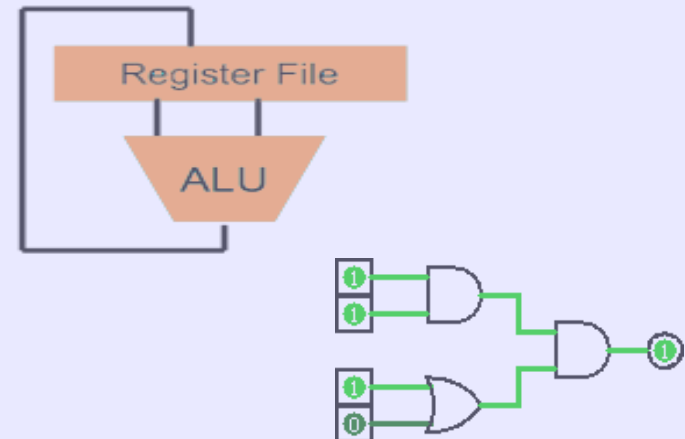
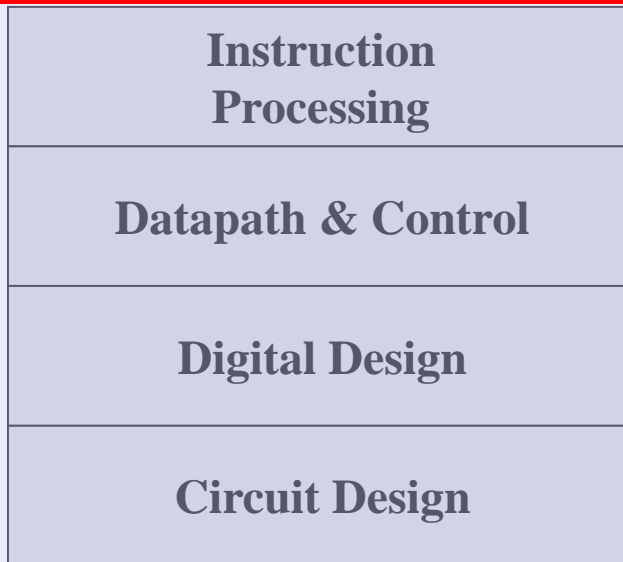
liuxd@act.buaa.edu.cn

Software  
layers of  
abstraction

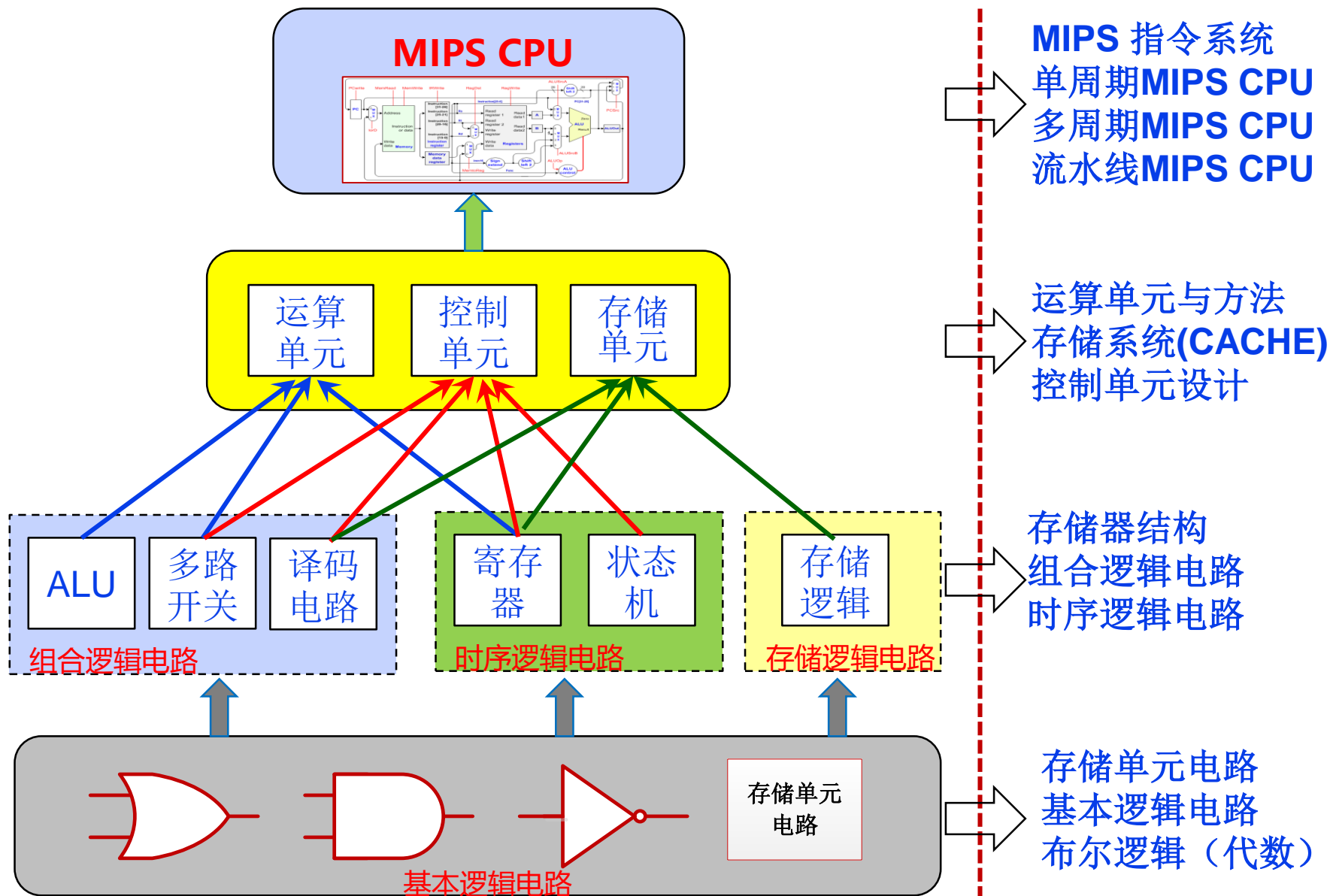


**Instruction Set Architecture (ISA)**

Hardware  
layers for  
design  
abstraction



# 课程教学内容视图






# 第一部分

## 计算机组成基础知识

- ❖ 计算机系统的基本组成与层次关系
- ❖ 计算机中数的表示（原码、反码、补码）
- ❖ 计算机的程序执行原理简介



# **第二部分**

## **布尔逻辑及组合逻辑电路**

# 内容大纲

---

## ■ 逻辑门电路

- 由晶体管和MOS管（晶体二极管、晶体三极管、NMOS、PMOS）构建门电路（与、或、非、与非、或非等）

## ■ 布尔代数

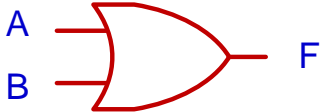
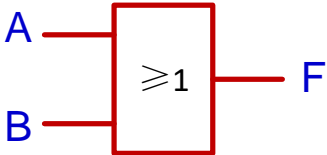
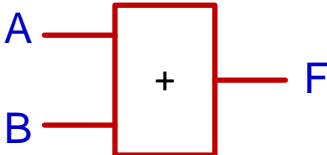

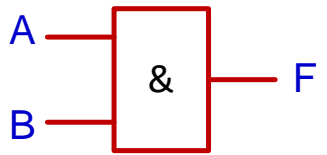
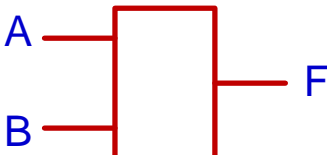
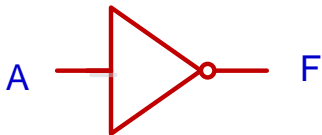
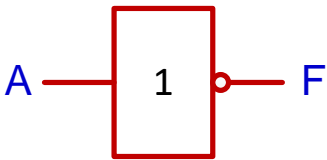
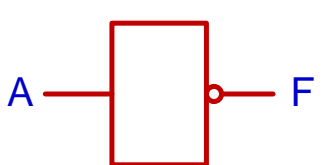
- 逻辑代数基本概念：逻辑常量/变量，典型逻辑运算
- 逻辑代数的运算法则：公理、定律、定理、基本公式及其推论
- 逻辑函数的表达式：真值表 -> 最小项表达式、最大项表达式
- 逻辑函数的简化法：合并乘积项法、吸收项法、配项法

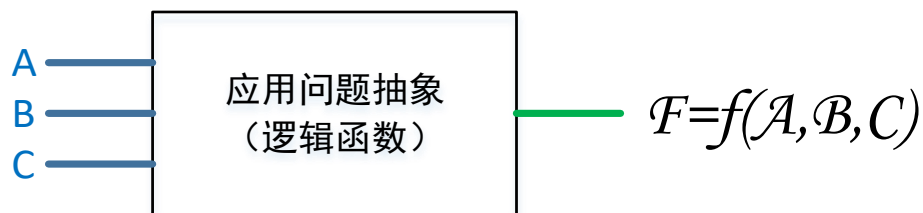
## ■ 基本组合逻辑部件设计

- 运算单元电路：加法、减法、乘法、比较器、ALU
- 编码器/译码器：三种编码器/译码器
- 多路选择器：数据选择、多功能运算

# 数字逻辑的数学基础：布尔逻辑代数

## ❖ 逻辑电路的符号表示

逻辑门	IEEE标准符号	国标符号	部标符号	逻辑表达式
或				$F = A + B$ $F = A \vee B$
与				$F = A \cdot B$ $F = A \wedge B$
非				$F = \overline{A}$ $F = \neg A$



## 逻辑函数的标准表达式:

- 最小项表达式: 最小项构成的与或式  $F(A,B,C) = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$
- 最大项表达式: 最大项构成的或与式  $F(A,B,C) = (A+B+C) \cdot (A+\overline{B}+\overline{C}) \cdot (\overline{A}+\overline{B}+\overline{C})$



## 逻辑函数表达式的化简:

- 代数化简: 利用公理、定理和规则进行化简
- 卡诺图化简: 利用卡诺图进行化简



# 最小项表达式

- ❖ 全部由最小项构成的与或式，也称标准与或式，可由最小项推导法直接从真值表中导出。

例：三人表决器设计（表决原则：少数服从多数）

A, B, C表示输入，1表示赞成，0表示反对

F表示输出，1表示通过，0表示不通过

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

$$F(A, B, C) = m_3 + m_5 + m_6 + m_7$$

$$F(A, B, C) = \sum m(3,5,6,7)$$

最小项  
表达式

真值表

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

**最小项推导法：**从真值表直接推导逻辑函数表达式

- ✓ 将输出为1的输入组合写成乘积项（与）的形式，其中取值为1的输入原变量表示，取值为0的输入用反变量表示；
- ✓ 然后把这些乘积项加（或）起来。

# 最大项表达式

❖ 全部由最大项构成的或与式，也称标准或与式，可由最大项推导法直接从真值表中导出。

真值表

例：三人表决器设计的输出表达式

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

$$F(A, B, C) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

$$F(A, B, C) = \Pi M(0, 1, 2, 4)$$

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

**最大项推导法：**从真值表直接推导逻辑函数表达式

- ✓ 将输出为0的输入组合写成和项（或）的形式，其中取值为0的输入原变量表示，取值为1的输入用反变量表示；
- ✓ 然后把这些和项乘（与）起来。

# 逻辑函数化简：代数法

## ❖ 逻辑函数化简常用方法

1. 合并乘积项法
2. 吸收项法
3. 配项法
4. 消除冗余项法

### 1、合并乘积项法：利用互补律消去1个变量

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

**解：**

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

## 2、吸收项法——利用分配律和互补律减少“与”项

化简  $F = A\bar{B} + \bar{A}B + ABCD + \bar{A}\bar{B}CD$

解：

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

提示：

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

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

### 3、配项法——利用互补律配项，配在乘积项上

化简  $F = AB + \bar{A}\bar{B}C + BC$

解：

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

# 逻辑函数化简：卡诺图化简

❖ 卡诺图：20实际50年代，美国工程师Karnaugh提出

B \ A	0	1
0	$\bar{A}\bar{B}$ $m_0$	$A\bar{B}$ $m_2$
1	$\bar{A}B$ $m_1$	$AB$ $m_3$

二变量卡诺图

C \ AB	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$ $m_0$	$\bar{A}B\bar{C}$ $m_2$	$AB\bar{C}$ $m_6$	$A\bar{B}\bar{C}$ $m_4$
1	$\bar{A}\bar{B}C$ $m_1$	$\bar{A}BC$ $m_3$	$ABC$ $m_7$	$A\bar{B}C$ $m_5$

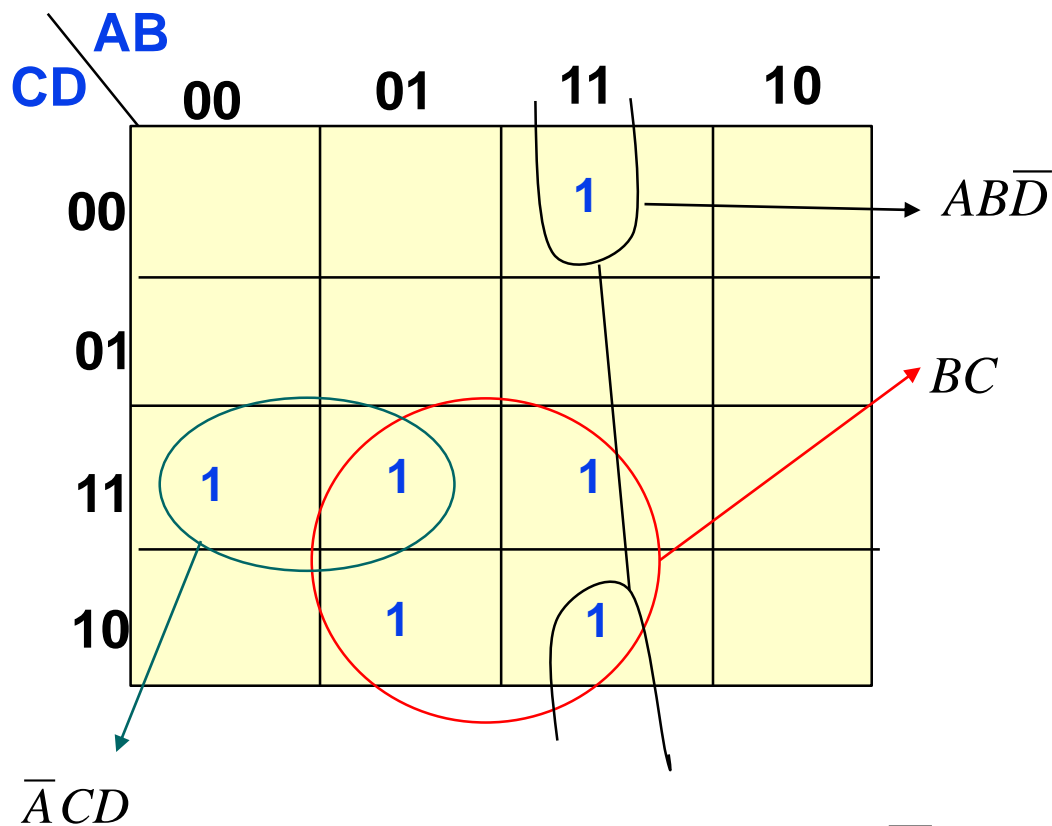
三变量卡诺图

CD \ AB	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$ $m_0$	$\bar{A}B\bar{C}\bar{D}$ $m_4$	$AB\bar{C}\bar{D}$ $m_{12}$	$A\bar{B}\bar{C}\bar{D}$ $m_8$
01	$\bar{A}\bar{B}C\bar{D}$ $m_1$	$\bar{A}BC\bar{D}$ $m_5$	$ABC\bar{D}$ $m_{13}$	$A\bar{B}C\bar{D}$ $m_9$
11	$\bar{A}\bar{B}CD$ $m_3$	$\bar{A}BCD$ $m_7$	$ABCD$ $m_{15}$	$A\bar{B}CD$ $m_{11}$
10	$\bar{A}\bar{B}C\bar{D}$ $m_2$	$\bar{A}BC\bar{D}$ $m_6$	$ABC\bar{D}$ $m_{14}$	$A\bar{B}C\bar{D}$ $m_{10}$

四变量卡诺图

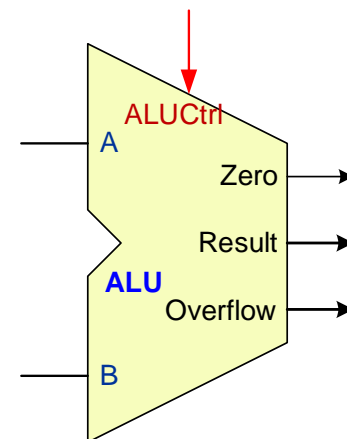
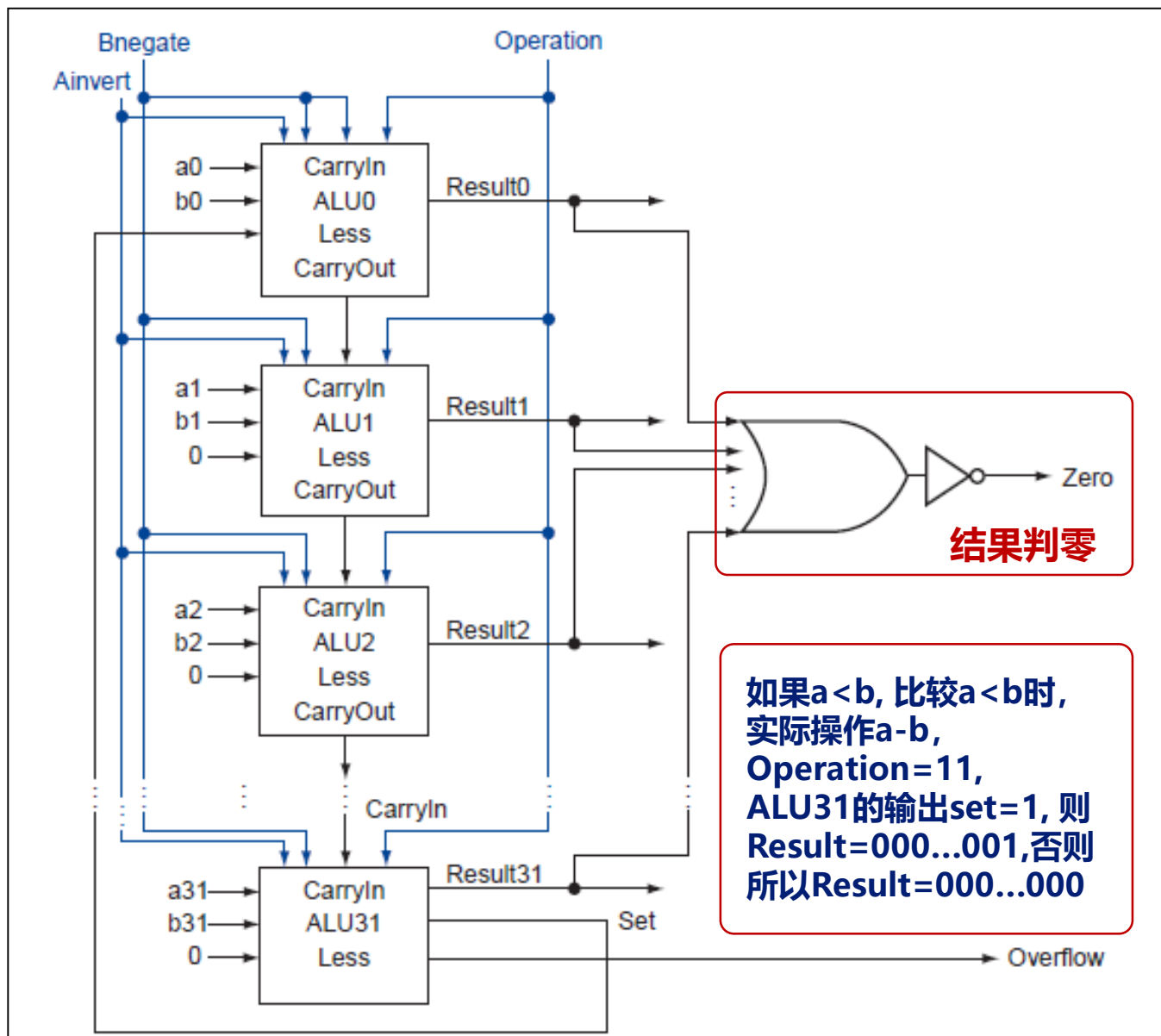
# 逻辑函数化简：卡诺图化简

$$\begin{aligned} F(ABCD) &= \bar{A}BC + BCD + \bar{A}CD + AB\bar{D} \\ &= \bar{A}BCD + \bar{A}BC\bar{D} + ABCD + \bar{A}\bar{B}CD + ABC\bar{D} + AB\bar{C}\bar{D} \\ &= \Sigma(3,6,7,12,14,15) \end{aligned}$$



$$F(ABCD) = AB\bar{D} + \bar{A}CD + BC$$

# ALU--32位ALU



ALUCtrl (4位)	输出
0000	Result = A&B
0001	Result = A B
0010	Result = A+B
0110	Result = A-B

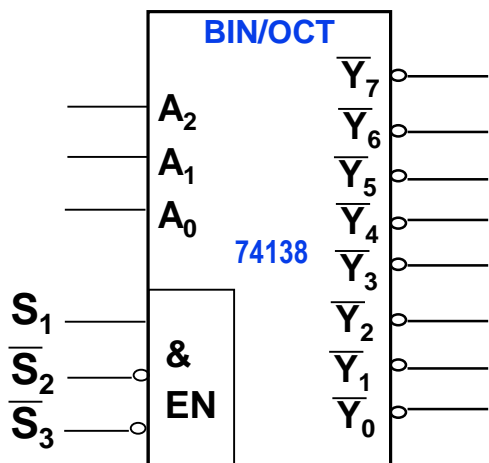
**4位ALUCtrl从高到低依次是: Ainvert, Bnegate和 Operation**



# 译码器 (3线-8线译码器)

## ❖ 3线-8线译码器 (74138)

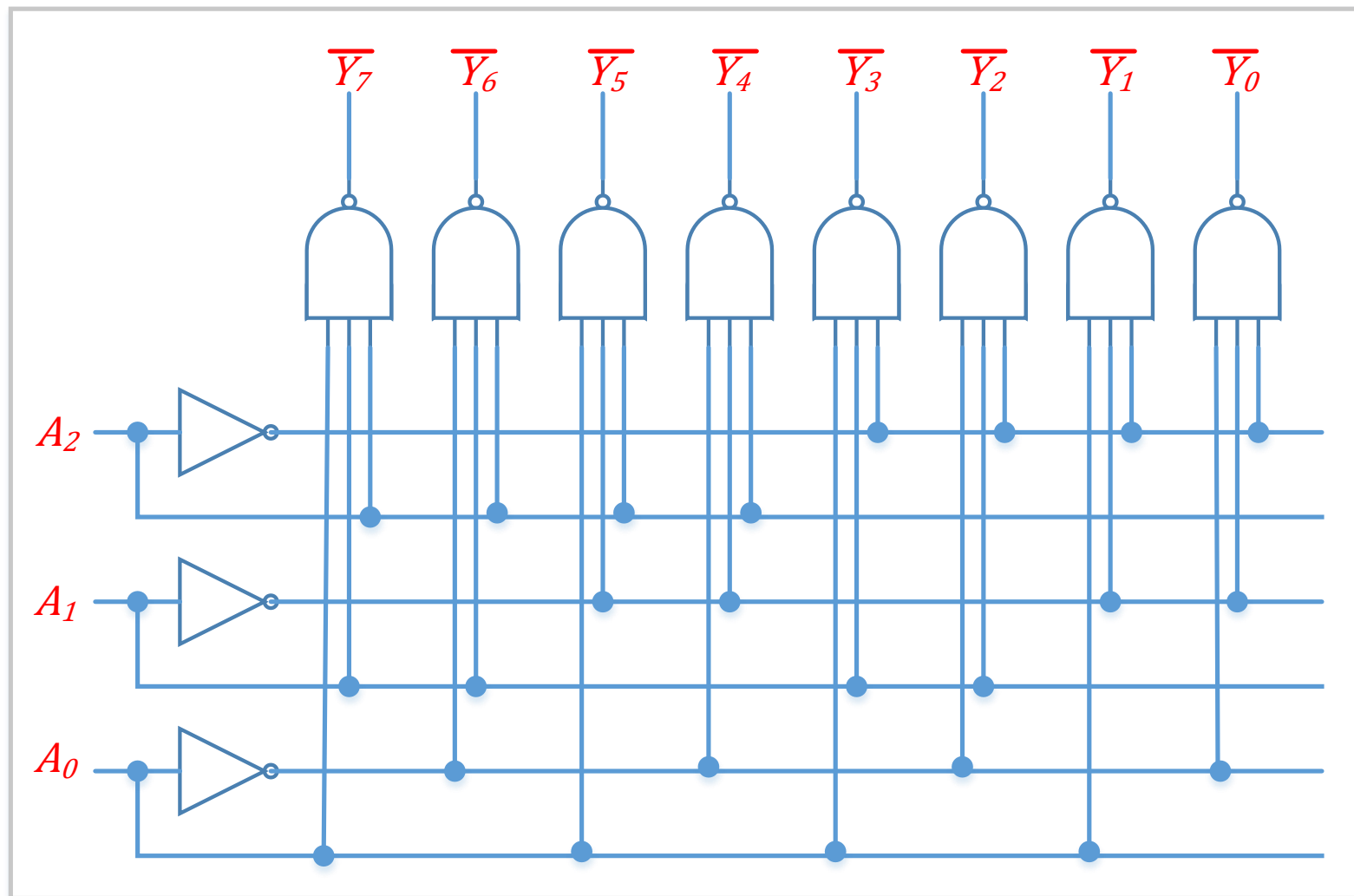
- 3个输入:  $A_2, A_1, A_0$ ; 000~111共8种输入组合。
- 8个输出:  $Y_7 \sim Y_0$ , 低电平输出有效; 任何时刻最多只有一个输出有效。当输入为000时,  $Y_0$ 输出有效; 当输入为001时,  $Y_1$ 输出有效。
- 3个使能控制:  $S_0, S_1, S_2$  为使能输入, 仅当它们分别为1、0、0时, 译码器才正常译码; 否则禁止工作。



功能表

$S_1 \overline{S_2} \overline{S_3}$	$A_2 A_1 A_0$	$\overline{Y_7} \overline{Y_6} \overline{Y_5} \overline{Y_4} \overline{Y_3} \overline{Y_2} \overline{Y_1} \overline{Y_0}$
$\neq 100$	X X X	1 1 1 1 1 1 1 1
$=100$	0 0 0	1 1 1 1 1 1 1 0
$=100$	0 0 1	1 1 1 1 1 1 0 1
$=100$	0 1 0	1 1 1 1 1 0 1 1
$=100$	0 1 1	1 1 1 1 0 1 1 1
$=100$	1 0 0	1 1 1 0 1 1 1 1
$=100$	1 0 1	1 1 0 1 1 1 1 1
$=100$	1 1 0	1 0 1 1 1 1 1 1
$=100$	1 1 1	0 1 1 1 1 1 1 1

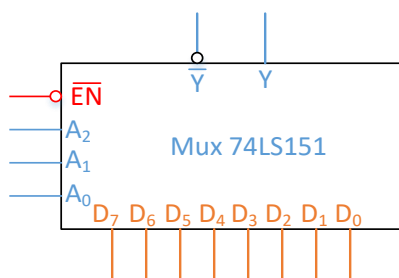
# 译码器 (3线-8线译码器)



# 多路选择器 (8选1数据选择器74151)

## ❖ 功能一：8选1数据选择器

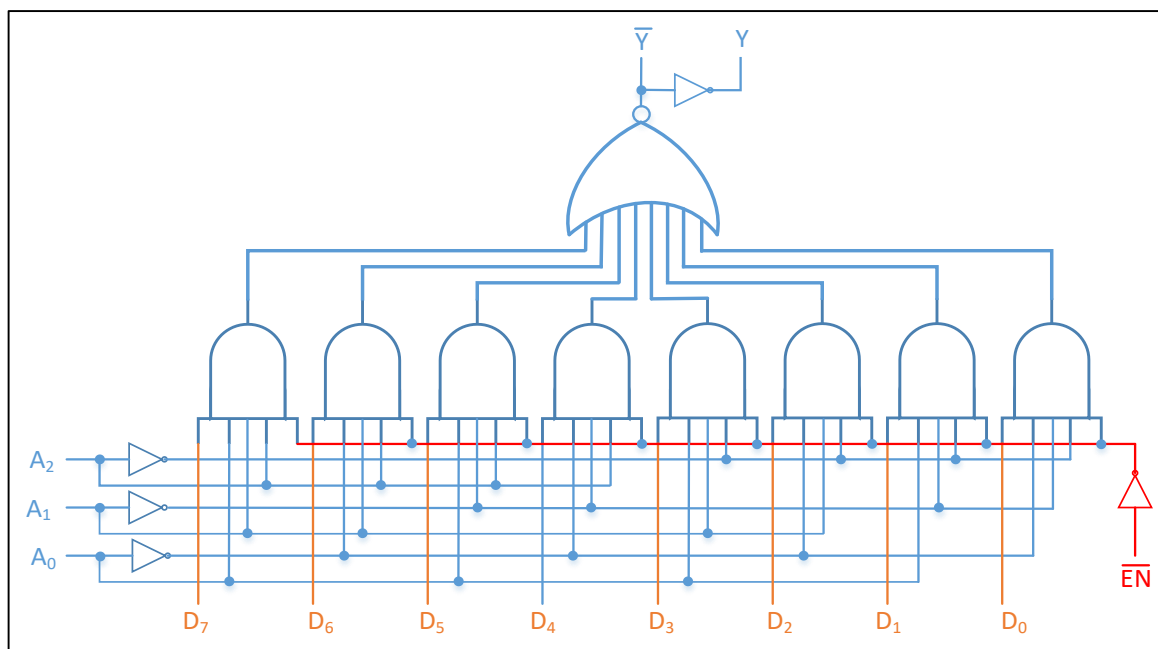
- $D_7 \sim D_0$ 为数据输入端
- $A_2A_1A_0$ 为选择控制端
- 使能控制输入 $\overline{EN}$ 低电平有效



功能表( $\overline{EN} = 0$ )

$A_2$	$A_1$	$A_0$	$Y$
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	$D_7$

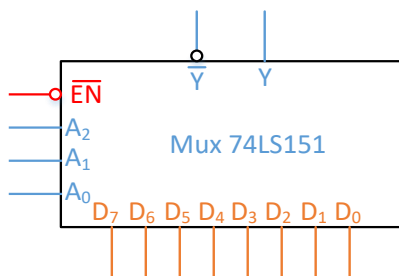
$$Y = \overline{A_2}\overline{A_1}\overline{A_0}D_0 + \overline{A_2}\overline{A_1}A_0D_1 + \overline{A_2}A_1\overline{A_0}D_2 + \overline{A_2}A_1A_0D_3 \\ + A_2\overline{A_1}\overline{A_0}D_4 + A_2\overline{A_1}A_0D_5 + A_2A_1\overline{A_0}D_6 + A_2A_1A_0D_7$$



# 多路选择器 (8选1数据选择器74151)

## ❖ 功能二：多功能运算电路

- 换一个视角看待输入信号 $D_7 \sim D_0$ 和 $A_2 \sim A_0$
- 在 $Y$ 的函数表达式中，输入变量 $A_2 \sim A_0$ 可以构成8个最小项
- $D_7 \sim D_0$ 可作为选择输入端，从8个最小项进行选取，实现不同运算0
- 8个最小项最多有 $2^8=256$ 种组合，可实现任意组合逻辑电路。



$$Y = \overline{A_2}\overline{A_1}\overline{A_0}D_0 + \overline{A_2}\overline{A_1}\overline{A_0}D_1 + \overline{A_2}\overline{A_1}\overline{A_0}D_2 + \overline{A_2}\overline{A_1}\overline{A_0}D_3 \\ + \overline{A_2}\overline{A_1}\overline{A_0}D_4 + \overline{A_2}\overline{A_1}\overline{A_0}D_5 + \overline{A_2}\overline{A_1}\overline{A_0}D_6 + \overline{A_2}\overline{A_1}\overline{A_0}D_7$$

$$Y = D_0m_0 + D_1m_1 + D_2m_2 + D_3m_3 \\ + D_4m_4 + D_5m_5 + D_6m_6 + D_7m_7$$

当 $D_7 \sim D_0$ 为00000000时， $Y=0$

当 $D_7 \sim D_0$ 为11111111时， $Y=1$

当 $D_7 \sim D_0$ 为00000001时， $Y=m_0$

当 $D_7 \sim D_0$ 为10100101时，

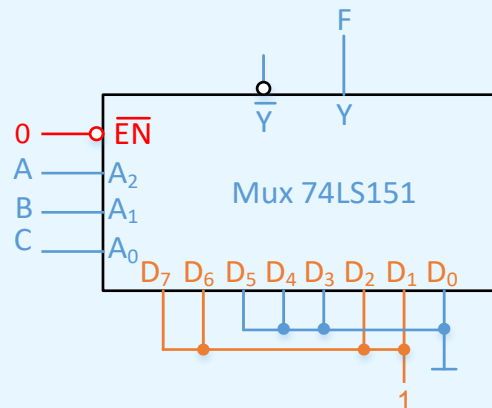
$$Y=m_7+m_5+m_2+m_0$$


例：用数据选择器实现逻辑函数：

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

$$\text{解： } F = \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB(\overline{C} + C) \\ = m_1 + m_2 + m_6 + m_7$$

则： $D_0=0, D_1=1, D_2=1, D_3=0,$   
 $D_4=0, D_5=0, D_6=1, D_7=1$





# **第三部分**

## **时序逻辑电路**

## ❖ 基本时序逻辑器件内容

### ➤ 锁存器和触发器

- SR锁存器、D锁存器
- D触发器
- JK触发器

## ❖ 有限状态机 (FSM)

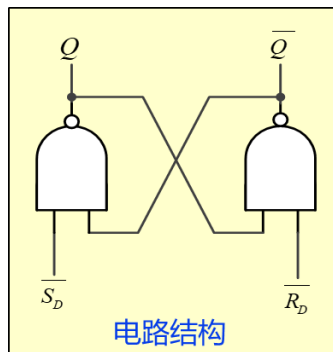
### ➤ Moore型FSM

### ➤ Mealy型FSM

## ❖ 时序逻辑电路设计分析

# 基本RS锁存器

## ❖ 基本RS锁存器



(1) 真值表(特性表)

$\overline{S_D}$	$\overline{R_D}$	$Q^n$	$Q^{n+1}$	功能
0	1	0	1	置1：次态为1
0	1	1	1	
1	0	0	0	置0：次态为0
1	0	1	0	
1	1	0	0	保持：次态不变
1	1	1	1	
0	0	0	约束条件， $\overline{S_D}$ 和 $\overline{R_D}$ 不能同时为0，即必有： $\overline{S_D} + \overline{R_D}=1$	
0	0	1		

将原态 $Q^n$ 作为输入变量列入真值表, 这种含有状态变量的真值表称为特性表

**特性方程:** 锁存器次态与原态及输入信号之间的逻辑函数表达式

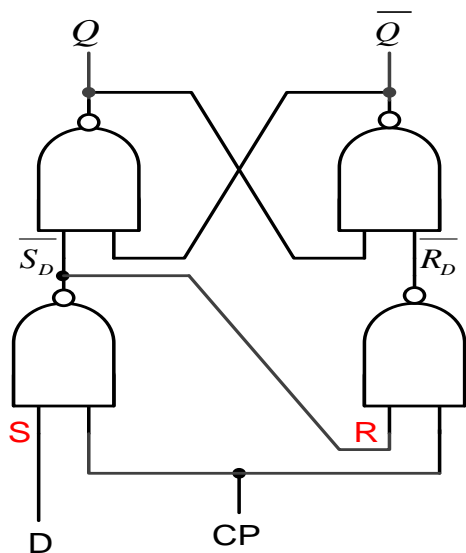
由特性表利用最小项推导法:

$$\begin{aligned}Q^{n+1} &= \overline{\overline{S_D} \overline{R_D} Q^n} + \overline{\overline{S_D} \overline{R_D} Q^n} + \overline{\overline{S_D} \overline{R_D} Q^n} \\&= \overline{\overline{S_D} \overline{R_D}} + \overline{\overline{S_D} \overline{R_D} Q^n} \\&= \overline{\overline{S_D} \overline{R_D}} + \overline{\overline{S_D} \overline{R_D} Q^n}\end{aligned}$$

利用约束条件化简:

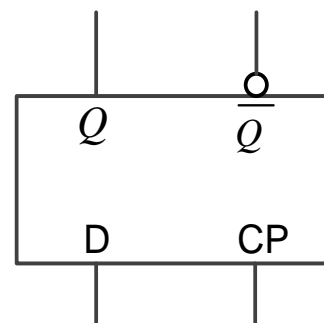
$$\begin{aligned}Q^{n+1} &= \overline{\overline{S_D} \overline{R_D}} + \overline{\overline{S_D} \overline{R_D} Q^n} \\&= (\overline{S_D} \overline{R_D} + \overline{\overline{S_D} \overline{R_D}}) (\overline{S_D} \overline{R_D} + Q^n) \\&= \overline{\overline{S_D} \overline{R_D}} (\overline{S_D} \overline{R_D} + Q^n) \\&= \overline{\overline{S_D} \overline{R_D}} + \overline{\overline{S_D} \overline{R_D} Q^n} \\&= \overline{\overline{S_D} \overline{R_D}} + \overline{\overline{S_D} \overline{R_D} Q^n} \quad \text{约束 } \overline{S_D} \overline{R_D} = 0 \\&= \overline{\overline{S_D} \overline{R_D} Q^n}\end{aligned}$$

# 钟控D锁存器



特性表 ( CP=1 )

D	$Q^n$	$Q^{n+1}$	功能
0	0	0	置0
0	1	0	
1	0	1	置1
1	1	1	



电路符号

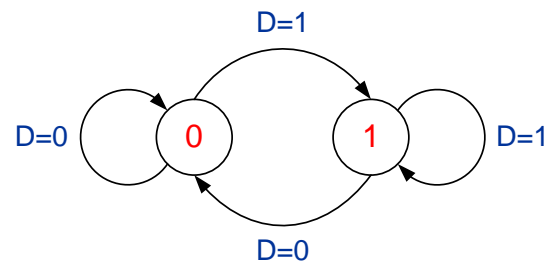
借用输入控制电路的一个与非门对D反相，作为R信号

## 特性方程

$$CP = 1 \text{ 时: } Q^{n+1} = D\bar{Q}^n + DQ^n = D$$

$$CP = 0 \text{ 时: } Q^{n+1} = Q^n$$

## CP=1 时的状态图

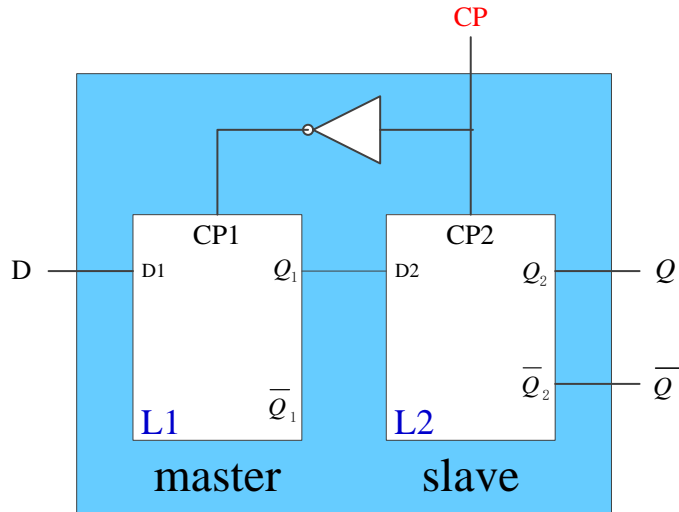




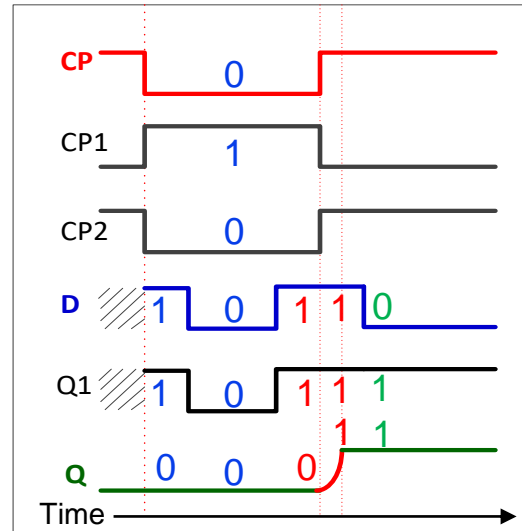
# D触发器

❖ **D触发器**：两个反相的D锁存器构成。

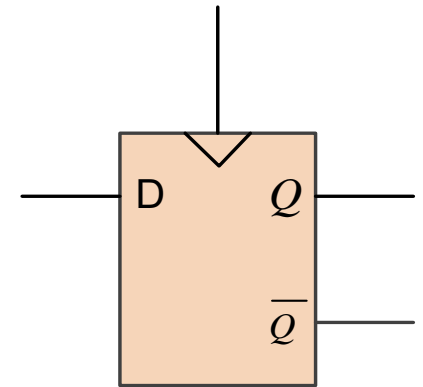
➤ 主从结构：L1为主锁存器，L2为从锁存器



电路结构



时序波形 (假定Q初态为0)



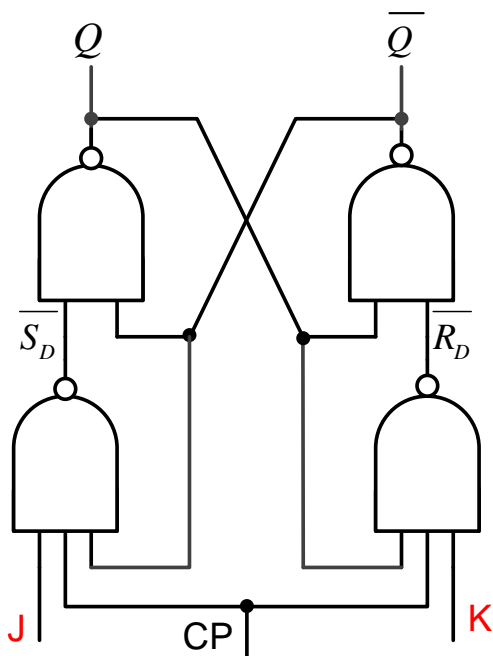
电路符号

特性方程

CP的有效沿 (上升沿或者下降沿)  $Q^{n+1} = D$

# 钟控JK触发器

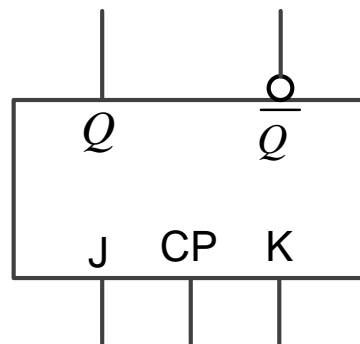
- ❖ D锁存器虽然没有约束条件，但功能较少（只有置0、置1功能）
- ❖ **JK触发器**：在钟控RS锁存器的基础上，增加两条反馈线，Q反馈到R钟控门的输入端，并把R改为K； $\bar{Q}$ 反馈到S门上，并把S改名为J。



钟控JK触发器的电路结构

## 特性方程

$$\begin{aligned} CP = 1 \text{ 时} \\ \overline{S_D} &= \overline{JQ^n}, \quad \overline{R_D} = \overline{KQ^n} \\ Q^{n+1} &= \overline{S_D} + \overline{R_D}Q^n = J\overline{Q^n} + \overline{KQ^n}Q^n \\ &= J\overline{Q^n} + \overline{K}Q^n \end{aligned}$$



电路符号

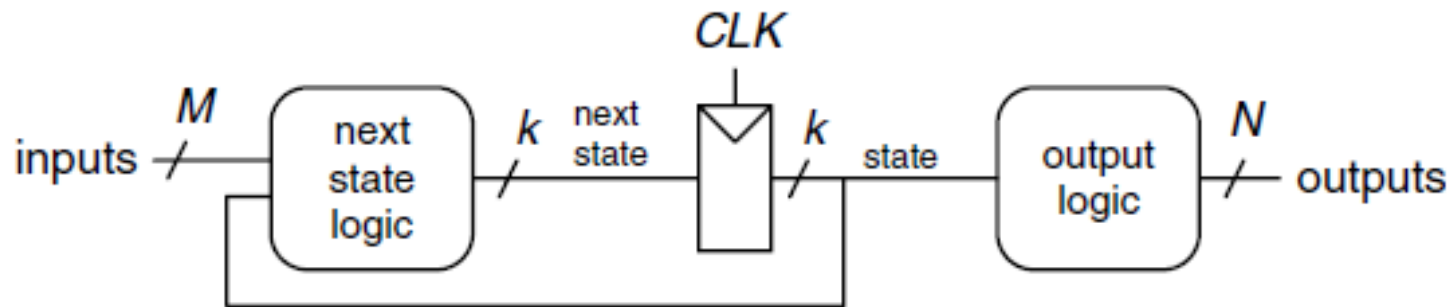
# Moore型FSM设计

## ❖ Moore型有限状态机

- **状态集合**：存在若干确定的状态（**K个状态**）；
- **对应于状态的操作集合**：对每个状态要执行某种操作（**N个参数**表示所有操作）；
- **引起状态转移的事件集合**：定时检测，当事件发生时，从一个状态转移到另一个状态（**M个参数**表示所有事件）
- **次态逻辑**：状态转移逻辑关系
- **输出逻辑**：状态到输出操作之间的逻辑关系

## ❖ Moore型FSM的设计

- 确定3个参数：M, K, N
- 设计2个逻辑：次态逻辑（状态转换图）、输出逻辑（真值表）

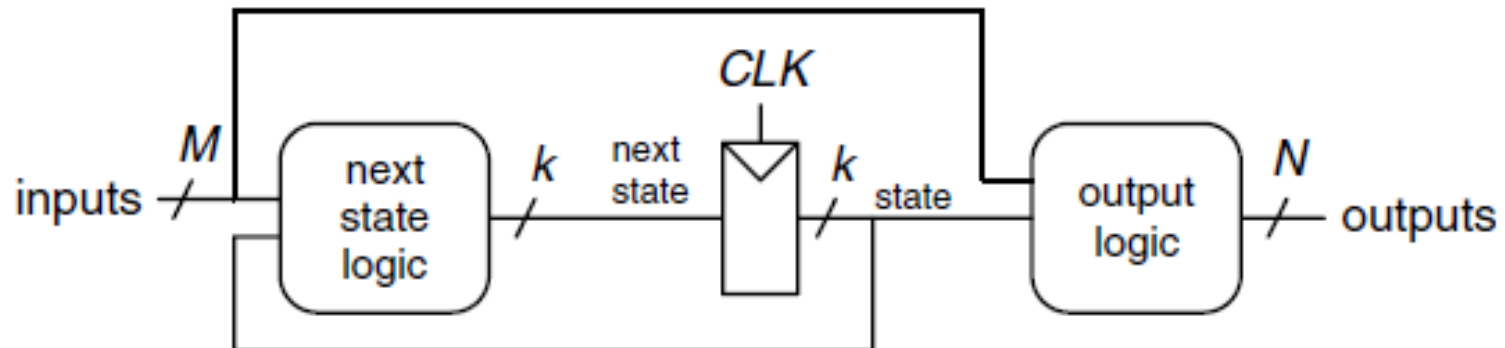


## ❖ Mealy型有限状态机

- 输出信号与当前状态及输入信号有关
- 输入M位，输出N位，状态K位（可以表示 $2^k$ 个不同状态）
- **次态逻辑**：组合逻辑，输入M+K位，输出K位次态
- **状态寄存器**：K位寄存器
- **输出逻辑**：组合逻辑，输入M+K位，输出N位信号

## ❖ Moore型FSM的设计

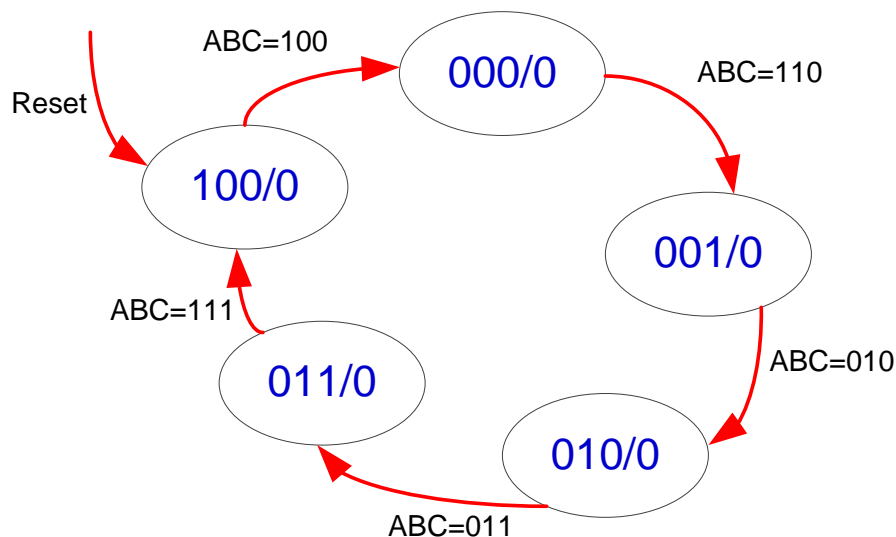
- 确定3个参数：M, K, N
- 设计2个逻辑：次态逻辑（状态转换图）、输出逻辑（真值表）



# Moore型FSM

## ❖ Moore型FSM的表示方法

- 状态图 (State Diagram) : 圆圈表示状态, 圈内 “ $Q_2Q_1Q_0 / D$ ” 分别表示状态组合 $Q_2Q_1Q_0$  (或状态编码) 及输出信号 $D$ ; 带箭头的线段表示状态转移, 线段上的文字表示转移发生时的信号输入
- 状态表 (State Table) : 状态转换表, 反映下一状态与当前状态和输入的关系



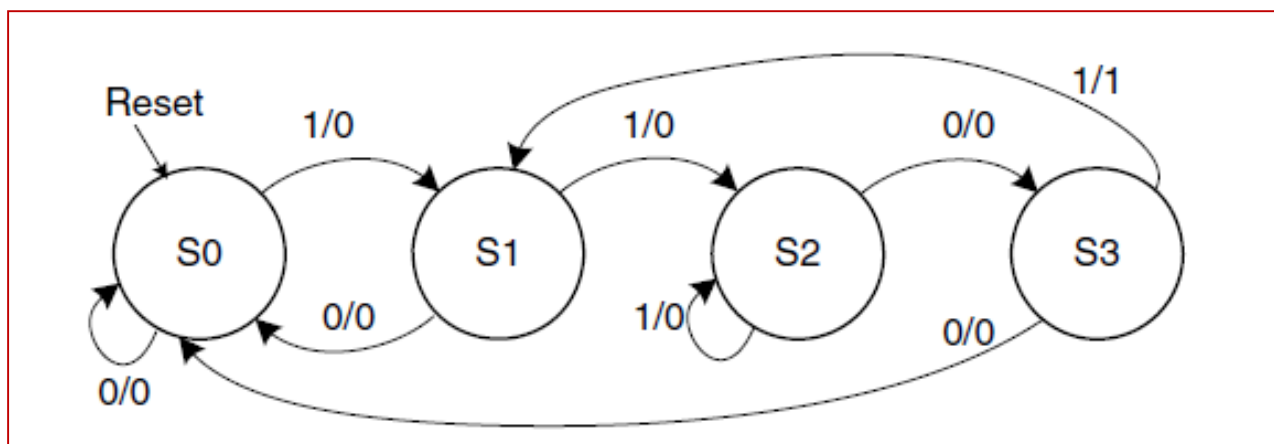
$Q_2Q_1Q_0 / D$

Moore型FSM状态转换图

# Mealy型FSM

## ❖ Mealy型FSM的表示方法 —— 状态图

- 圆圈表示状态，带箭头的线段表示状态转移
- 状态圈内 “S0”、“S1” 等代表状态名（对应状态编码）
- 与Moore型FSM不一样的是，输出信号不再标注在圈内，而是以“输入/输出”的形式标注在状态转移线上：“输入”表示引起状态转换的输入信号；“输出”表示状态转换同时产生的输出信号



Mealy型FSM状态转换图

# Moore型FSM设计

**【例】交通信号灯控制器：**南北Academic大道信号灯 $L_A$ ；东西Bravado大道信号灯 $L_B$ ，信号灯红、绿、黄三色；东西、南北大道分别安装有传感器 $T_A$ 和 $T_B$ 检测是否有人到达（等待）；信号灯控制周期5秒。

**解：**

(1) 假定 $T_A$ 和 $T_B$ 为1是表示此条路上有人到达（等待）；为0表示无人到达（等待）。

(2) 假定信号灯 $L_A$ 、 $L_B$ 编码：**00表示绿**，**01表示黄**，**10表示红**

(3) 信号灯控制器FSM模型

➤ 输入： $T_A$ 和 $T_B$ ，2位

➤ 输出： $L_A$ 和 $L_B$ ，4位

➤ 状态：共4个不同状态

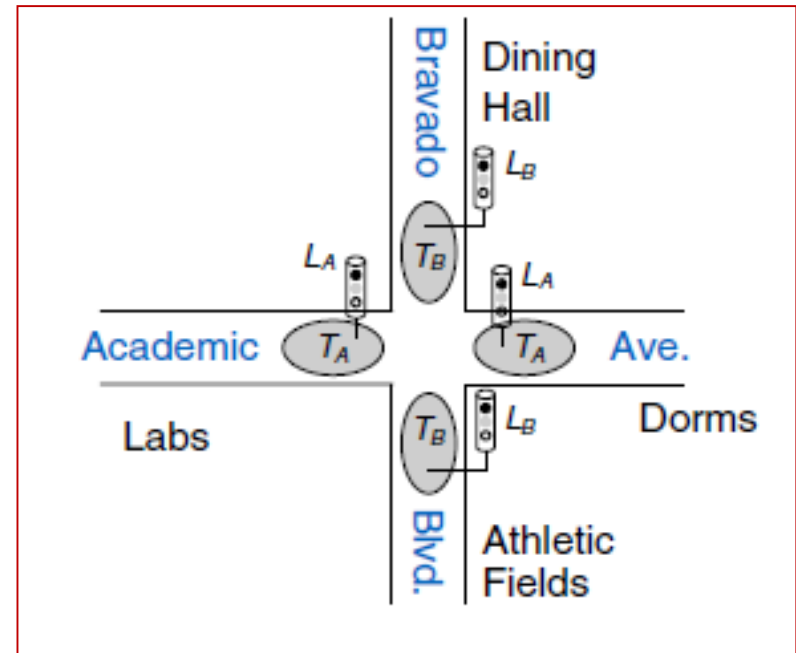
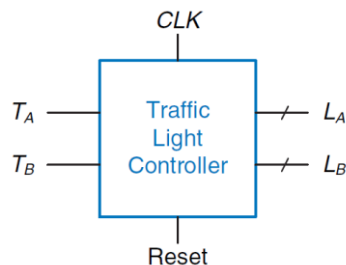
■ S0:  $L_A$ 绿 且  $L_B$ 红

■ S1:  $L_A$ 黄 且  $L_B$ 红

■ S2:  $L_A$ 红 且  $L_B$ 绿

■ S3:  $L_A$ 红 且  $L_B$ 黄

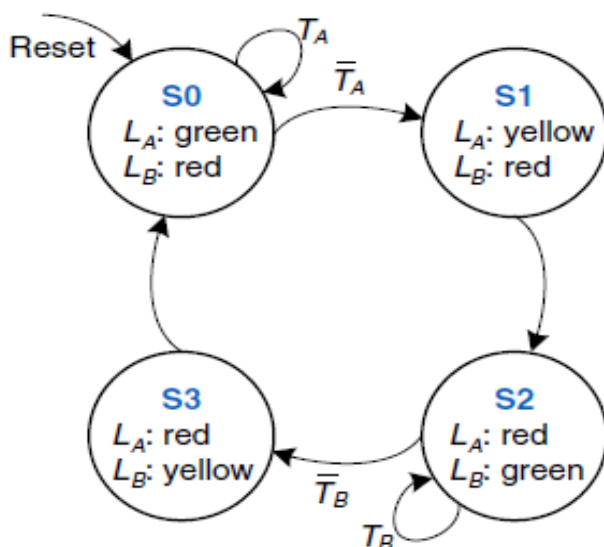
➤ 状态寄存器：2位



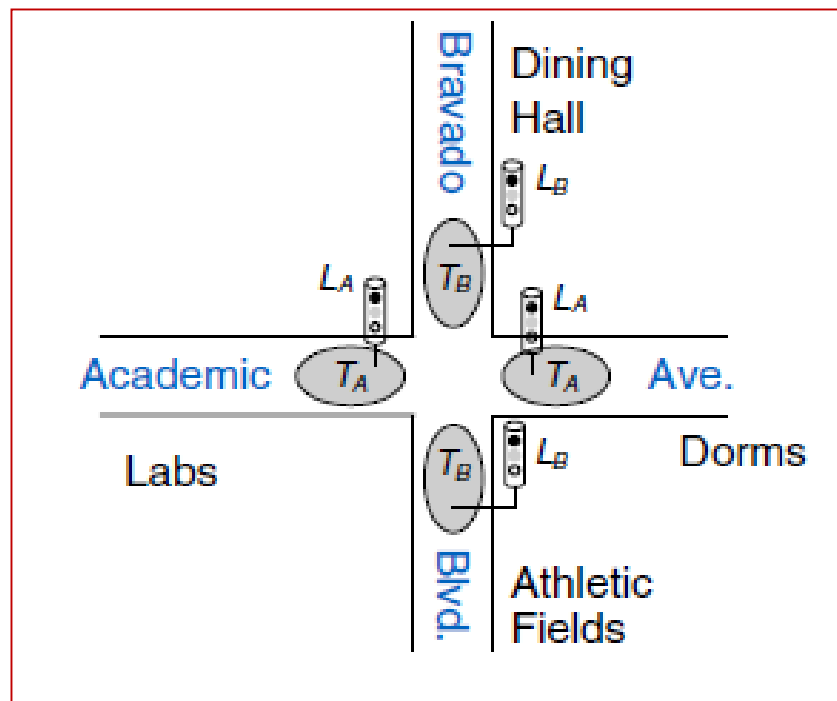
# Moore型FSM设计

解 (续1) :

(4) 控制器复位时 $L_A$ 绿、 $L_B$ 红。每隔5秒检测路口人员到达情况并决定下一步如何改变信号灯。此时若东西大道 $T_A=1$ ，信号灯不变，若 $T_A=0$ ，则 $L_A$ 变为黄并保持5秒，然后 $L_A$ 变为红且 $L_B$ 同时变为绿。南北大道按同样方式处理。得到状态转换图如下。



状态转换图





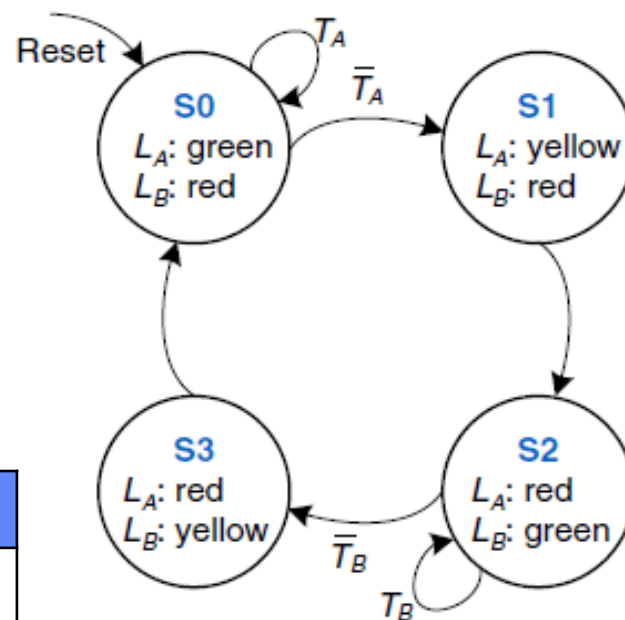
# Moore型FSM设计

解 (续2) :

(5) 根据状态转换图得到状态表和输出逻辑表。

当前状态 ( $S_1S_0$ )	输入	下一状态 ( $S'_1S'_0$ )
S0 (00)	TA=1, TB=X	S0 (00)
S0 (00)	TA=0, TB=X	S1 (01)
S1 (01)	TA=X, TB=X	S2 (10)
S2 (10)	TA=X, TB=1	S2 (10)
S2 (10)	TA=X, TB=0	S3 (11)
S3 (11)	TA=X, TB=X	S0 (00)

当前状态 ( $S_1S_0$ )	输出 $L_A$	输出 $L_B$
S0 (00)	绿 (00)	红 (10)
S1 (01)	黄 (01)	红 (10)
S2 (10)	红 (10)	绿 (00)
S3 (11)	红 (10)	黄 (01)



状态转换图

# Moore型FSM设计

解 (续3) :

(6) 根据状态表和输出逻辑表写出次态逻辑表达式和输出逻辑表达式。

$$\begin{aligned} S_1' &= \overline{S_1}S_0 + S_1\overline{S_0}T_B + S_1\overline{S_0}\overline{T_B} \\ &= \overline{S_1}S_0 + S_1\overline{S_0} \\ &= S_0 \oplus S_1 \end{aligned}$$

$$S_0' = \overline{S_1}\overline{S_0}T_A + S_1\overline{S_0}\overline{T_B}$$

$$L_{A1} = S_1 \quad L_{A0} = \overline{S_1}S_0$$

$$L_{B1} = \overline{S_1} \quad L_{B0} = S_1S_0$$

当前状态 ( <b>S<sub>1</sub>S<sub>0</sub></b> )	输入	下一状态 ( <b>S'<sub>1</sub>S'<sub>0</sub></b> )
S0 ( <b>00</b> )	TA=1, TB=X	S0 ( <b>00</b> )
S0 ( <b>00</b> )	TA=0, TB=X	S1 ( <b>01</b> )
S1 ( <b>01</b> )	TA=X, TB=X	S2 ( <b>10</b> )
S2 ( <b>10</b> )	TA=X, TB=1	S2 ( <b>10</b> )
S2 ( <b>10</b> )	TA=X, TB=0	S3 ( <b>11</b> )
S3 ( <b>11</b> )	TA=X, TB=X	S0 ( <b>00</b> )

当前状态 ( <b>S<sub>1</sub>S<sub>0</sub></b> )	输出L <sub>A</sub>	输出L <sub>B</sub>
S0 ( <b>00</b> )	绿 ( <b>00</b> )	红 ( <b>10</b> )
S1 ( <b>01</b> )	黄 ( <b>01</b> )	红 ( <b>10</b> )
S2 ( <b>10</b> )	红 ( <b>10</b> )	绿 ( <b>00</b> )
S3 ( <b>11</b> )	红 ( <b>10</b> )	黄 ( <b>01</b> )

# Moore型FSM设计

解 (续4) :

(7) 根据逻辑表达式画出逻辑电路图。

$$L_{A1} = S_1 \quad L_{A0} = \overline{S_1} S_0$$

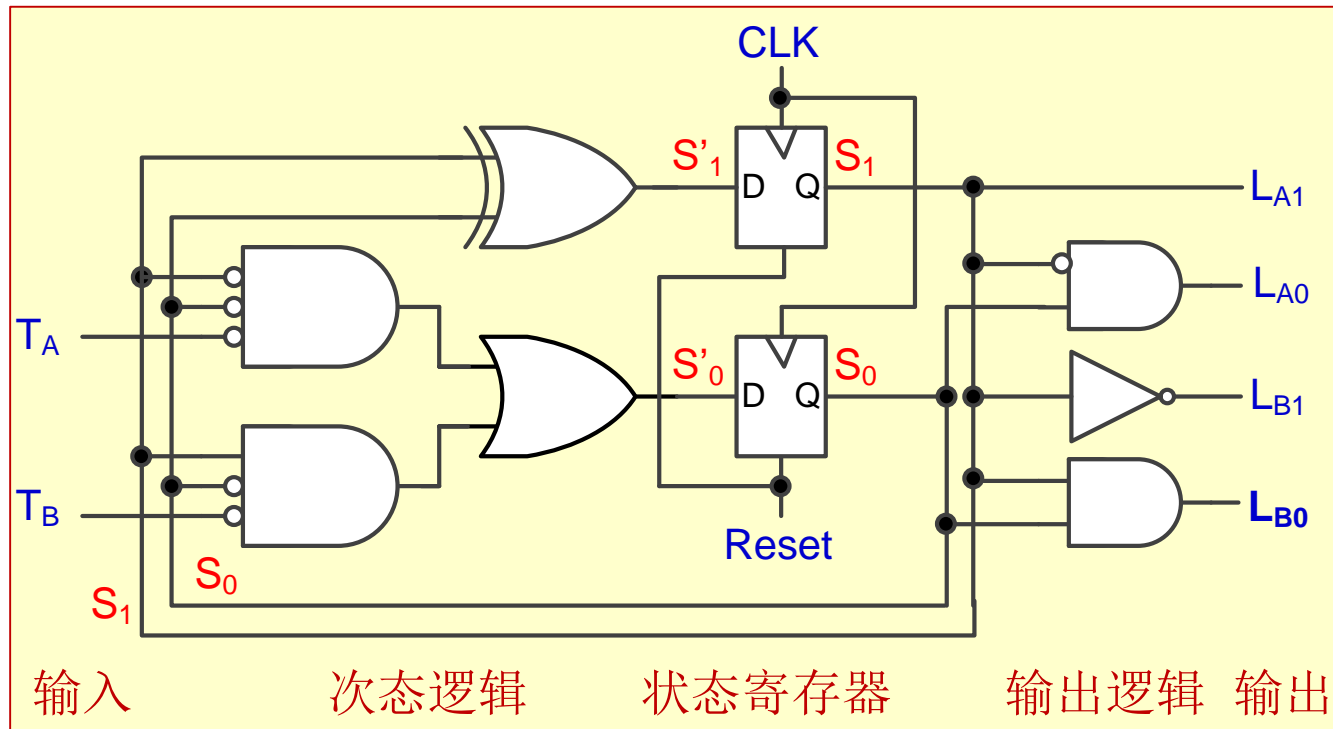
$$L_{B1} = \overline{S_1} \quad L_{B0} = S_1 S_0$$

$$S'_1 = \overline{S_1} S_0 + S_1 \overline{S_0} T_B + S_1 \overline{S_0} \overline{T_B}$$

$$= \overline{S_1} S_0 + S_1 \overline{S_0}$$

$$= S_0 \oplus S_1$$

$$S'_0 = \overline{S_1} \overline{S_0} T_A + S_1 \overline{S_0} \overline{T_A}$$



信号灯控制器状态机逻辑电路图

**【例】二进制序列检测器：**检测器接收到二进制序列“1101”时，输出检测标志为1，否则输出检测标志为0。**不重复检测，即收到1101输出1后，下一次从下一个输入信号开始检测。**

**解：**

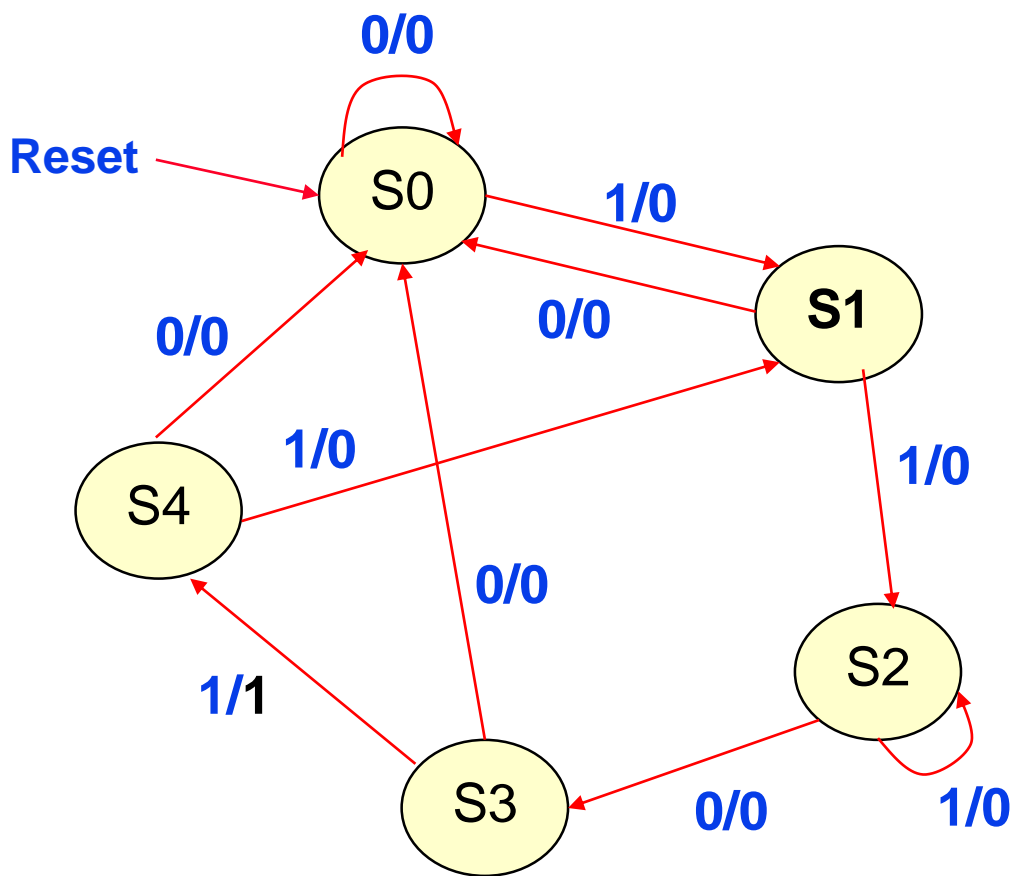
(1) 检测器FSM模型

- 输入： 二进制输入信号 A， 1位
- 输出： 检测标志信号 Y， 1位
- 状态： 共5个不同状态
  - S0： 未收到第一个有效位（输入为0， 输出0）
  - S1： 收到第一个有效位（输入为1， 输出0）
  - S2： 收到第二个有效位（即S1后输入为1， 输出0）
  - S3： 收到第三个有效位（即S2后输入为0， 输出0）
  - S4： 连续收到四个有效位（即S3后输入为1， 输出1）
- 状态寄存器： 3位

# Mealy型FSM设计

解（续1）：

（2）画出状态转换图

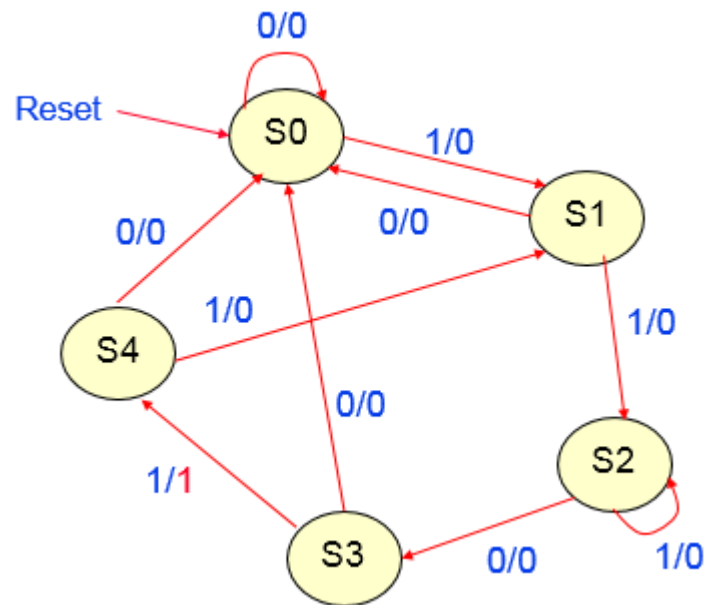


# Mealy型FSM设计

解（续2）：

（3）根据状态转换图得到状态转换表。

当前状态 ( $S_2S_1S_0$ )	输入 (A)	下一状态 ( $S'_2S'_1S'_0$ )	输出 (Y)
S0 (000)	0	S0 (000)	0
S0 (000)	1	S1 (001)	0
S1 (001)	0	S0 (000)	0
S1 (001)	1	S2 (010)	0
S2 (010)	0	S3 (011)	0
S2 (010)	1	S2 (010)	0
S3 (011)	0	S0 (000)	0
S3 (011)	1	S4 (100)	1
S4 (100)	0	S0 (000)	0
S4 (100)	1	S1 (001)	0



# Mealy型FSM设计

解（续3）：

（4）根据状态转换表写出次逻辑和输出逻辑表达式。

$$S_2' = \overline{S_2} S_1 S_0 A$$

$$\begin{aligned} S_1' &= \overline{S_2} \overline{S_1} S_0 A + \overline{S_2} S_1 \overline{S_0} A + \overline{S_2} S_1 \overline{S_0} A \\ &= \overline{S_2} \overline{S_1} S_0 A + \overline{S_2} S_1 \overline{S_0} \end{aligned}$$

$$\begin{aligned} S_0' &= \overline{S_2} \overline{S_1} S_0 A + \overline{S_2} S_1 \overline{S_0} A + S_2 \overline{S_1} \overline{S_0} A \\ &= \overline{S_1} \overline{S_0} A + \overline{S_2} S_1 \overline{S_0} A \end{aligned}$$

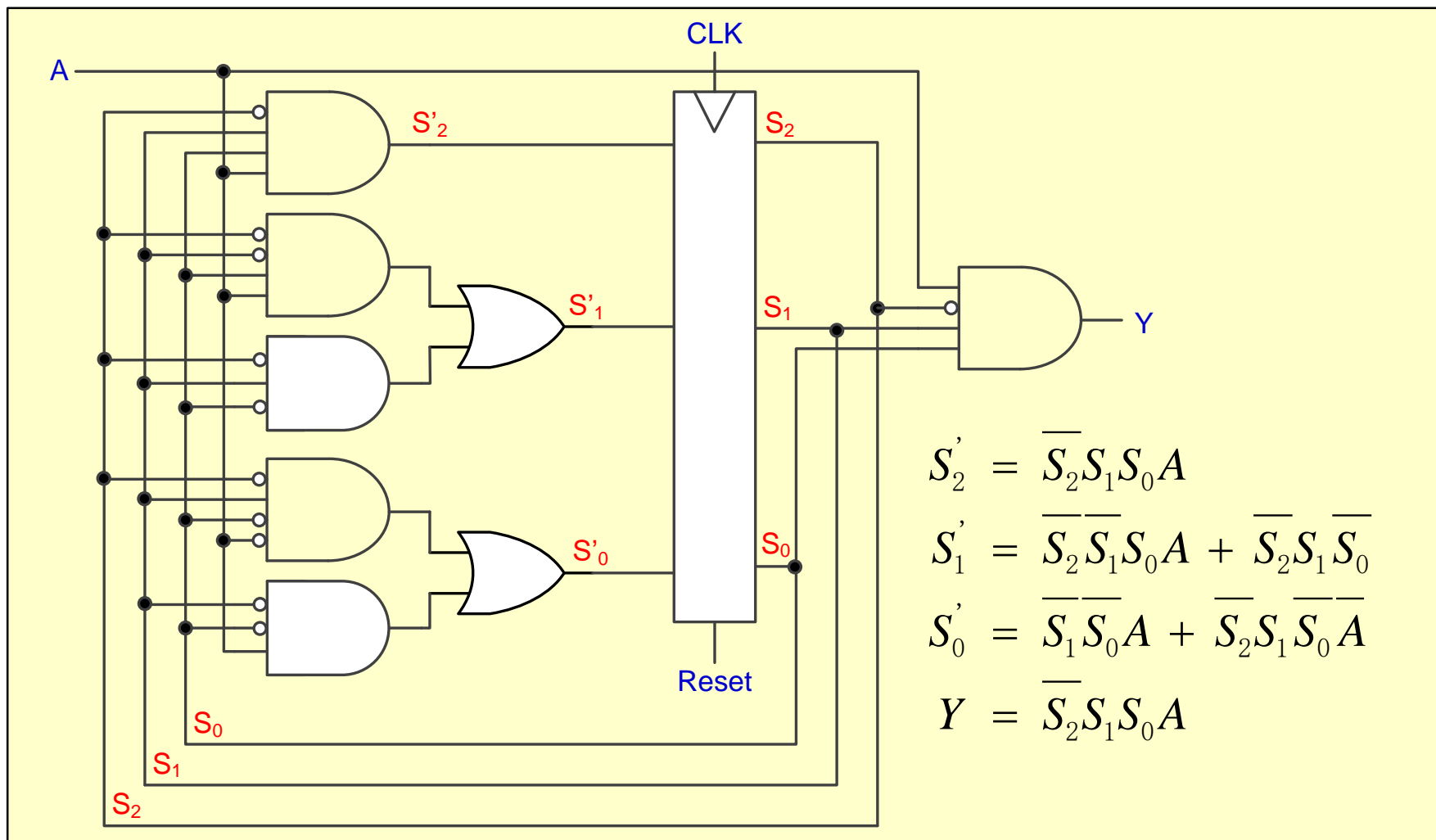
$$Y = \overline{S_2} S_1 S_0 A$$

当前状态 ( <b>S<sub>2</sub>S<sub>1</sub>S<sub>0</sub></b> )	输入 ( <b>A</b> )	下一状态 ( <b>S'<sub>2</sub>S'<sub>1</sub>S'<sub>0</sub></b> )	输出 ( <b>Y</b> )
S0 ( <b>000</b> )	<b>0</b>	S0 ( <b>000</b> )	<b>0</b>
S0 ( <b>000</b> )	<b>1</b>	S1 ( <b>001</b> )	<b>0</b>
S1 ( <b>001</b> )	<b>0</b>	S0 ( <b>000</b> )	<b>0</b>
S1 ( <b>001</b> )	<b>1</b>	S2 ( <b>010</b> )	<b>0</b>
S2 ( <b>010</b> )	<b>0</b>	S3 ( <b>011</b> )	<b>0</b>
S2 ( <b>010</b> )	<b>1</b>	S2 ( <b>010</b> )	<b>0</b>
S3 ( <b>011</b> )	<b>0</b>	S0 ( <b>000</b> )	<b>0</b>
S3 ( <b>011</b> )	<b>1</b>	S4 ( <b>100</b> )	<b>1</b>
S4 ( <b>100</b> )	<b>0</b>	S0 ( <b>000</b> )	<b>0</b>
S4 ( <b>100</b> )	<b>1</b>	S1 ( <b>001</b> )	<b>0</b>

# Mealy型FSM设计

解（续4）：

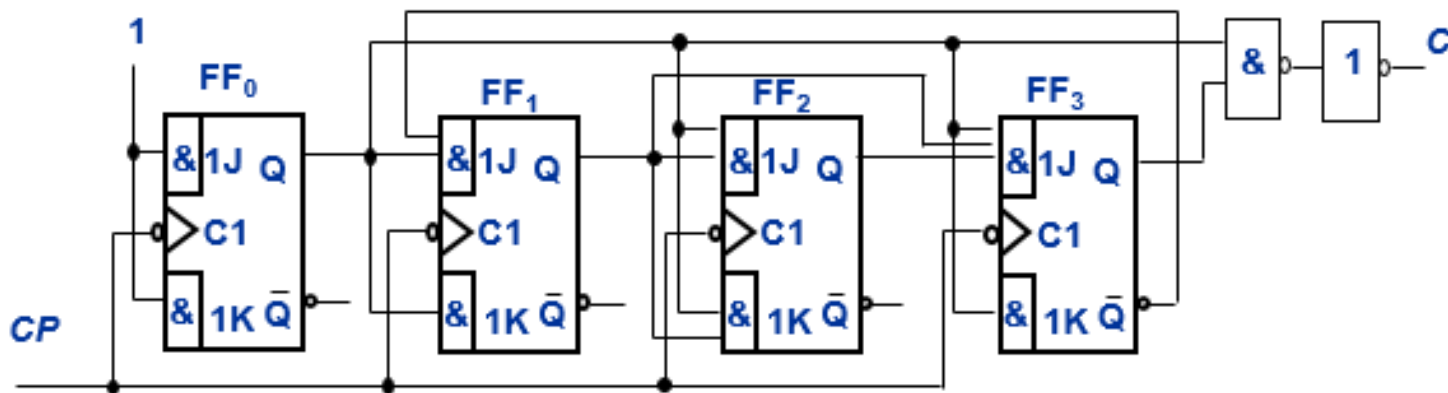
（4）根据逻辑表达式画出逻辑图。





# 同步计数器

【例1】分析下图电路，说明电路的特点。



解：(1) 写出逻辑表达式（4个负边沿触发的JK触发器组成的电路）

$$J_0 = K_0 = 1;$$

$$J_1 = \overline{Q_3}^n Q_0^n, K_1 = Q_0^n;$$

$$J_2 = K_2 = Q_1^n Q_0^n;$$

$$J_3 = \underline{Q_2^n Q_1^n Q_0^n}, K_3 = Q_0^n$$

$$C = Q_3^n Q_0^n = Q_3^n Q_0^n$$

$$Q_0^{n+1} = J_0 \overline{Q_0}^n + \overline{K_0} Q_0^n = \overline{Q_0}^n$$

$$Q_1^{n+1} = J_1 \overline{Q_1}^n + \overline{K_1} Q_1^n = \overline{Q_3}^n Q_0^n \overline{Q_1}^n + \overline{Q_0}^n Q_1^n$$

$$Q_2^{n+1} = J_2 \overline{Q_2}^n + \overline{K_2} Q_2^n = Q_1^n Q_0^n \overline{Q_2}^n + \overline{Q_1}^n \overline{Q_0}^n Q_2^n$$

$$Q_3^{n+1} = J_3 \overline{Q_3}^n + \overline{K_3} Q_3^n = Q_2^n Q_1^n Q_0^n \overline{Q_3}^n + \overline{Q_0}^n Q_3^n$$

$$CP_0 = CP_1 = CP_2 = CP_3 = CP \downarrow \text{——同步电路}$$

# 同步计数器

解（续）：（2）写出状态转换表

状态转换表

$Q_3^n Q_2^n Q_1^n Q_0^n$	$Q_3^{n+1} Q_2^{n+1} Q_1^{n+1} Q_0^{n+1}$	C
0 0 0 0	0 0 0 1	0
0 0 0 1	0 0 1 0	0
0 0 1 0	0 0 1 1	0
0 0 1 1	0 1 0 0	0
0 1 0 0	0 1 0 1	0
0 1 0 1	0 1 1 0	0
0 1 1 0	0 1 1 1	0
0 1 1 1	1 0 0 0	0
1 0 0 0	1 0 0 1	0
1 0 0 1	0 0 0 0	1
1 0 1 0	1 0 1 1	0
1 0 1 1	0 1 0 0	1
1 1 0 0	1 1 0 1	0
1 1 0 1	0 1 0 0	1
1 1 1 0	1 1 1 1	0
1 1 1 1	0 0 0 0	1

$$Q_0^{n+1} = \overline{Q_0^n}$$

$$Q_1^{n+1} = \overline{Q_3^n} Q_0^n \overline{Q_1^n} + \overline{Q_0^n} Q_1^n$$

$$Q_2^{n+1} = Q_1^n Q_0^n \overline{Q_2^n} + \overline{Q_1^n} Q_0^n Q_2^n$$

$$Q_3^{n+1} = Q_2^n Q_1^n Q_0^n \overline{Q_3^n} + \overline{Q_0^n} Q_3^n$$

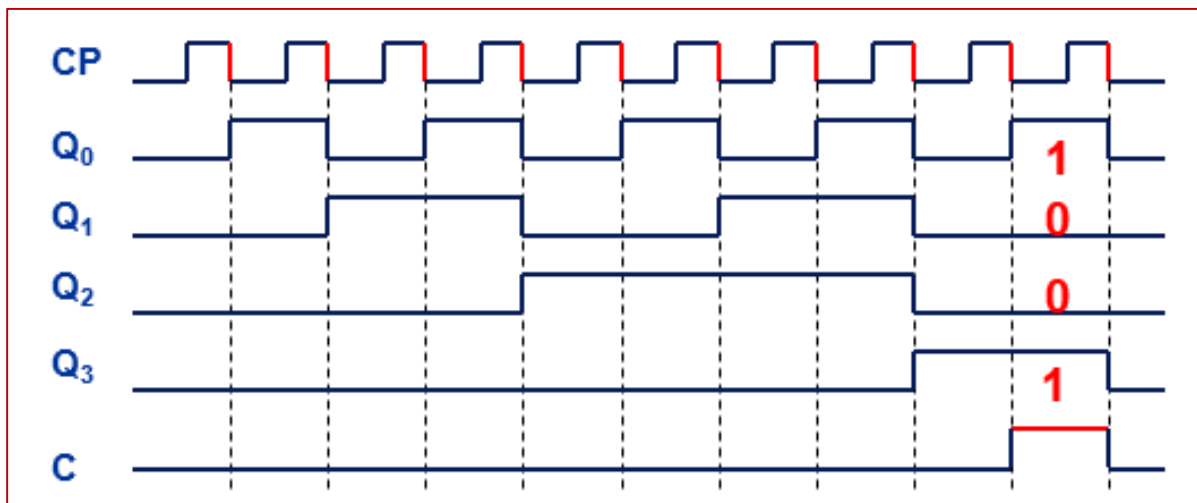
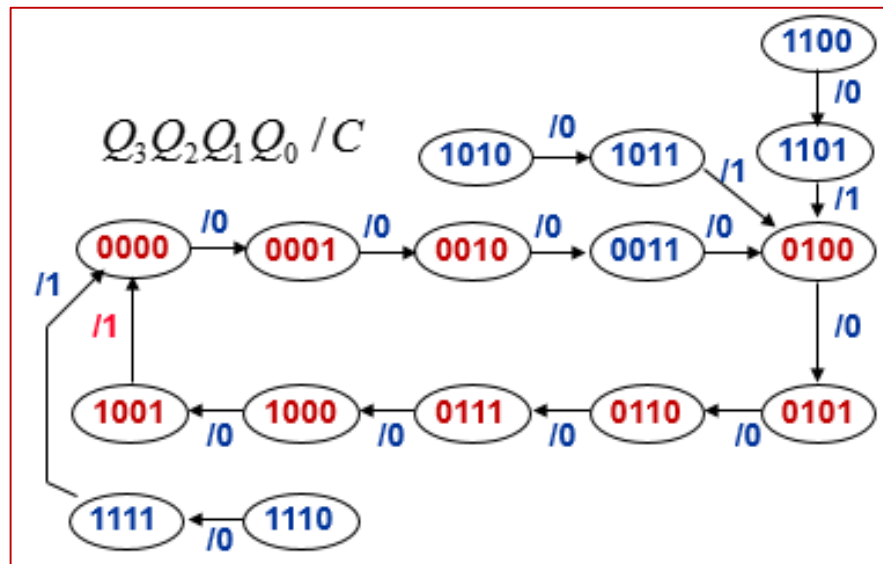
$$C = Q_3^n Q_0^n$$

# 同步计数器

解（续）：

(3) 画出状态转移图（时序图）

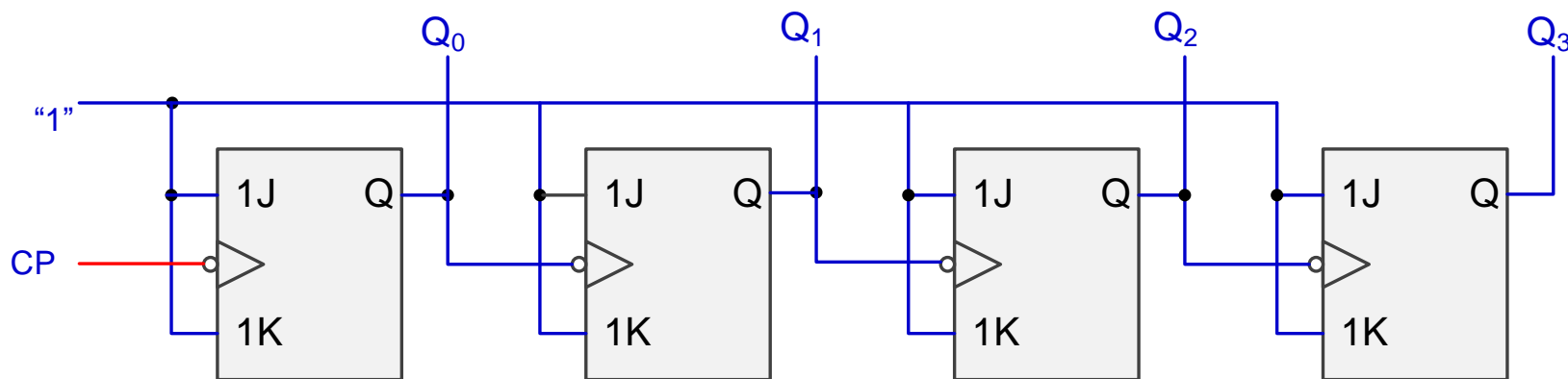
- 画状态转换图时一定要画出全部状态的变化。
- 画时序图时只画出有效状态构成的计数循环的变化；注意触发器的时钟特性！



**CP下降沿时  
触发器翻转！**

当 $Q_3Q_2Q_1Q_0=1001$ 时，**C=1**；下一个CP下降沿到来时， $Q_3Q_2Q_1Q_0$ 变为**0000**，完成一个计数循环。

## 【例3】分析下图异步二进制（M=16）加法计数器电路（N=4）



### (1) 状态方程

$$Q_0^{n+1} = \overline{Q_0^n} \cdot CP \downarrow; Q_1^{n+1} = \overline{Q_1^n} \cdot Q_0 \downarrow;$$

$$Q_2^{n+1} = \overline{Q_2^n} \cdot Q_1 \downarrow; Q_3^{n+1} = \overline{Q_3^n} \cdot Q_2 \downarrow;$$

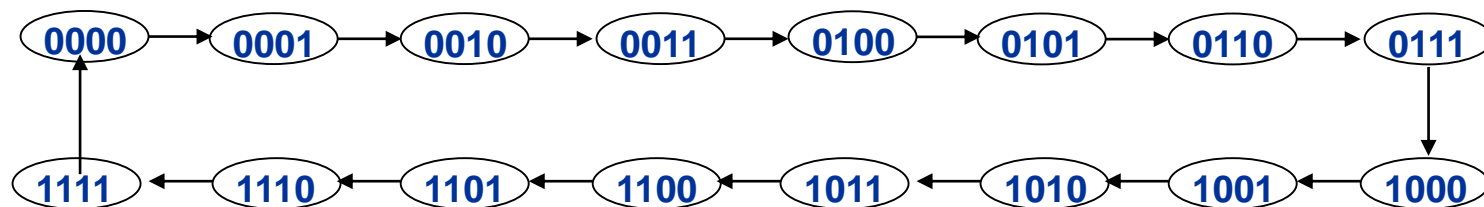
## (2) 状态转换表

和同步计数器相比，  
4个触发器状态发生变化的  
时间点有什么不同？

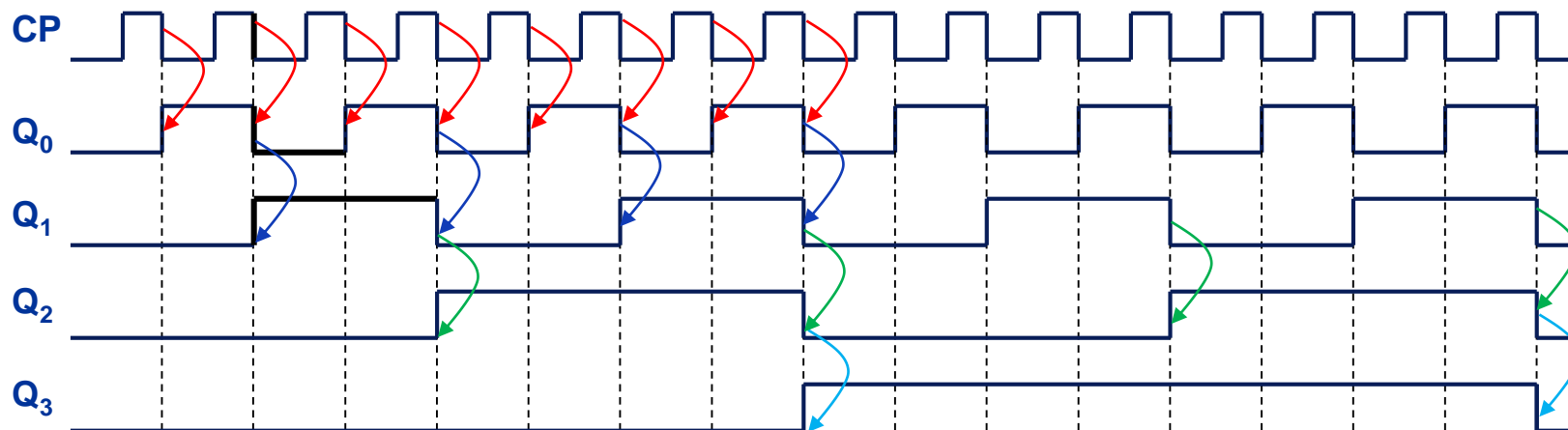
$Q_3^n$ $Q_2^n$ $Q_1^n$ $Q_0^n$	$Q_3^{n+1}$ $Q_2^{n+1}$ $Q_1^{n+1}$ $Q_0^{n+1}$
0 0 0 0	0 0 0 1
0 0 0 1	0 0 1 0
0 0 1 0	0 0 1 1
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	1 0 1 0
1 0 1 0	1 0 1 1
1 0 1 1	1 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	1 1 1 0
1 1 1 0	1 1 1 1
1 1 1 1	0 0 0 0

# 异步计数器

## (3) 状态转换图



## 时序图





# **第四部分**

## **层次化存储系统**

# 内容大纲

## ❖ 主存储器

- 存储单元电路：SRAM、DRAM、ROM
- 主存储器的结构
  - SRAM芯片的内部结构
  - DRAM芯片的内部结构（地址管脚复用）
- 存储器的扩展
- DRAM的刷新

## ❖ 高速缓冲存储器

- CACHE的基本原理与结构
- CACHE的映射方式
- CACHE的性能计算

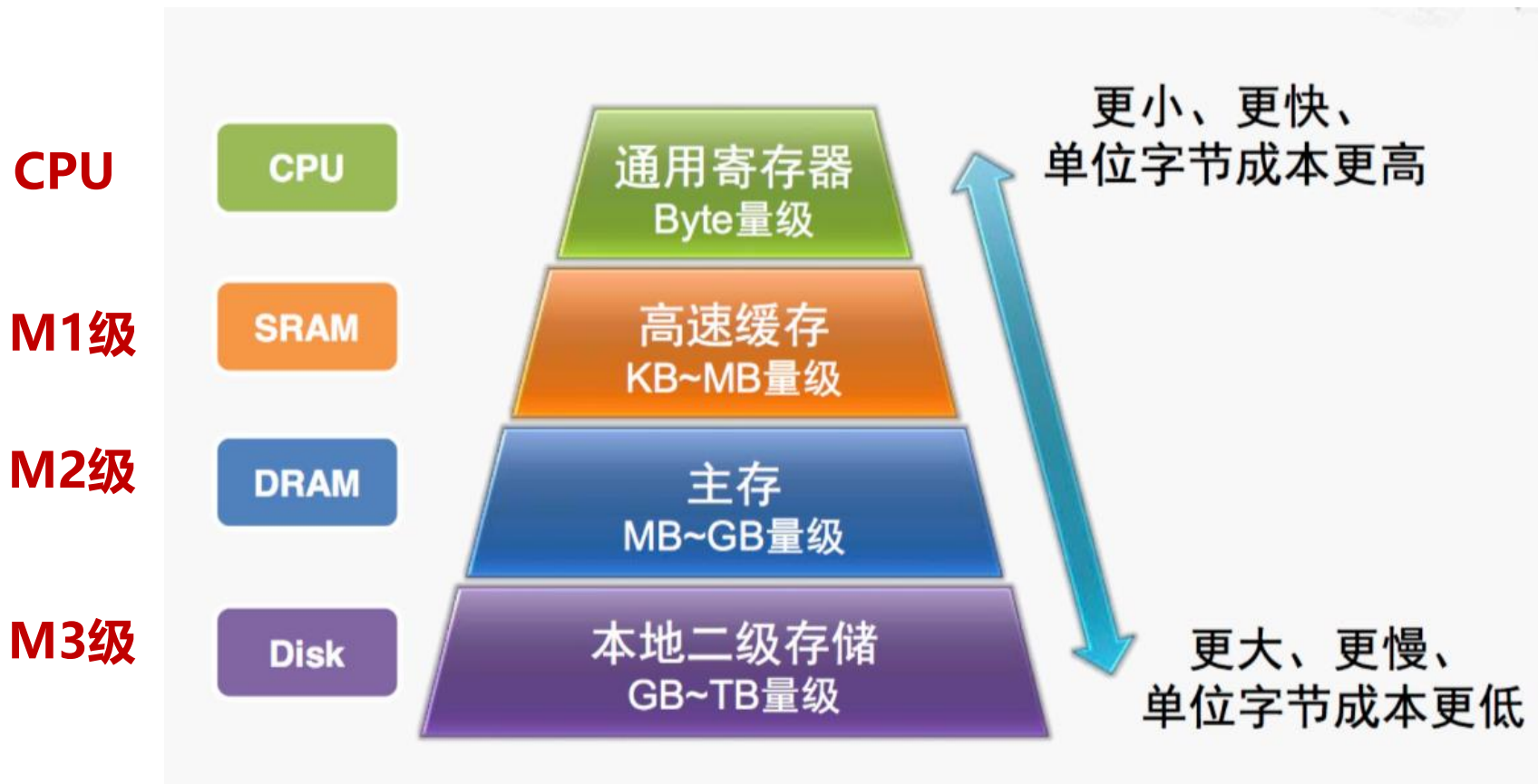
## ❖ 虚拟存储系统

- 页式虚拟存储器
- TLB



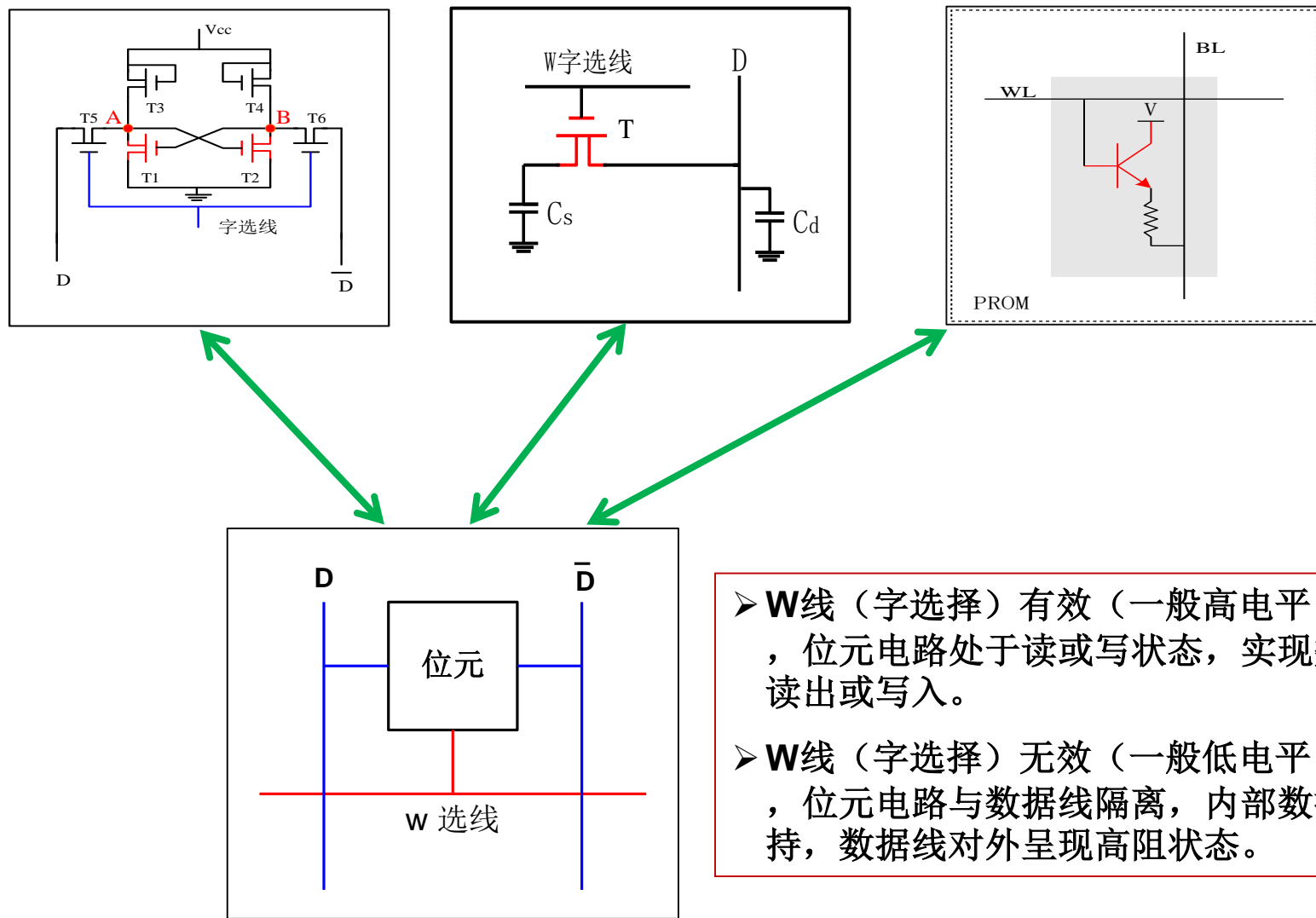
# 存储系统层次结构

- ❖ 目标：针对典型应用**平均访问时间最短**
- ❖ 途径：优化存储系统的组织（**多级分层结构**）



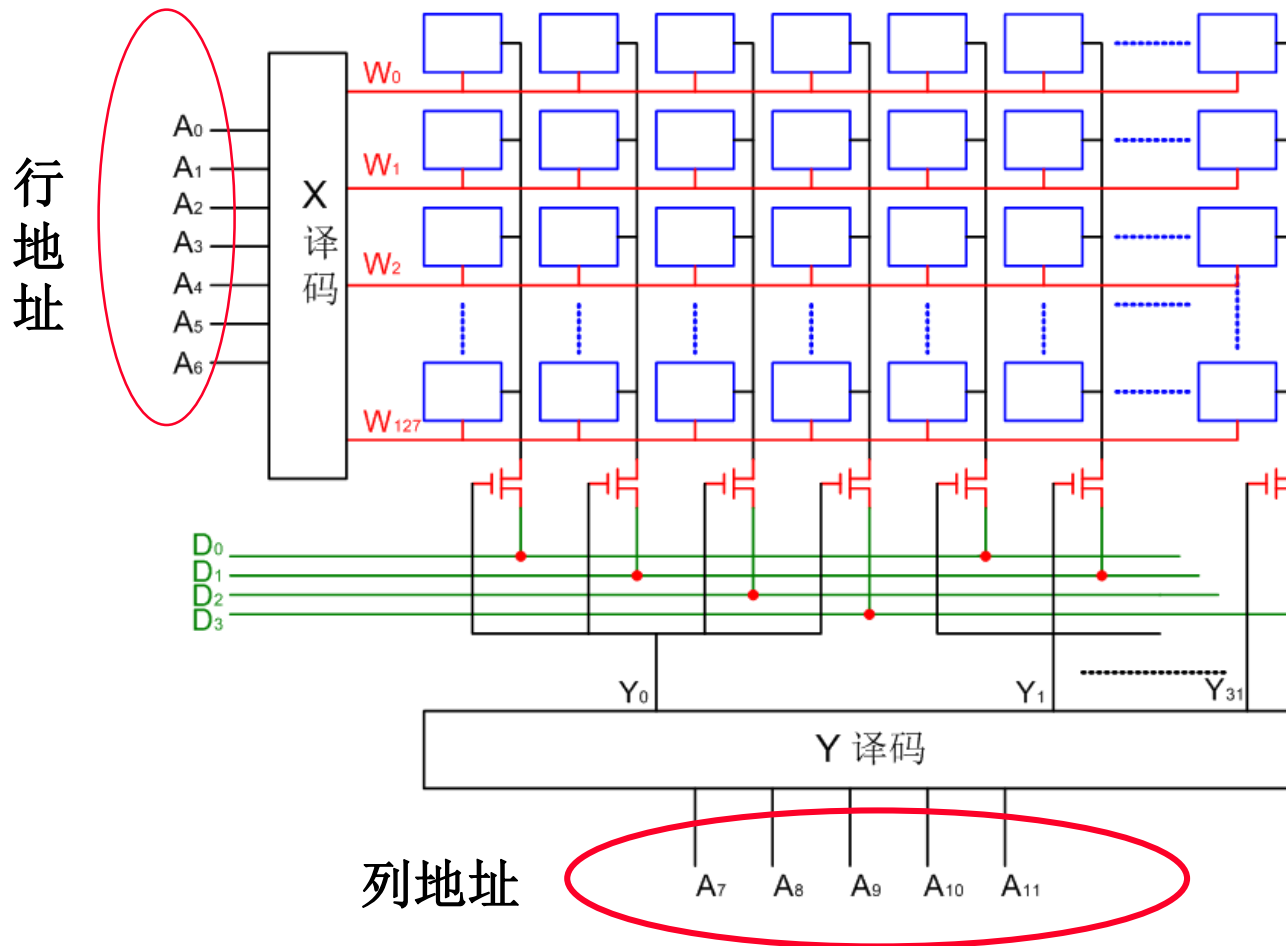
**效果：存储系统速度接近M1级，容量价格接近M3级**

# 存储单元电路



# SRAM存储芯片结构

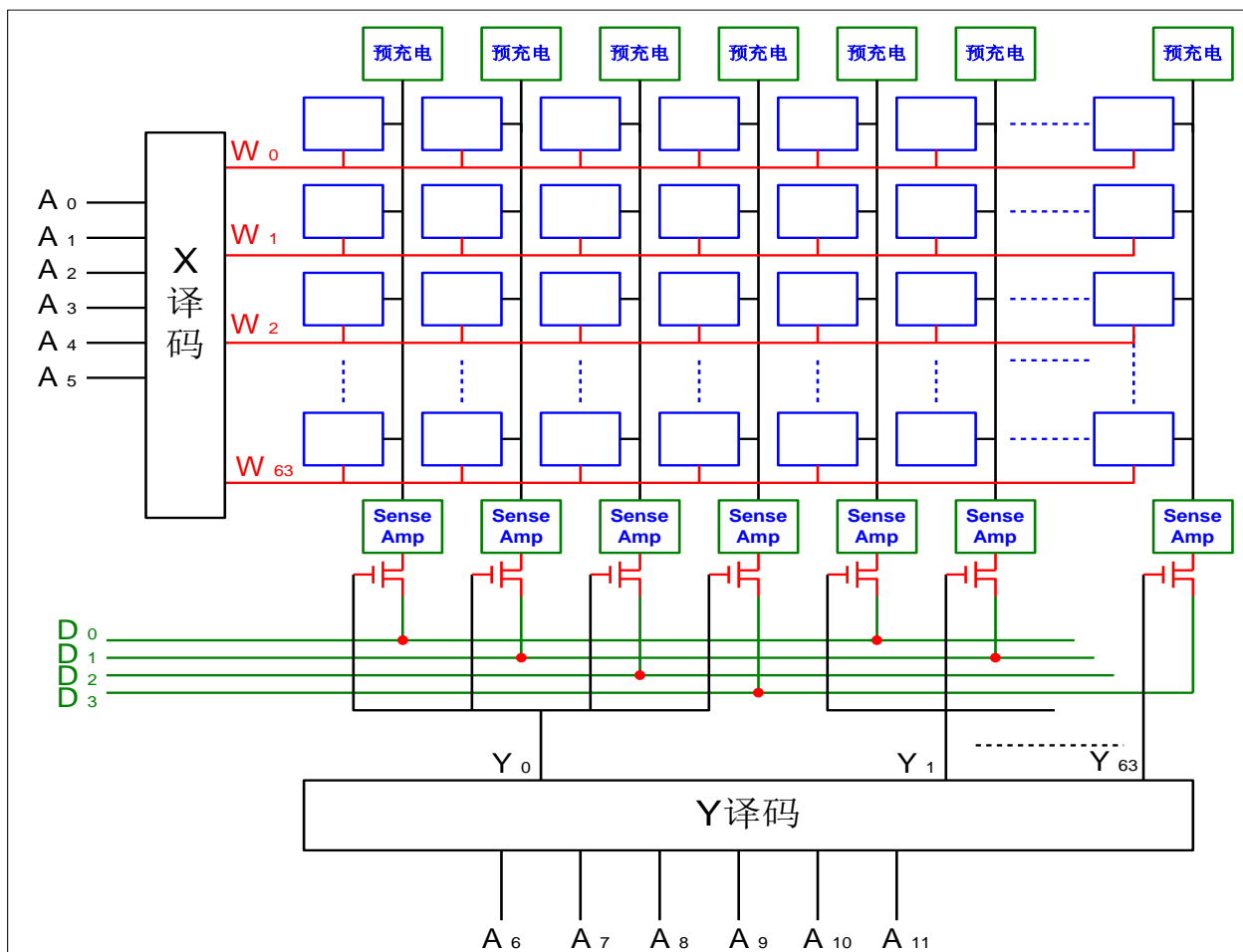
- ❖ 二维地址结构（SRAM）： $4096 \times 4$ ：4096 个字，每个字 4 位
  - 存储矩阵： $2^7 \times 2^7$  (128行 $\times$ 128列)
  - 一行包括32个字，要进行32选1的译码（Y译码），列地址5位



# DRAM存储芯片结构

## ❖ 4096\* 4 DRAM

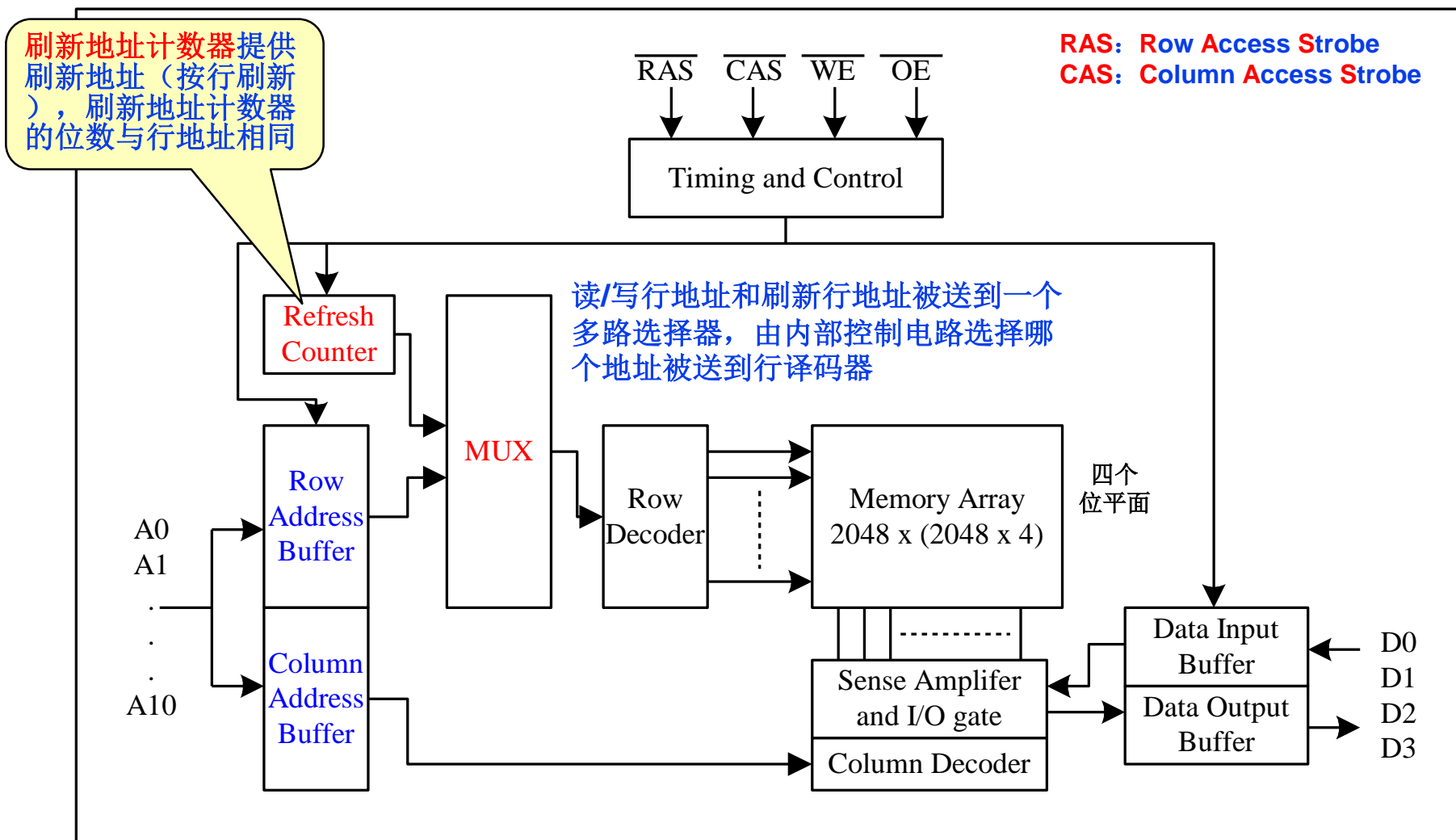
- 4096个字 =  $2^{12}$ , 行列地址管脚复用, 行列地址各6位。
- 存储矩阵:  $2^6 \times (2^6 \times 4)$  (64行  $\times$  256列)



按行刷新，每次刷新1行（256个单元）

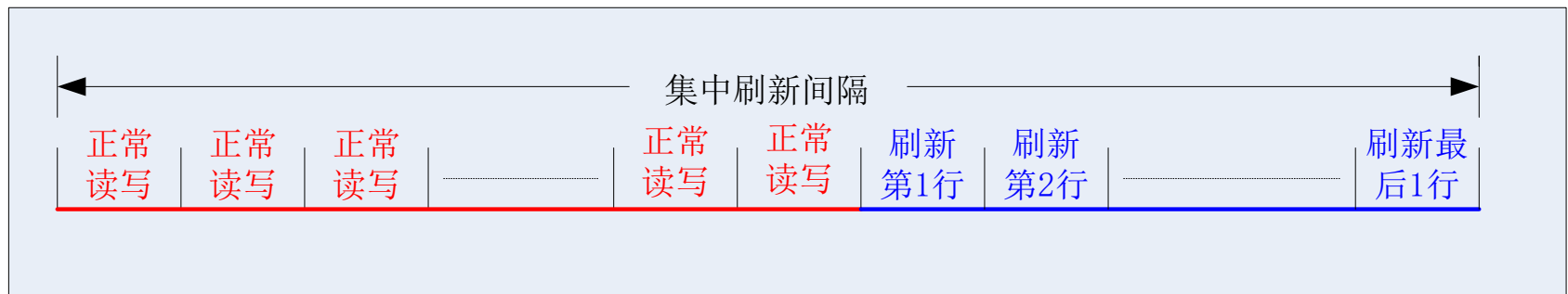
# 存储芯片结构示例

## ❖ DRAM 4M×4 DRAM芯片结构(内部包含刷新电路)



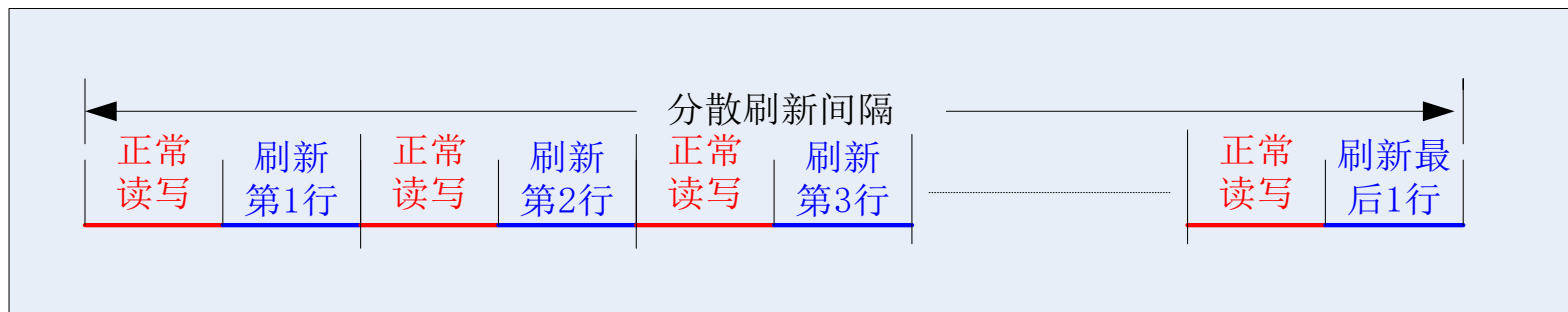
## ❖ 集中刷新方式

- 将最大刷新周期分成两部分，在一个时间段内刷新存储器所有行，另一个时间段CPU访问内存，刷新电路不工作。
- 集中刷新时段内存储器不能正常读写（访存死区），**该方法很少使用。**
- 刷新周期（集中刷新间隔）= 最大刷新周期



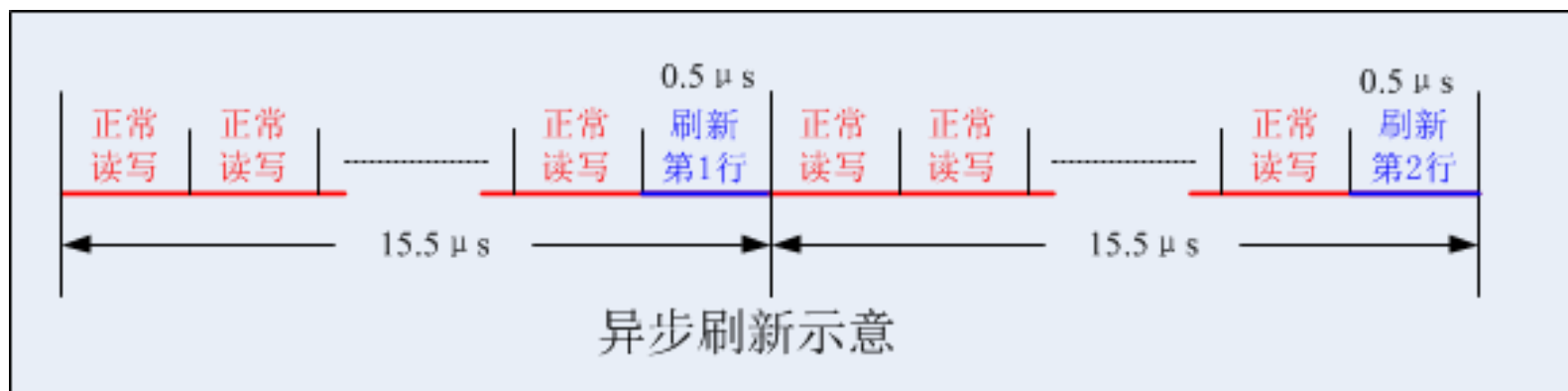
## ❖ 分散刷新方式

- 每个存储周期分为两段: 前一段用于正常读写, 后一段用于刷新操作。一个存储周期刷新1行, 下一存储周期刷新另一行, 直至最后1行后刷新后, 又开始刷新第1行。
- 存储周期加长, 效率降低, **很少使用**。
- 刷新周期 (分散刷新间隔) = 刷新行数 × 存储周期 ≤ 最大刷新周期



## ❖ 异步刷新方式

- 结合前两种方式，保证在一个**最大刷新周期内**将存储芯片内的所有行刷新一遍，且只刷新一遍。
- 刷新周期（异步刷新间隔）= 最大刷新周期
- 假定**最大刷新周期为2ms**，以128行为例，在2ms时间内必须轮流对每一行刷新一次，即每隔 $2\text{ms}/128=15.5\mu\text{s}$ 刷新一行。这时假定读/写与刷新操作时间都为 $0.5\mu\text{s}$ ，则可用前 $15\mu\text{s}$ 进行正常读/写操作，最后 $0.5\mu\text{s}$ 完成刷新操作。





## ➤ 混合扩展的基本思路

1. 确定每个芯片的地址位数、数据位数。
2. 确定整个存储空间所需的地址总线和数据总线的数量。
3. 计算所需芯片的数量，确定每个芯片在整个存储空间中的地址空间范围、位空间范围
4. 所有芯片的地址全部连接到地址总线对应的地址线上。
5. 同一字空间的存储芯片CS信号连在一起。
6. 同一位空间的数据线连在一起，并连接到对应的数据总线上。
7. 根据每个芯片的地址空间范围，设计芯片所需要的片选信号逻辑，CS逻辑电路的输入一定是地址总线中没有连接到芯片的地址管脚上的那部分地址线。
8. 统一读写控制。

# 存储器芯片的扩展 (混合扩展)

例：4Kx4 SRAM存储芯片构成16Kx8的存储器

➤ 4K×4芯片：

- 12个地址管脚 **A11~A0**
- 4个数据管脚 **D3~D0**
- 1个片选输入管脚 **CS#**
- 1个读写控制管脚 **WE#**
- 芯片地址空间：**000H~FFF H**

➤ CPU向存储器提供：

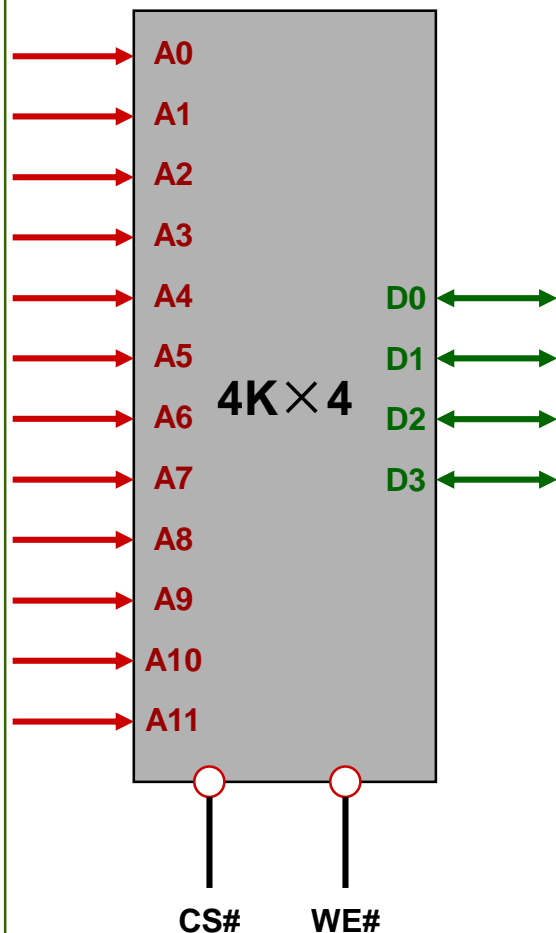
- 地址总线14根：**AB13~AB0**
- 数据总线8根：**DB7~DB0**
- 读写控制信号：**MemW**
- 存储器地址空间：**0000H~3FFF H**

➤ 需要芯片数： $(16K \times 8) / (4K \times 4) = 8$ 片

- 分4组（字扩展），每组2个芯片（位扩展）

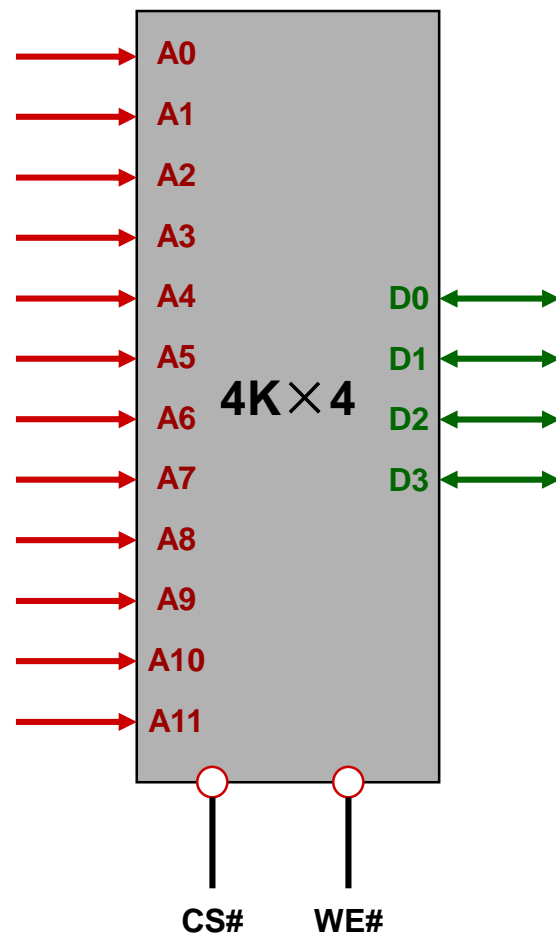
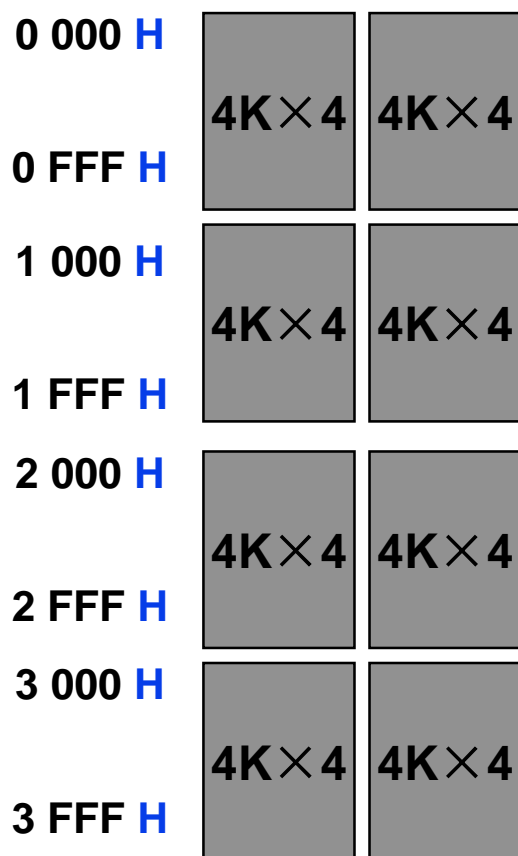
➤ 一个2-4译码器产生4个片选信号

- 译码器输入：**AB13~AB12**
- 译码器输出：分别接4组芯片片选管脚



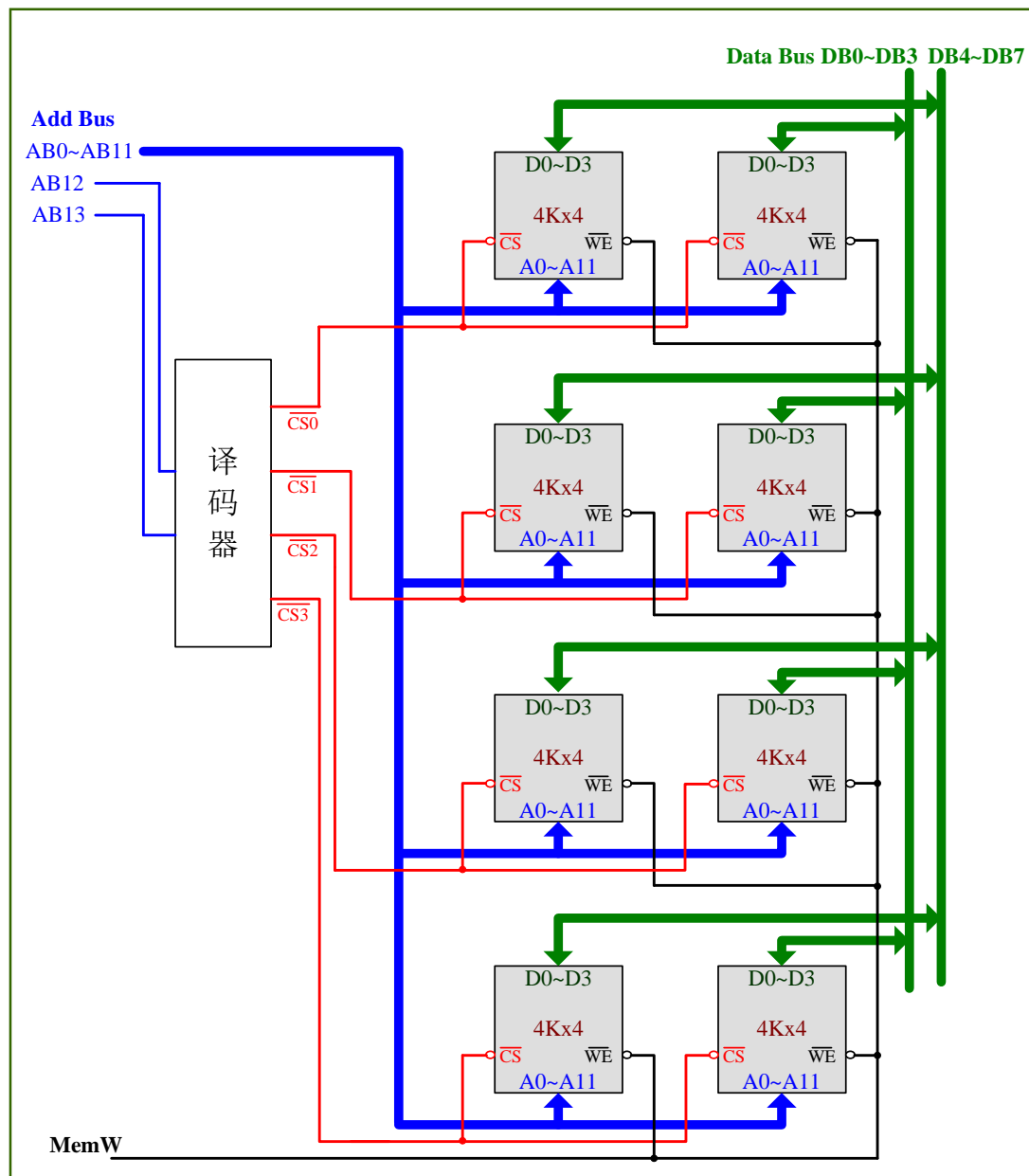
# 存储器芯片的扩展 (混合扩展)

## 4Kx4 SRAM存储芯片构成16Kx8的存储器地址空间划分



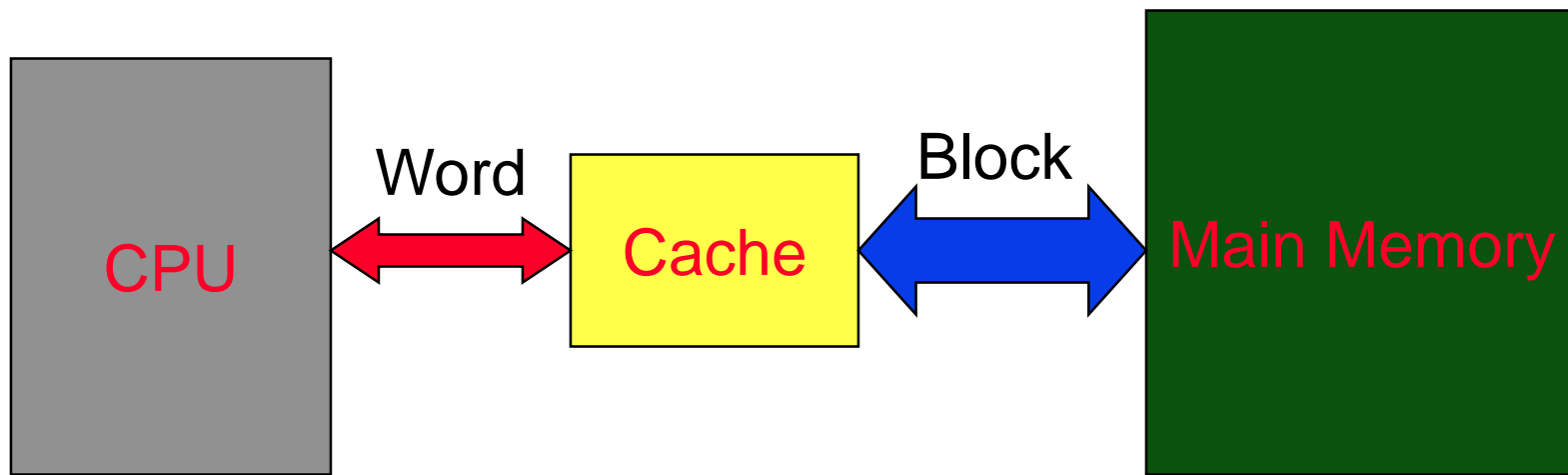
# 存储器芯片的扩展 (混合扩展)

4Kx4 SRAM存储芯片构成16Kx8的存储器连接图



# 高速缓冲存储器(Cache)的动机与原理

- ❖ **动机：解决CPU和主存储器之间的性能差距问题**
- ❖ **程序访问内存的局部性特征：时间局部性，空间局部性**
- ❖ **Cache：CPU和主存间的一容量较小的高速缓存，其中总是存放最活跃（被频繁访问）的程序块和数据，大多数情况下，CPU能直接从这个高速缓存中取得指令和数据，而不必访问主存。**
- ❖ **Cache与主存之间按照数据块（Block）为单位进行数据交换。**



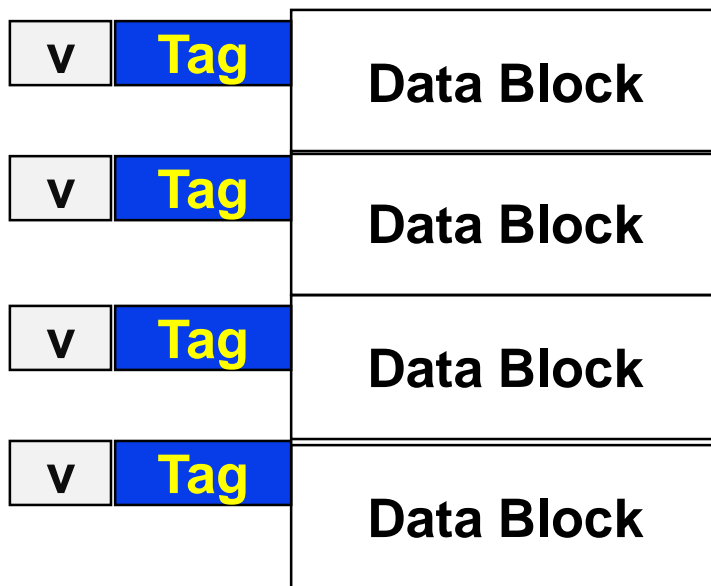
## ❖ Cache要解决的问题

- ① **快速访问**: 具备快速访问的能力 (采用SRAM) ;
- ② **数据交换**: 与主存交换数据的能力, 将主存最活跃单元所在数据 (或指令) 块复制到Cache中;
- ③ **地址判断**: 由于CPU总是以主存地址访问存储器, 所以Cache应具备有判断CPU当前要访问的内容是否在Cache中的能力, 并具有根据主存地址在Cache中访问相应单元的能力;
- ④ **替换决策**: 具备在Cache容量不够时替换Cache中某些内容的决策能力。

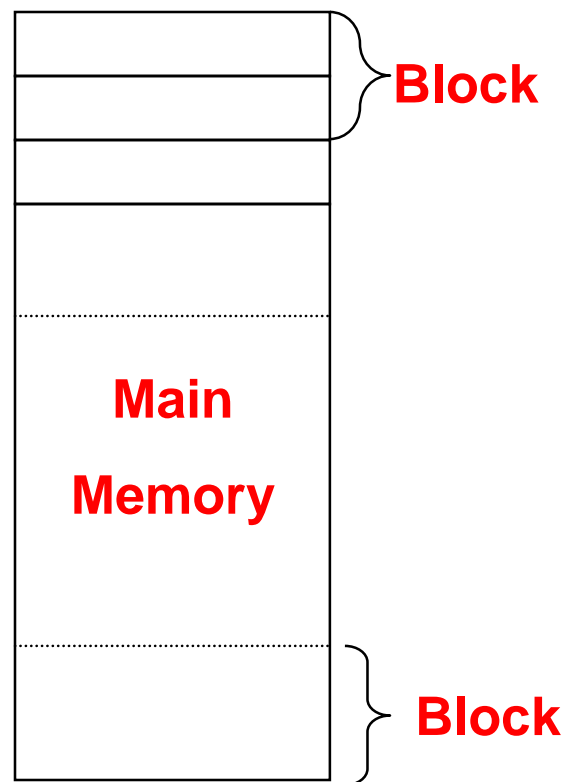
# 高速缓冲存储器(Cache)的原理

## ❖ Cache的基本结构

- **存储机构**：保存数据，存取数据，一般采用SRAM构成。以Block（若干字）为单位；
- **地址机构**：地址比较机制，地址映射机制，地址标记（Tag），一个Block具有一个Tag；
- **替换机构**：记录Block的使用情况，有效位（v）记录对应数据块中的数据是否有效；替换策略。



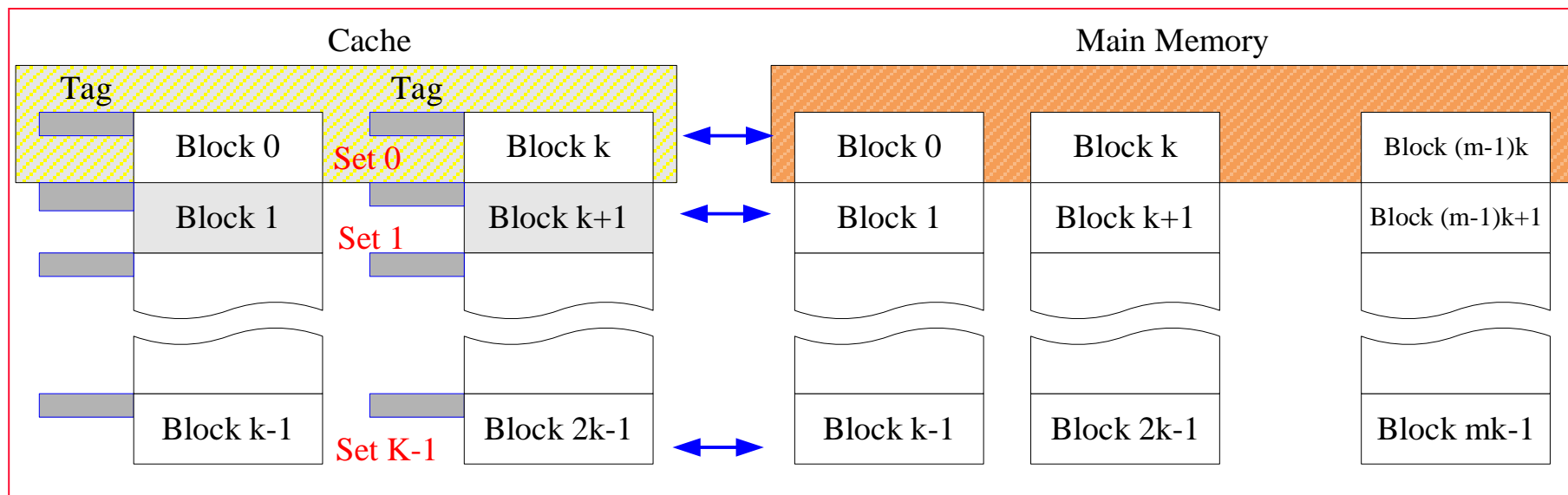
Cache 的基本结构



# Cache与主存之间的映射：组相联

## ❖ 组相联 (Set Associative)

- Cache包含**M**块，分 **K** 组，每组包含 **L** 块， $M=K*L$ ;
- 主存块 **J** 按  $I = J \bmod K$  的规则映射到 Cache **组 I** 中的任意块;
- 主存可以视为逻辑上也分成 **K** 组，主存组M内的一个主存块只能映射到Cache组M内，但可以是组M内任意Cache块。





# Cache与主存之间的映射：组相联

## ❖ 组相联映射

➤ 主存的地址格式：

Tag	Index	Offset
-----	-------	--------

➤ Tag的内容：主存中与该Cache数据块对应的数据块的组内块地址。

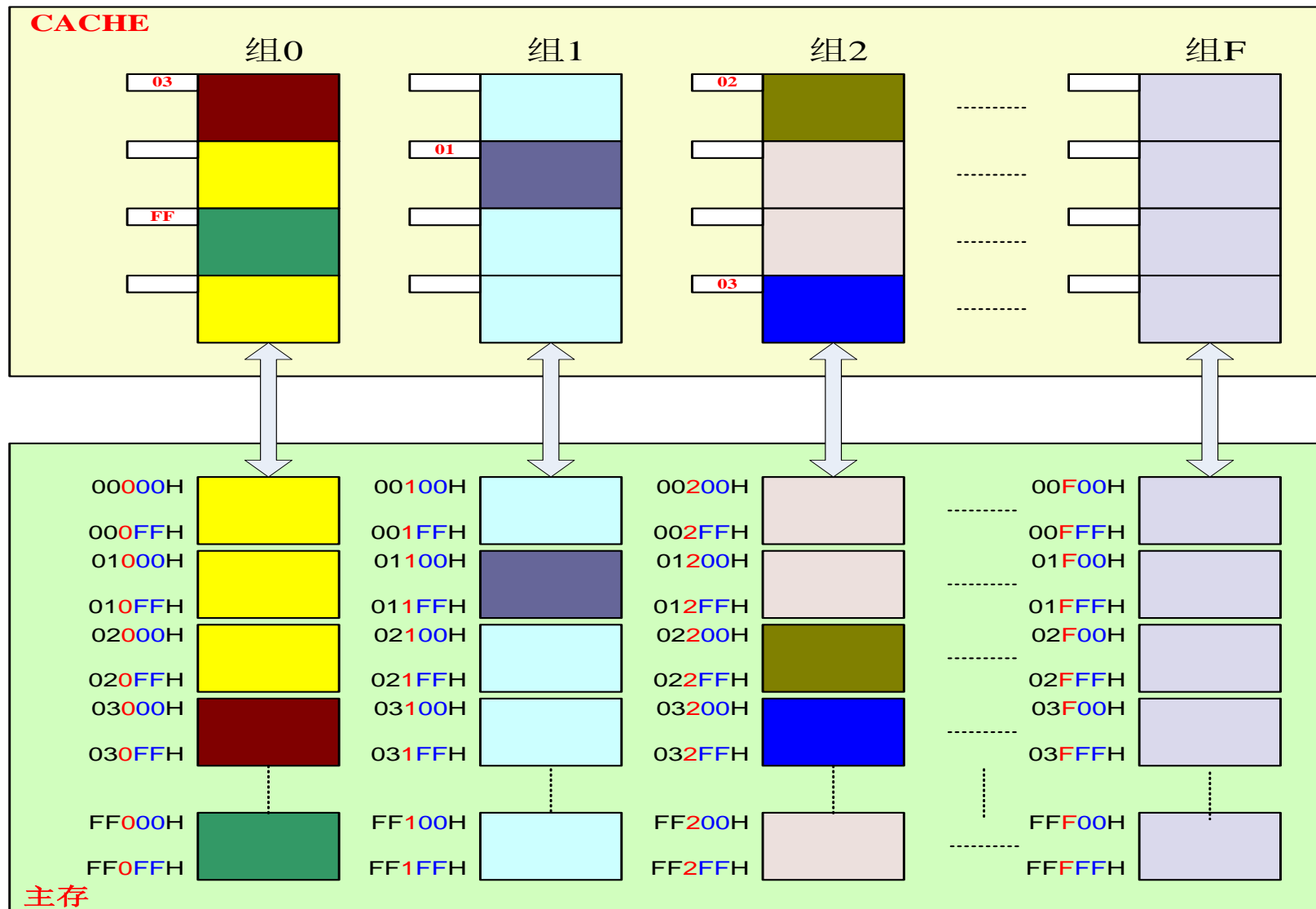
## ❖ 举例

- 主存容量1M 字节，4路组相联（每组包含4个Block） Cache容量16K字节，Block大小256 字节
- Cache分多少组？ 每组包含多少块？
- Cache的Tag需要多少位？

解：

- Cache 组数 =  $2^{14} \div (2^8 \times 2^2) = 2^4 = 16$  组
- 主存每组块数 =  $2^{20} \div (2^8 \times 2^4) = 2^8 = 256$  块/组
- 主存地址：20 位，高8 位为组内块地址（tag），中间4 位为组地址（index），低 8 位为块内地址（offset）
- Cache的Tag应该为 8 位。

# Cache与主存之间的映射 — 组相联



### 3 种映射机制的关系

主存地址格式

Tag (a1位)	Index (a2位)	Offset(a3位)
-----------	-------------	-------------

Cache包含**M**块，主存块**J**的映

❖ 组相联

➤ Cache分成**K**组，每组包含**L**块， $M=K \times L$

➤ 映射关系： $I = J \bmod K$ ，映射到 Cache 组**I**中的任意块（**1:L**映射）

❖ 全相联（组相联特例：1个组的组相联）

➤ 等价于**K=1**，**L=M**的组相联，1个组的组相联

➤ 映射关系： $I = J \bmod K$ ，映射到 Cache 组**0**中的任意块（**1:L**映射）

❖ 直接映射（组相联特例：1路组相联）

➤ 等价于**K=M**，**L=1**的组相联，1路组相联（每组只有1个Cache块）

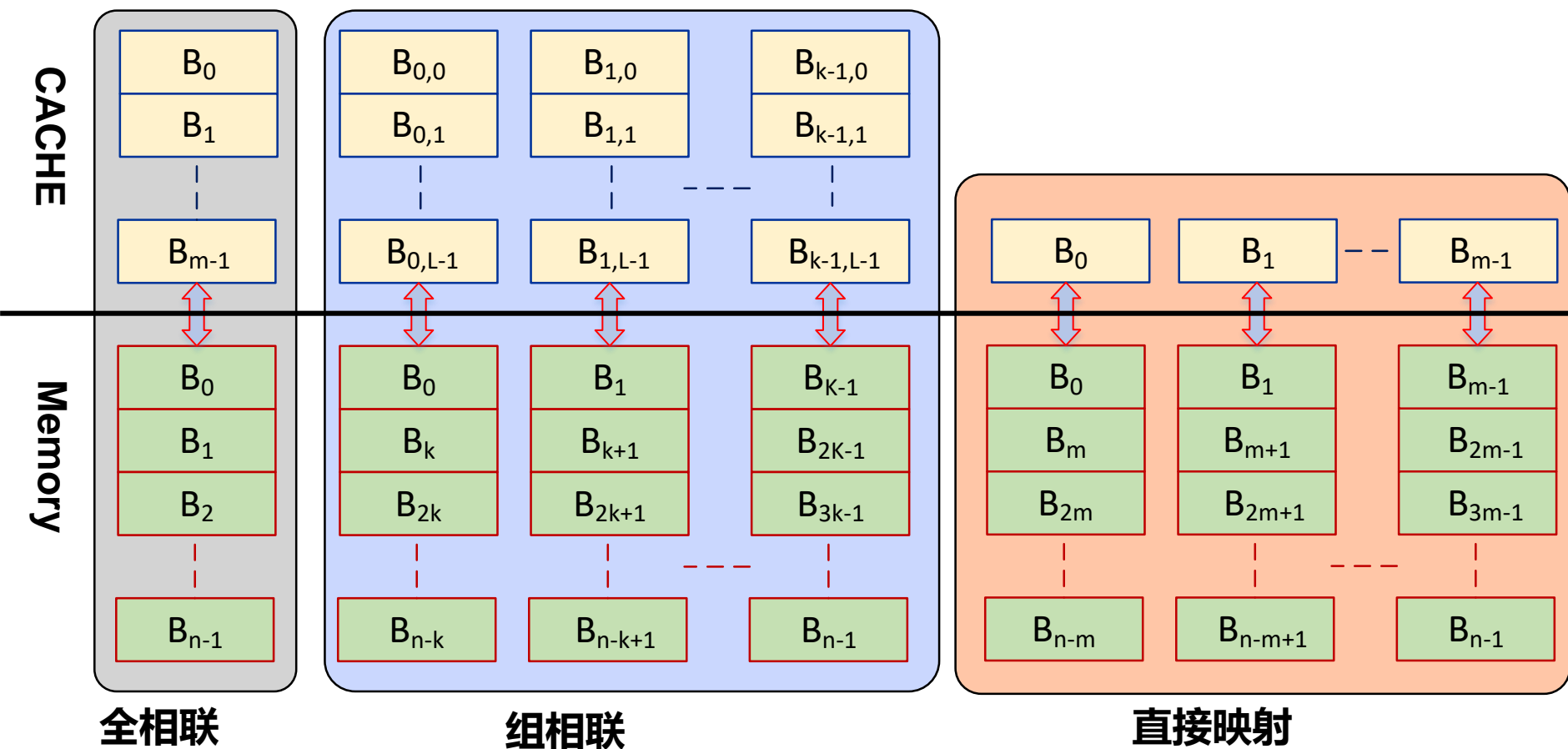
➤ 映射关系： $I = J \bmod K$ ，映射到 Cache 组**I**中的任意块（**1:L**映射）

$$a3 = \log_2 E$$

$$a2 = \log_2 K$$

$$a1 = \log_2 \frac{N}{M}$$

### 3 种映射机制的关系



# Cache的性能计算

## ■存储访问时间

若： $T_m$ 为主存储器的访问周期；

$T_c$ 为Cache的访问周期；

$H$ 为Cache命中率

则存储系统的等效访问周期 $T$ 为：

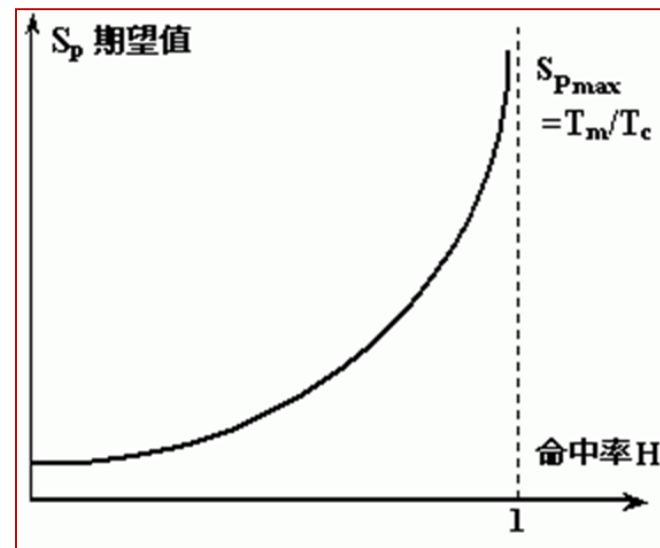
$$\begin{aligned} T &= T_c \times H + (T_c + T_m) \times (1 - H) \\ &= T_c + T_m \times (1 - H) \end{aligned}$$

## ■加速比SP (Speedup)

存储系统的加速比 $S_p$ 为：

$$S_p = \frac{T_m}{T} = \frac{T_m}{T_c + T_m \times (1 - H)}$$

假设：CPU访问Cache失效时，直接访问主存直接获得数据，读取主存块并保存到Cache中的时间忽略不计。



加速比与命中率的关系

# 虚拟存储系统概述

## ■ 虚拟存储器 (Virtual Memory)

### ➤ 模式一（很少用）：虚拟存储=主存+辅存

- Overlay技术：程序分段，段长<内存大小；程序员负责换入换出

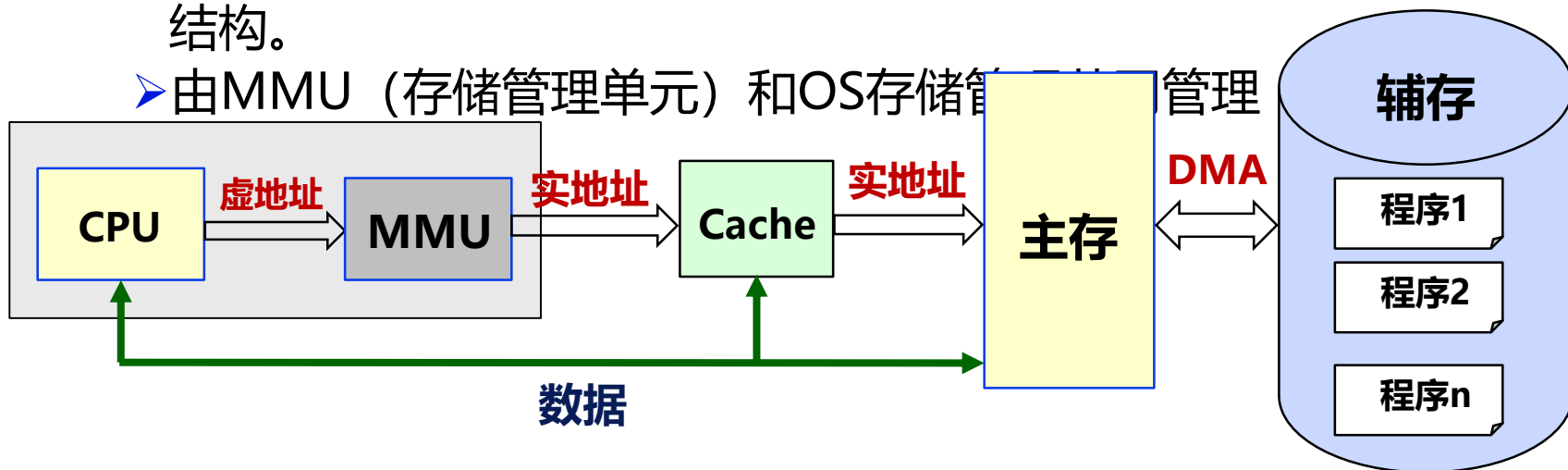
### ➤ 模式二：主存作为辅存（虚存）的高速缓存（Cache）

- 现代虚拟存储系统普遍采用的模式
- 采用虚拟地址访存，虚存驻留在辅存，基于局部性原理调入
- 重定位 (Relocation) 技术：同一虚页可以加载到内存不同位置

### ➤ 虚拟存储器能从逻辑上为用户提供一个比物理内存容量大得多、可寻址的“主存储器”。

### ➤ 虚拟存储器的容量与物理主存大小无关，而受限于计算机的地址结构。

### ➤ 由MMU（存储管理单元）和OS存储管理单元管理



## ■ 虚存空间与物理空间

- 用户程序空间：用户程序给出的指令和数据地址不是实际主存地址，称为**虚地址或逻辑地址**，其对应的存储空间称为**虚存空间**或逻辑地址空间。
- 物理内存空间：指令和数据的实际主存地址称为**实地址或物理地址**，其对应的存储空间称为**物理空间**或主存空间。

## ■ 虚拟存储器要解决的问题

- 虚存空间与物理空间之间的数据交换：交换哪些数据？每次交换多少？
- 虚地址与实地址的转换问题：虚地址格式、实地址格式、怎么判断当前访问的虚地址对应的数据是不是在物理空间中，如何把虚地址转换为实地址？如何加速这种判断和转换？
- 缺失处理和替换策略：访问的内容不在物理空间中怎么处理？

# 页式虚拟存储器

## ❖ 页式虚拟存储管理

### ➤ 基本思想

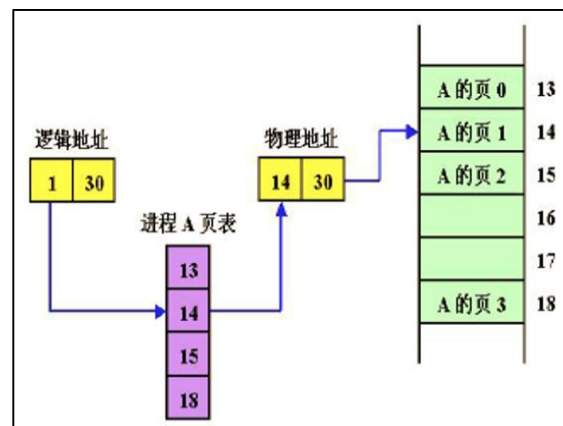
- 内存按照固定的大小分页（存储页，**实页**），每个进程也按相同大小分页（程序页，**虚页**）；
- 内存按页顺序编号（**实页号**，PPN: Physical Page Number），每个独立编址的程序空间有自己的页号顺序（**虚页号**，VPN: Virtual Page Number）；
- 操作系统将**辅助存储器中进程的虚页**装入到**内存中的实页**中；
- 进程运行无需占用连续的实页，也无需把所有的虚页都装入内存；
- 操作系统为每一个进程维护一个页表（Page Table），通过页表实现逻辑地址到物理地址的转换。**地址转换由CPU中的MMU实现。**

➤ 逻辑地址：程序中指令所使用的地址，  
也称虚拟地址或虚地址

➤ 物理地址：内存中存放指令或数据的实际地址，

➤ **页式调度：按页交换**

- **优点：**页内零头小，页表对程序员来说是透明的，地址变换快，调入操作简单；
- **缺点：**各页不是程序的独立模块，不便于实现程序和数据保护。



逻辑地址→物理地址



# 页式虚拟存储器

## ❖ 页式虚拟存储器

➤ **页表**：记录虚页与实页的映射关系，实现虚实地址转换，在内存中，操作系统为每道程序建立一个页表。页表用虚页号作为索引，页表项包括虚页对应的**实页号**和**有效位**。

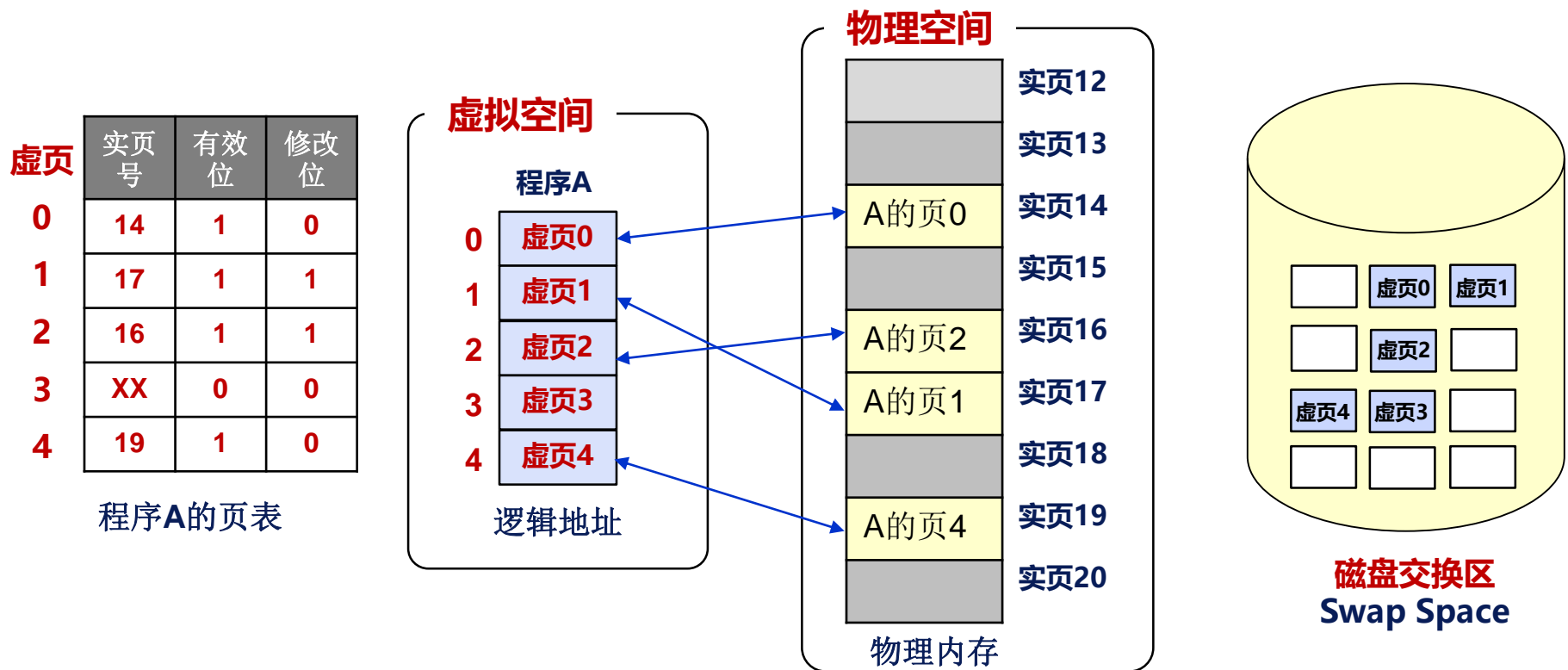
➤ **页表寄存器**：保存页表在内存中的首地址。

➤ **虚地址格式**：

<b>虚页号(VPN)</b>	<b>页内位移(VPO)</b>
-----------------	------------------

➤ **实地址格式**：

<b>实页号(PPN)</b>	<b>页内位移(PPO)</b>
-----------------	------------------



# 页式虚拟存储器

## ❖ 页表

- 每个进程有一个页表，页表项(Page Table Entry)数取决于虚拟地址的结构。
- 页表项包括：实页号、装入位、修改位等
- 页表由OS为进程创建和维护，在内存OS地址空间中，用户无法访问
- 页表首地址记录在页表基址寄存器(Page Table Base Register)。

页表首地址

	装入位	修改位	替换控制位	其他	实页号
0 虚页	1				11
1 虚页	1				13
2 虚页	1				16
3 虚页	1				10
4 虚页	1				14

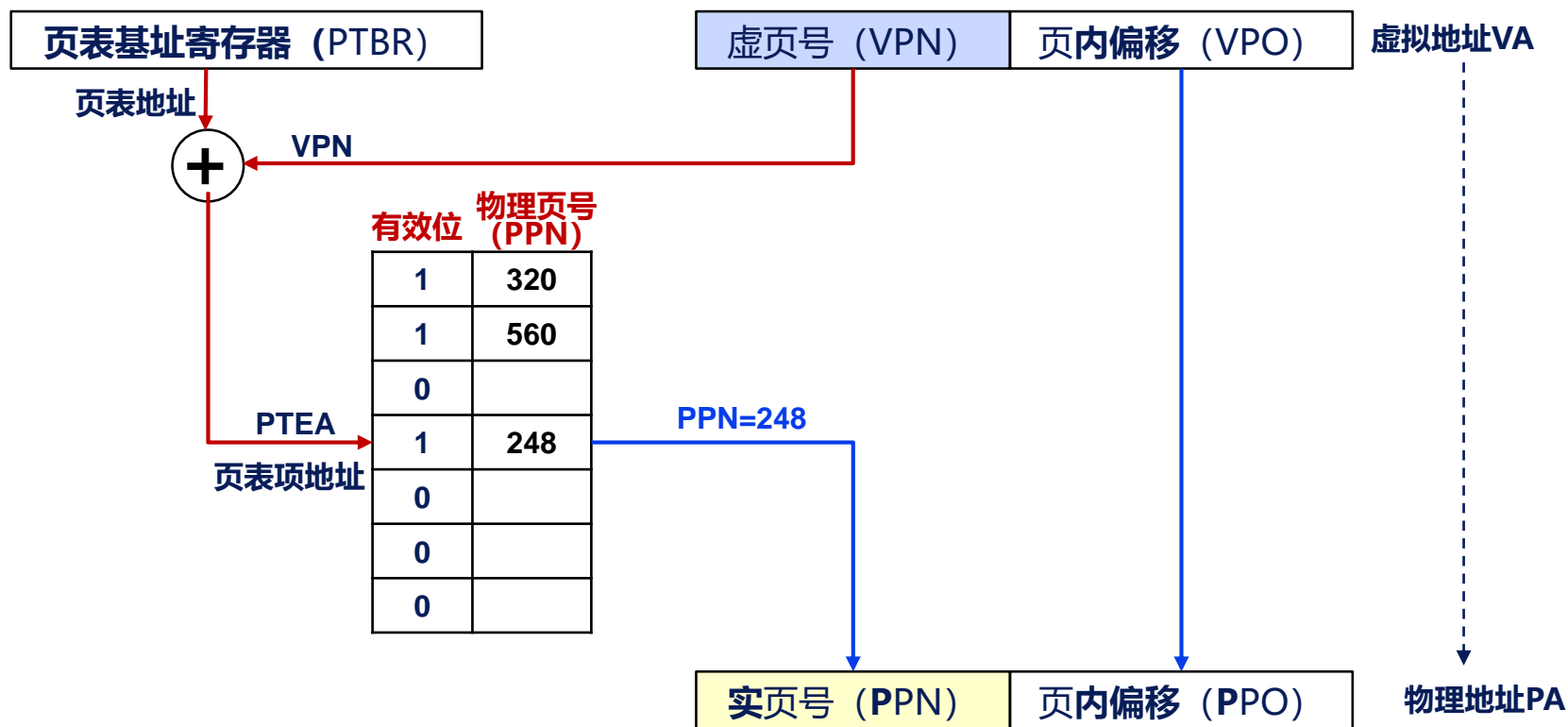
用户程序 A 的页表

## ❖ 页表空间问题

- 页表可能很大。如VAX系统中，用户程序（进程）虚拟存储空间最大可达2GB，按512字节分页，有 $2^{22}$ 页，页表可以包括 $2^{22}$ 个页表项。显然，这么大的页表需要占用很大的内存空间。
- 多进程运行，多个页表同时都在内存。多个同时运行的进程的页表空间超过内存可分配空间的可能性是存在的。
- 采用多级页表机制：将页表分页，当前使用的页的页表项所在页在内存，其余在外存，页表也采用按页交换机制。

# 页式虚拟存储器

## 虚拟地址到物理地址的转换



- VA: 虚拟地址 (Virtual Address)
- PA: 物理地址 (Physical Address)
- PTE: 页表项 (Page Table Entry)
- PTEA: 页表项地址 (Page Table Entry Address)
- PTBR: 页表基址寄存器 (Page Table Base Register)

# 页式虚拟存储器

## ❖ 举例

某计算机虚拟地址32位，物理内存128MB，页大小4KB。

- (1) 程序虚拟空间最多可有多少页？
- (2) 页表项共有多少位？每个页表最大占多少内存空间？
- (3) 若部分页表项内容如右表，求对应虚拟地址00002240H和00005380H的物理地址

虚页 有效位 物理页号

0	1	0004H
1	0	0032H
2	1	0008H
3	1	0010H
4	0	0020H
5	0	0024H
6	0	

## ❖ 解答

(1) 虚地址32位：虚页号 (20位) + 页内偏移 (12位)

实地址27为：实页号 (15位) + 页内偏移 (12位)

每个程序虚拟空间最多可有： $2^{20}$ 个虚页

(2) 每个页表项：1位 (有效位) + 15位 (实页号) = 16位

最大页表所占空间： $2^{20} \times 16 = 16\text{Mb} = 2\text{MB}$

(3) 查页表，00002虚页对应的实页号是0008，有效位为1，所以虚拟地址00002240H对应的物理地址是0008240H；00005虚页不在内存中 (有效位为0)。


多道程序运行时，所有程序的页表都在内存中，页表占用内存空间不可忽视，极端情况下，页表有可能消耗所有内存空间。  
(采用多级页表解决)

# 页式虚拟存储器

## ❖ 加快地址转换，采用快表TLB（Translation Lookaside Buffer，转换后备缓冲器）

- 问题：每次虚拟存储器的访问带来两次存储器访问，一次访问页表，一次访问所需数据（或指令），简单的虚拟存储器速度慢。
- 解决办法：使用Cache存储部分活跃的页表项，称为TLB（快表），它包含了最近使用的那些页表项。
- TLB内容（全相联模式）：标记（**虚页号**）、数据块（**实页号**）、有效位、修改位。
- **TLB一般采用全相联**

快表（TLB）	有效位	修改位	标记（tag）	数据
			<b>虚页号</b>	实页号
			<b>虚页号</b>	实页号
			<b>虚页号</b>	实页号
			<b>虚页号</b>	实页号



# **第五部分**

## **MIPS指令系统与汇编语言**

## ❖ 指令系统概述

### ➤ 指令格式

- 固定长度指令、变长指令
- 固定长度操作码，变长操作码

### ➤ 寻址方式

## ❖ MIPS指令系统

### ➤ MIPS的指令格式

### ➤ MIPS的寻址方式

### ➤ MIPS典型指令功能

## ❖ MIPS汇编语言编程

## ❖ 指令的表示

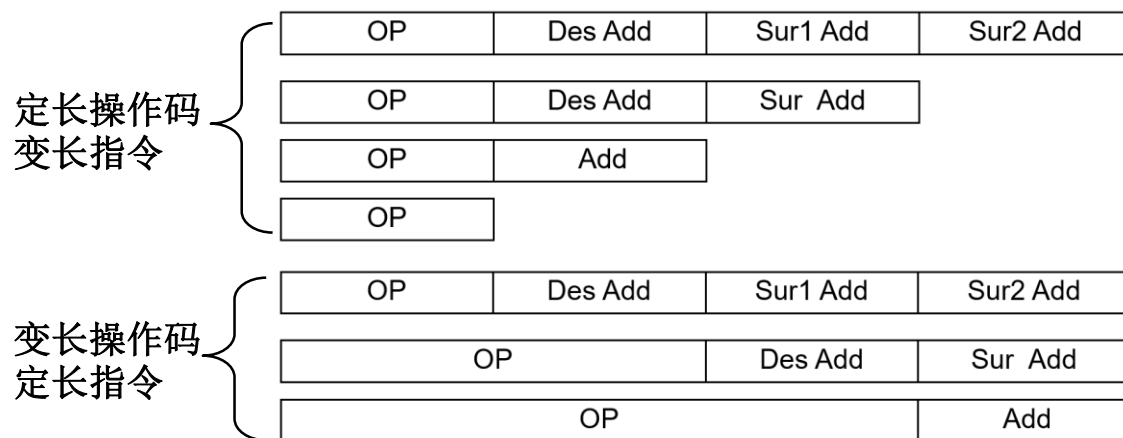
- 机器表示：二进制代码
- 符号化表示：助记符，如 MOV AX, BX

## ❖ 操作数地址的数目

- 三地址：Des  $\leftarrow$  (Sur1) OP (Sur2)
- 双地址：Des  $\leftarrow$  (Sur) OP (Des)
- 单地址：累加器作为默认操作数的双操作数型，或单操作数型
- 无地址：隐含操作数型，或无操作数型

## ❖ 指令编码长度

- 定长指令、变长指令





## ❖ 操作码结构

- 固定长度操作码：操作码长度（占二进制位数）固定不变。
  - 硬件设计简单
  - 指令译码时间开销较小
  - 指令空间利用率较低
- 可变长度操作码：操作码长度随指令地址数目的不同而不同。
  - 硬件设计相对复杂
  - 指令译码时间开销较大
  - 指令空间利用率较高

## ❖ 指令长度

- 定长指令系统，如MIPS指令
- 变长指令系统：一般为字节的整数倍，如80X86指令

## ❖ 形式地址与有效地址

- 形式地址：指令中直接给出的地址编码。
- 有效地址：根据形式地址和寻址方式计算出来的操作数在内存单元中的地址。
- 寻址方式：根据形式地址计算到操作数的有效地址的方式（算法）

## ❖ 常用寻址方式

- 立即寻址
- 寄存器直接寻址
- 基址寻址
- 相对寻址：基址寻址的特例，程序计数器PC作为基址寄存器
- 堆栈寻址

## ❖ 操作数的位置

- 存储器（存储器地址）
- 寄存器（寄存器地址）
- 输入输出端口（输入输出端口地址）

# MIPS指令系统：指令格式

## ❖ MIPS 指令格式

- Op: 6 bits, Opcode
- Rs: 5 bits, The first register source operand
- Rt: 5 bits, The second register source operand
- Rd: 5 bits, The register destination operand
- Shamt: 5 bits, Shift amount ( shift instruction)
- Func: 6 bits, function code ( another Opcode)
  - R-Type指令OP字段为 “000000” , 具体操作由func字段给定

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R-Type	Op	Rs	Rt	Rd	Shamt	Func
I-Type	Op	Rs	Rt	16 bit Address or Immediate		
J-Type	Op	26 bit Address ( for Jump Instruction)				

# MIPS指令系统：寻址方式

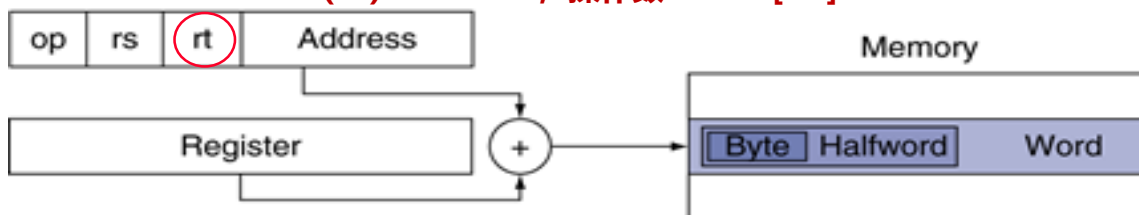
## 1. 立即寻址 操作数=Immediate



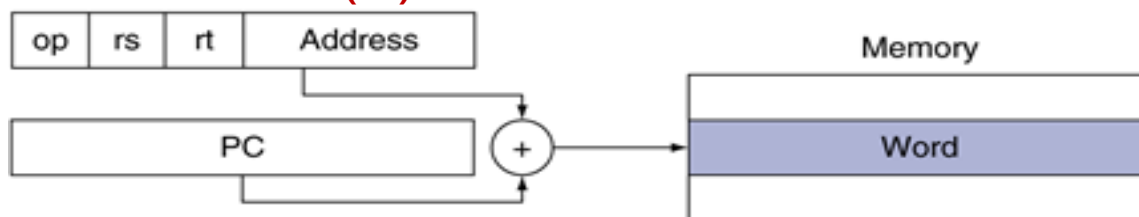
## 2. 寄存器寻址 $EA=Rs$ , 操作数= (Rs)



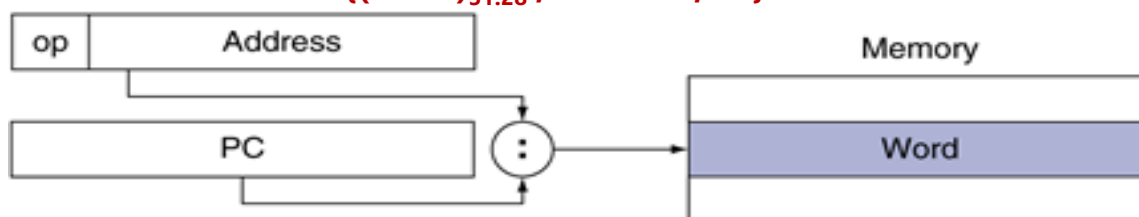
## 3. 基址寻址 $EA=(Rt)+Address$ , 操作数=Mem[EA]



## 4. PC相对寻址 $EA=(PC)+Address$



## 5. 伪直接寻址 $EA=\{(PC+4)_{31:28}, Address, 00\}$



操作码直接隐含寻址方式，指令代码中不需要寻址方式编码。

# MIPS指令系统：指令类型

## ❖ 算术、逻辑、移位指令

- **R类型**指令：两个源操作数都是寄存器，目的操作数是寄存器
- **I类型**指令：源操作数一个是寄存器、一个是**16**位立即数，目的操作数是寄存器

## ❖ 取数/存储指令

- **I类型**指令：存储器与通用寄存器之间传送数据，基址寻址

## ❖ 条件分支指令：控制程序执行顺序

- **跳转指令**：**J类型指令**（**26**位绝对转向地址）或**R类型指令**（**32**位的寄存器地址）
- **转移指令**：**I类型指令**，**PC**相对寻址。


## ❖ 函数调用指令（特殊指令）

- **R类型**指令
- 系统调用**SYSCALL**
- 断点**BREAK**

# MIPS指令系统：指令示例

Instruction	Example	Meaning	Comments
add	add \$1,\$2,\$3	$\$1 \leftarrow \$2 + \$3$	3 operation
subtract	sub \$1,\$2,\$3	$\$1 \leftarrow \$2 - \$3$	3 operation
add immediate	addi \$1,\$2,100	$\$1 \leftarrow \$2 + 100$	+ constant
multiply	mult \$2,\$3	$Hi,Lo \leftarrow \$2 \times \$3$	64-bit signed product
divide	div \$2,\$3	$Lo \leftarrow \$2 \div \$3$ $Hi \leftarrow \$2 \bmod \$3$	Lo = quotient Hi = remainder
move from Hi	mfhi \$1	$\$1 \leftarrow Hi$	Get a copy of Hi
move from Lo	mflo \$1	$\$1 \leftarrow Lo$	Get a copy of Lo
and	and \$1,\$2,\$3	$\$1 \leftarrow \$2 \& \$3$	Logical AND
or	or \$1,\$2,\$3	$\$1 \leftarrow \$2   \$3$	Logical OR
store	sw \$3,\$4(\$5)	$Mem(\$5+\$4) \leftarrow \$3$	Store Word
load	lw \$1,\$2(\$3)	$\$1 \leftarrow Mem(\$3+\$2)$	Load word
jump and link	jal 1000	$\$31 = PC + 4$ Go to 1000	Procedure call
jump register	jr \$31	Go to \$31	procedure return
set on less than	slt \$1,\$2,\$3	if ( $\$2 < \$3$ ) then $\$1 = 1$ else $\$1 = 0$	

- ❖ 理解指令功能
- ❖ 掌握汇编语言程序编程的基本技巧
- ❖ 分析汇编语言程序的功能
- ❖ 修改、补全汇编语言程序



# **第六部分**

## **MIPS处理器通路设计**



## ❖ MIPS处理器设计概述

- 结构、指令集、数据通路的基本组件

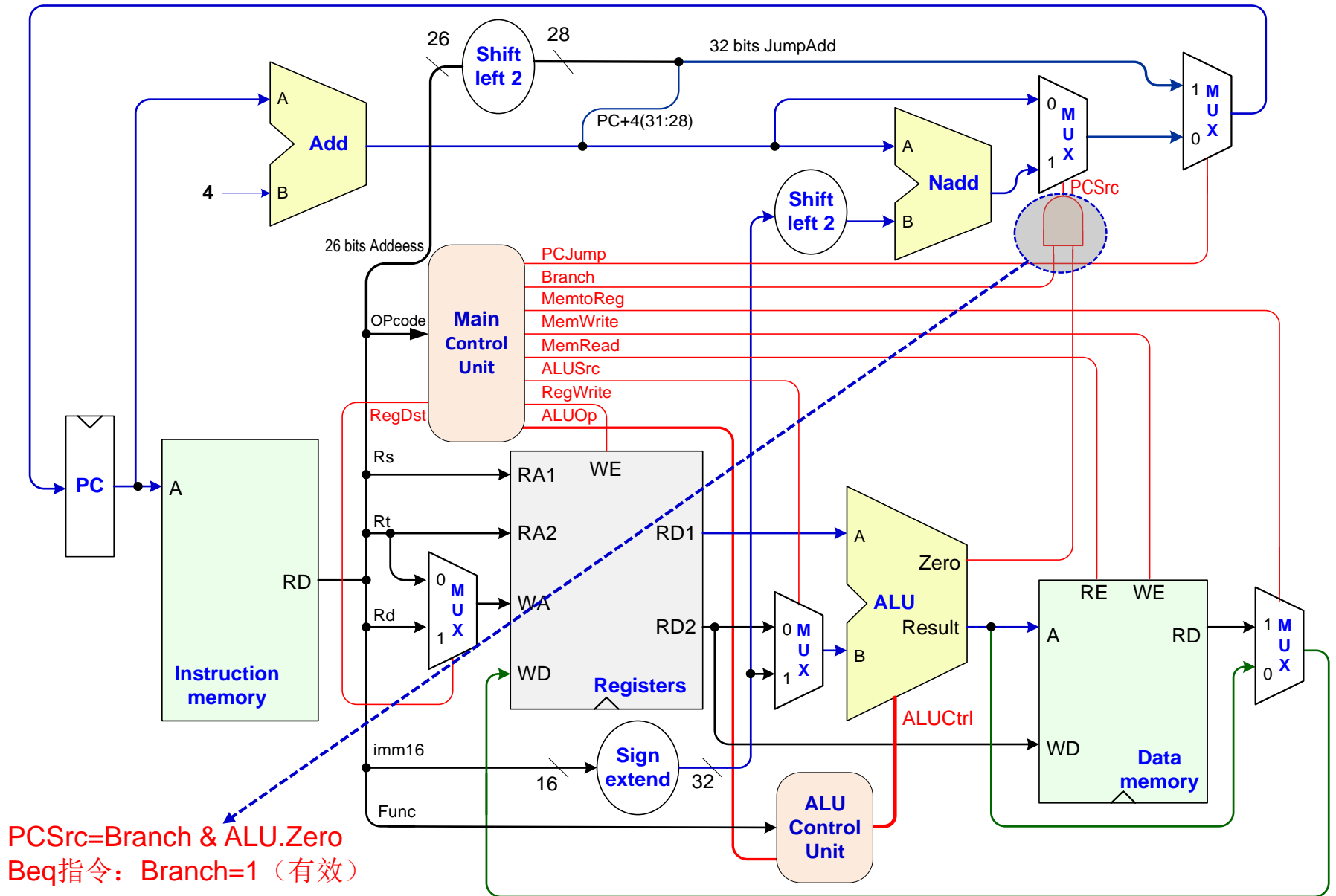
## ❖ 单周期处理器设计

- 单周期数据通路和控制器设计
- 单周期处理器性能分析

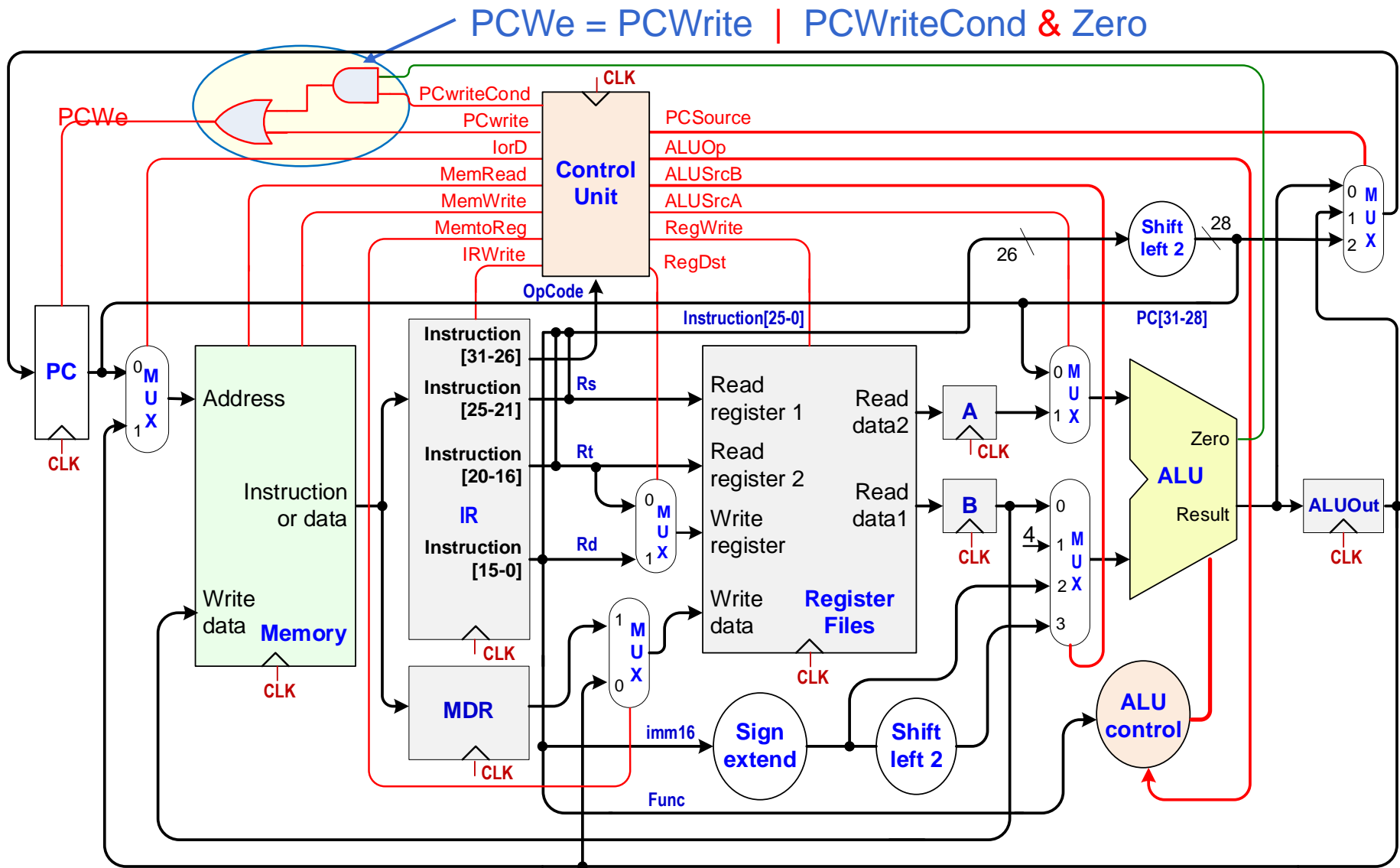
## ❖ 流水线处理器设计

- 流水线数据通路和控制器设计
- 流水线冒险及其处理

# 单周期模型机数据通路 (带控制单元)



# 多周期控制器设计：完整数据通路与控制信号



# 单周期性能与多周期性能

数据通路各部分以及各类指令的执行时间

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

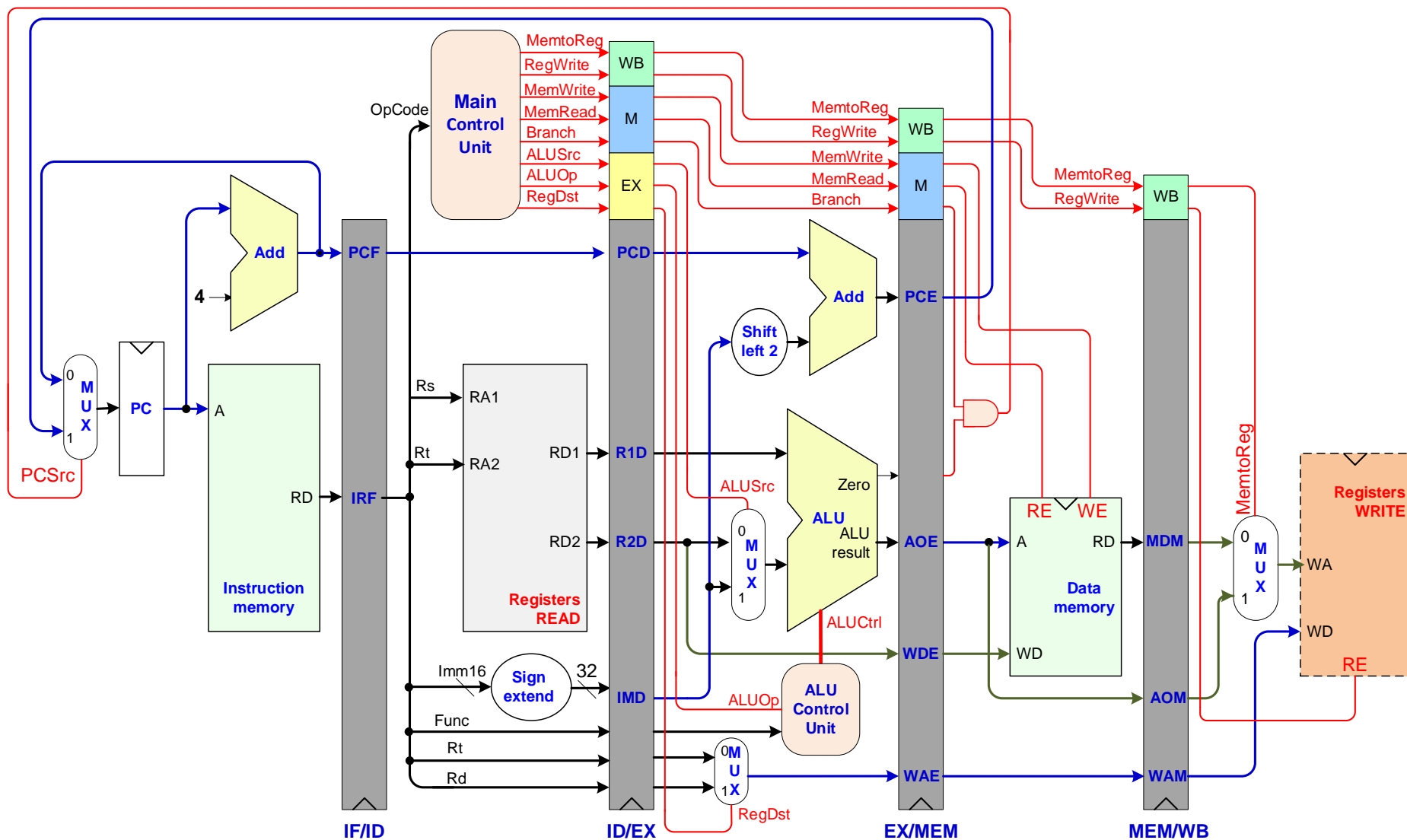
## ❖ 单周期方案

- 所有指令执行周期=时钟周期=600ps
- 某些类型指令本可以在更短时间内完成，造成时间浪费。

## ❖ 多周期方案

- 指令执行分解为多个步骤，每一步一个时钟周期，则指令执行周期为多个时钟周期，不同指令执行的时钟周期数不一样。时钟周期=200ps
- R类型指令4个时钟周期=800ps
- 取数指令5个时钟周期=1000ps
- J指令1个时钟周期=200ps

# MIPS流水线数据通路：控制信号



MIPS流水线数据通路（含控制信号，无转发）

❖ **流水线冒险 (Hazard, 也称流水线相关问题)**：流水线相近指令出现某些关联，下一个时钟周期不能执行下一条指令，指令流水线必须停顿。

➤ **结构冒险 (structural hazard)**：硬件不支持多条指令在同一个时钟周期执行。若系统只有一个存储器部件，就会带来结构冒险问题。

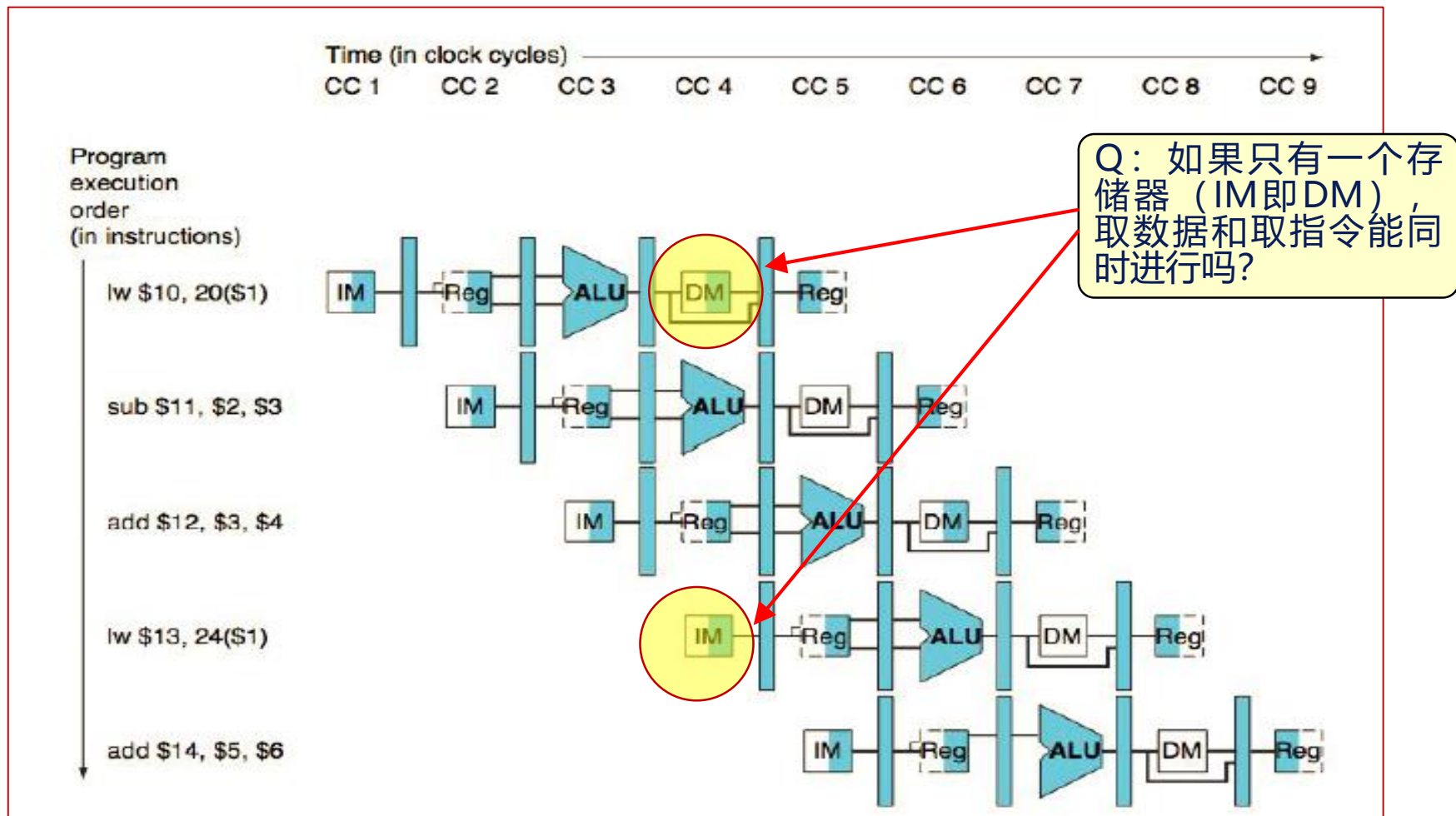
- lw/sw指令执行需要访问存储器，指令取指阶段需要访问存储器，将出现存储器使用冲突

➤ **数据冒险 (data hazard)**：指令执行所需的数据暂时不可用而造成的指令执行的停顿。数据冒险一般发生在相近指令共用一个存储单元或寄存器时（也称数据相关）。

➤ **控制冒险 (control hazard)**：也称为分支冒险 (branch hazard)，必须根据前一条指令的执行结果才能确定下一条真正要执行的指令，此时流水线中取得的可能不是真正要执行的指令。

# 流水线的冒险：结构冒险

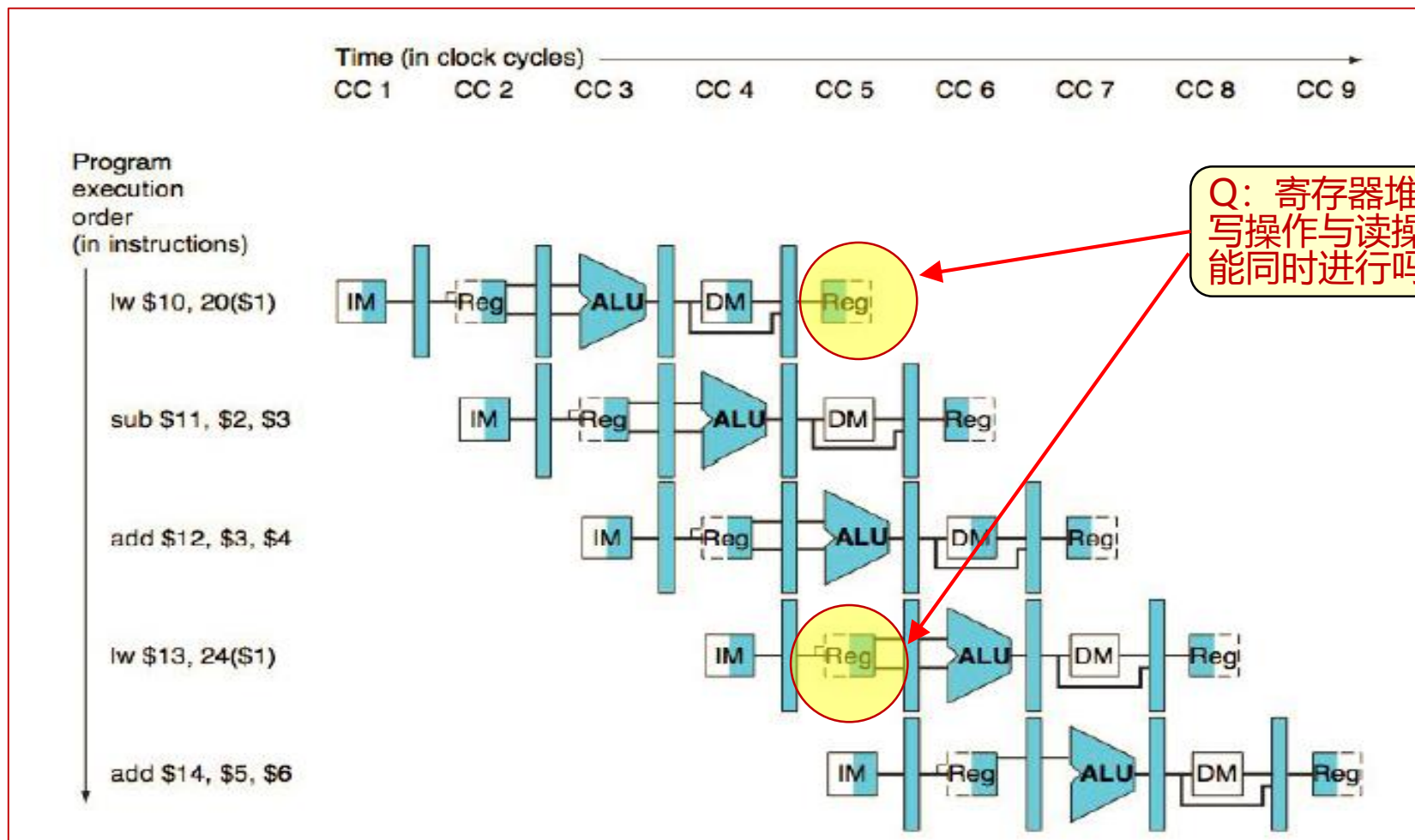
## ❖ 存储器同时访问问题



**A: 假定数据通路只有一个存储器, 此时存储器访问将发生冲突!**

# 流水线的冒险：结构冒险

## ❖ 寄存器堆同时访问问题



**A: 寄存器堆在结构上支持读操作与写操作同时进行。**



# 流水线的冒险：数据冒险

## ❖ 数据冒险 (Data hazard)

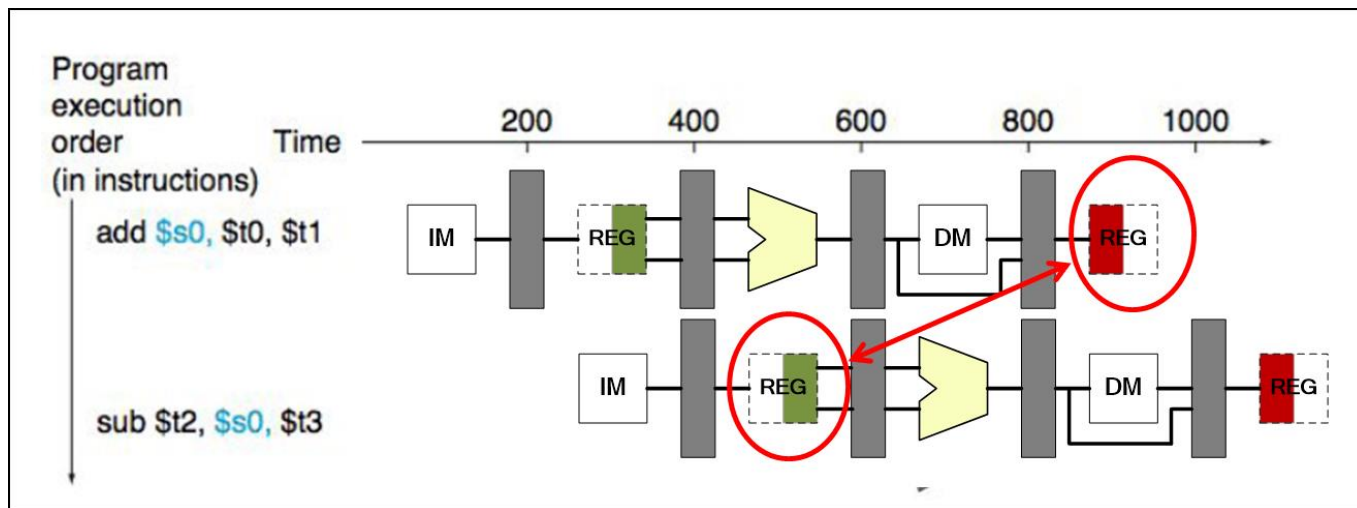
- 后面指令的执行需要使用前面指令的结果（寄存器值），若此时结果尚未形成，带来数据冒险（Read after Write, RAW, 写后读），指 RAW 同一个寄存器的情况，基于产生结果的指令类型分两种类型：

### 1. 运算类指令

- add \$s0, \$t0, \$t1
- sub \$t2, \$s0, \$t3

### 2. 取内存数据指令

- lw \$s0, 100(\$t0)
- sub \$t2, \$s0, \$t3



# 数据冒险的处理：旁路转发

## ❖ 旁路转发策略

指令  
执行  
次序  
↓

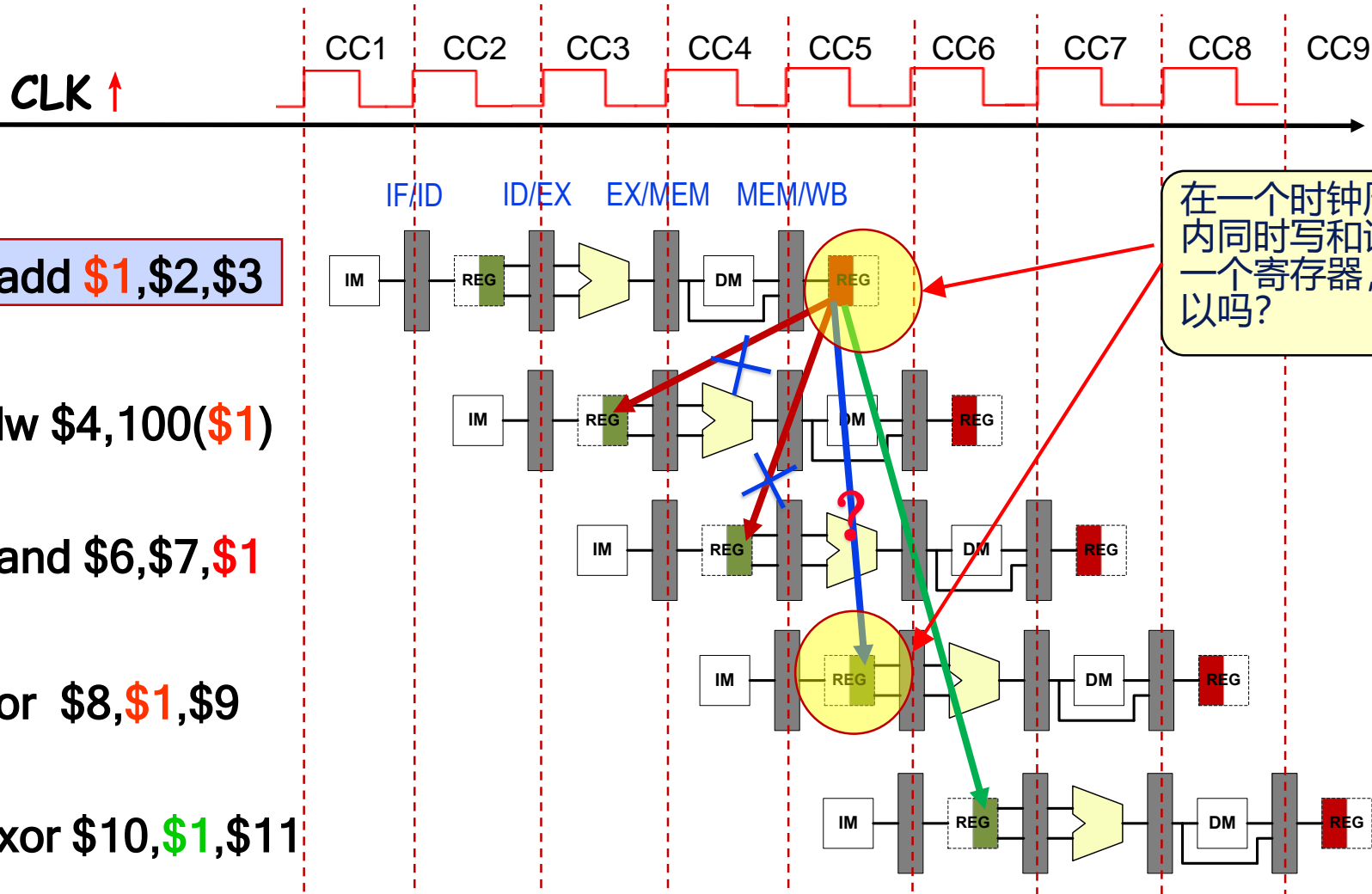
add \$1,\$2,\$3

lw \$4,100(\$1)

and \$6,\$7,\$1

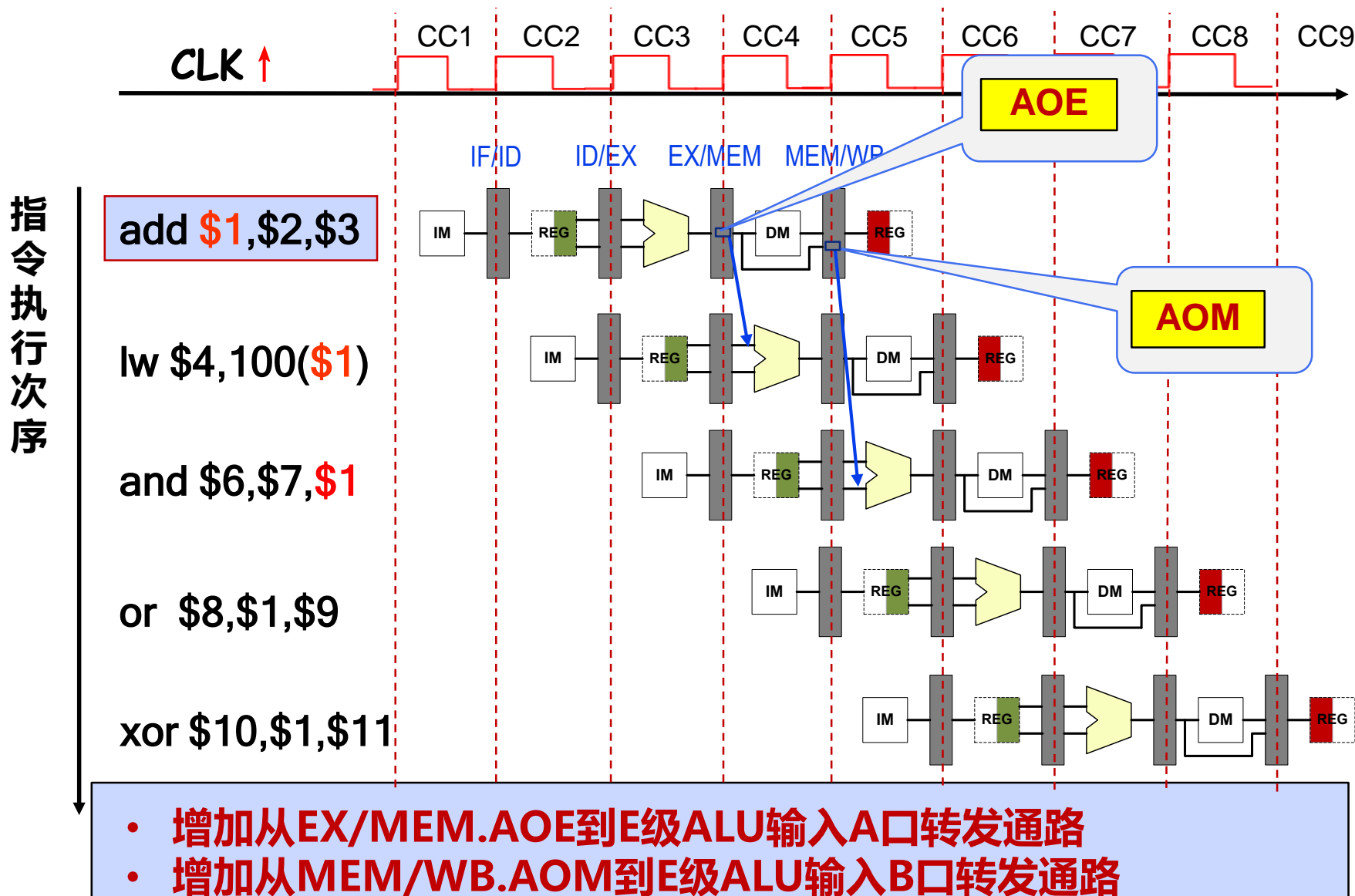
or \$8,\$1,\$9

xor \$10,\$1,\$11



# 数据冒险的处理：旁路转发

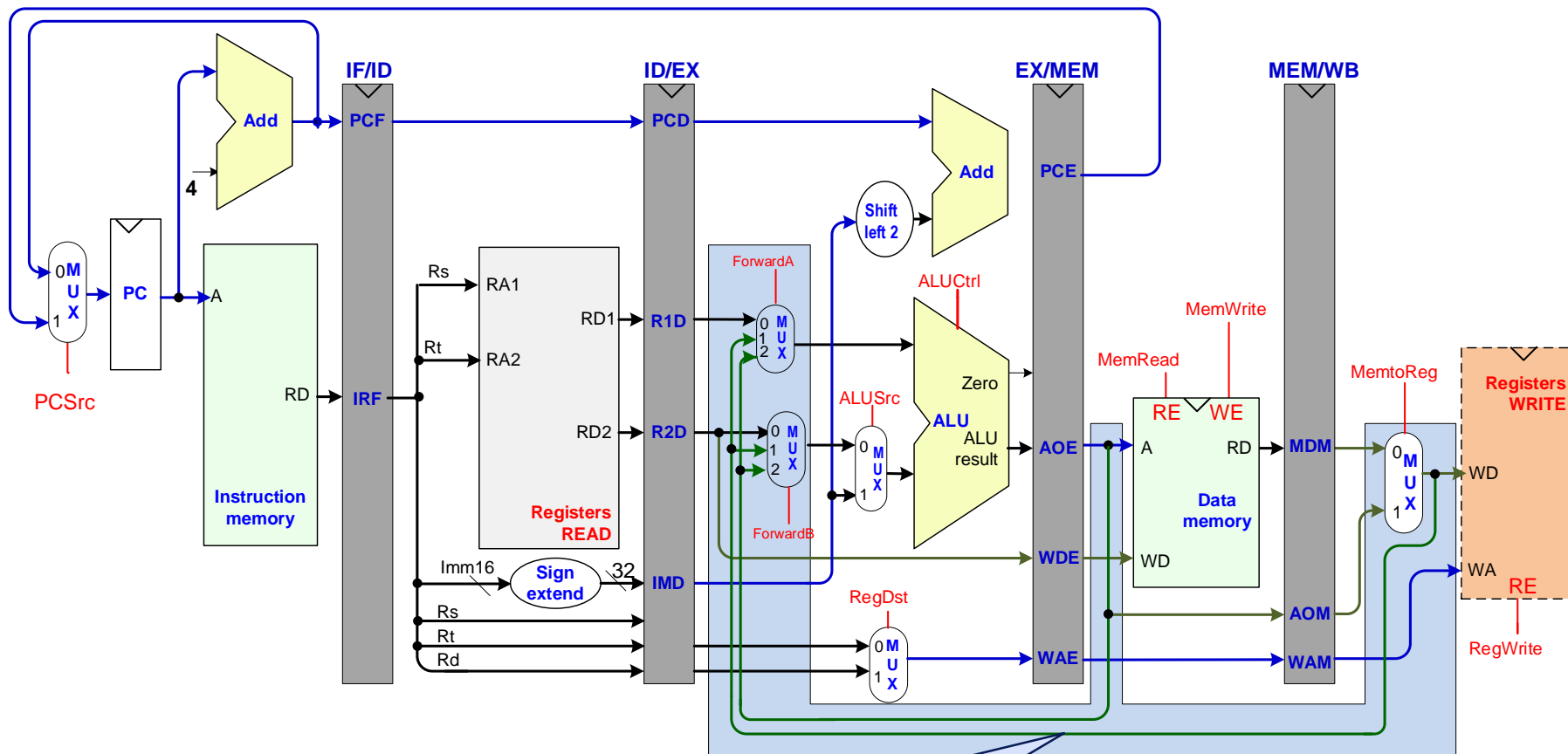
## ❖ 场景一：R类型Rd（写），后续指令Rs或Rt（读）





# 数据冒险的处理：旁路转发

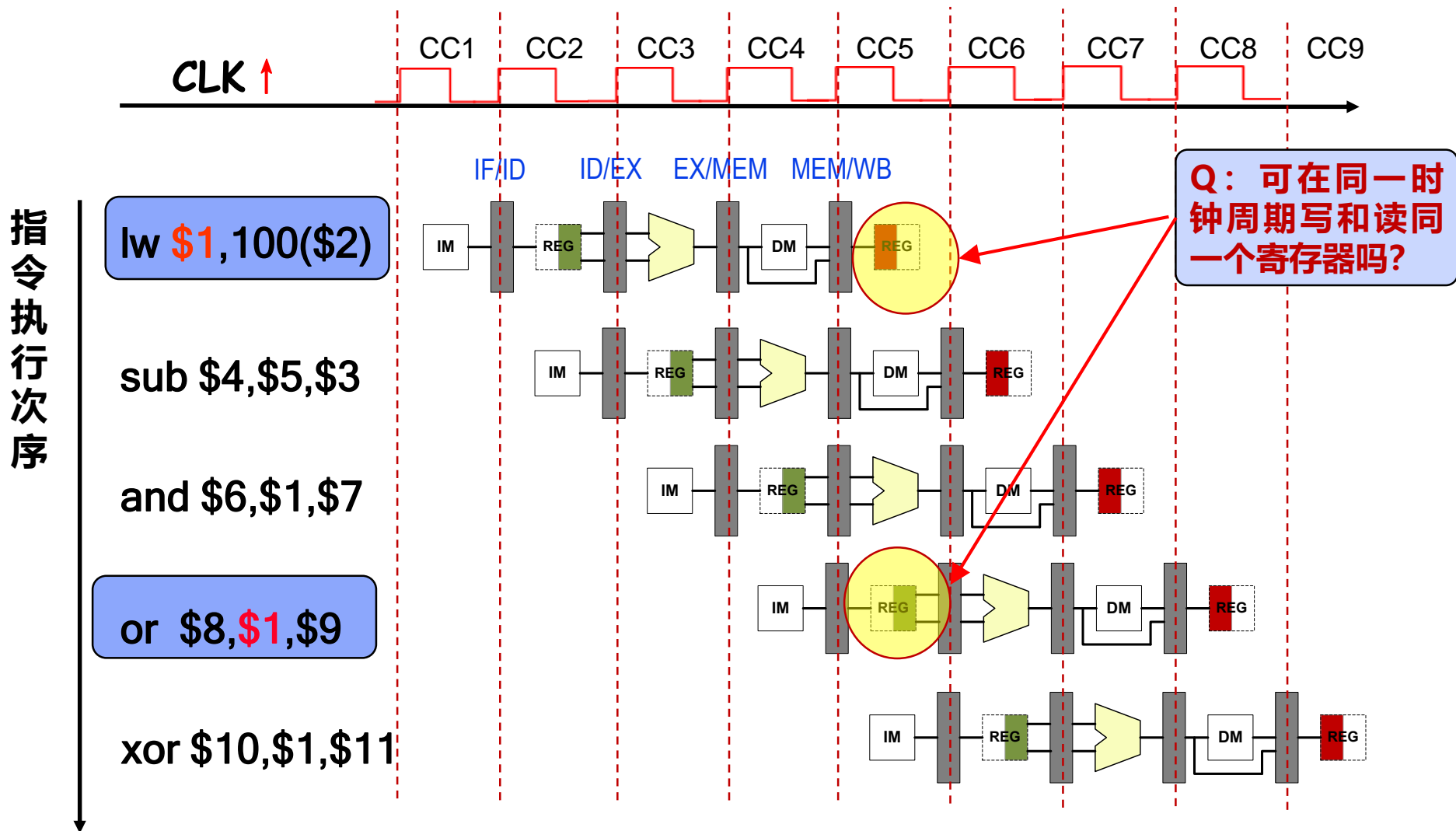
- ❖ 转发综合：增加AOE@EX/MEM、AOM@MEM/WB、MDM@MEM/WB到E级ALU两个输入端的转发通路，以及相应多路选择电路和转发控制电路



MEM/WB.AOM、  
MEM/WB.MDM到E级ALU  
输入端的转发通路合并从  
MUX的输出端转发

# 数据冒险的处理：同时写和读同一个寄存器

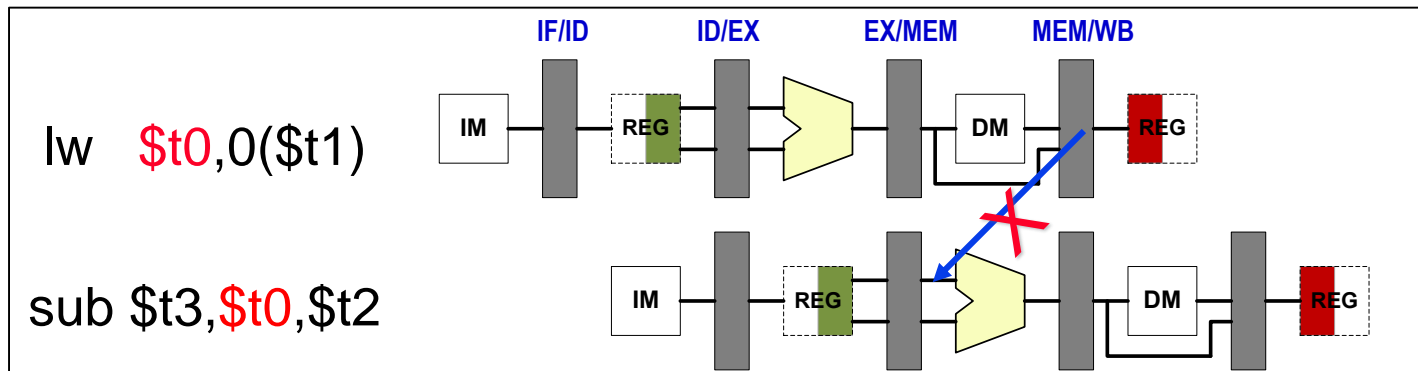
## ❖ 场景三：同一时钟写和读同一个寄存器



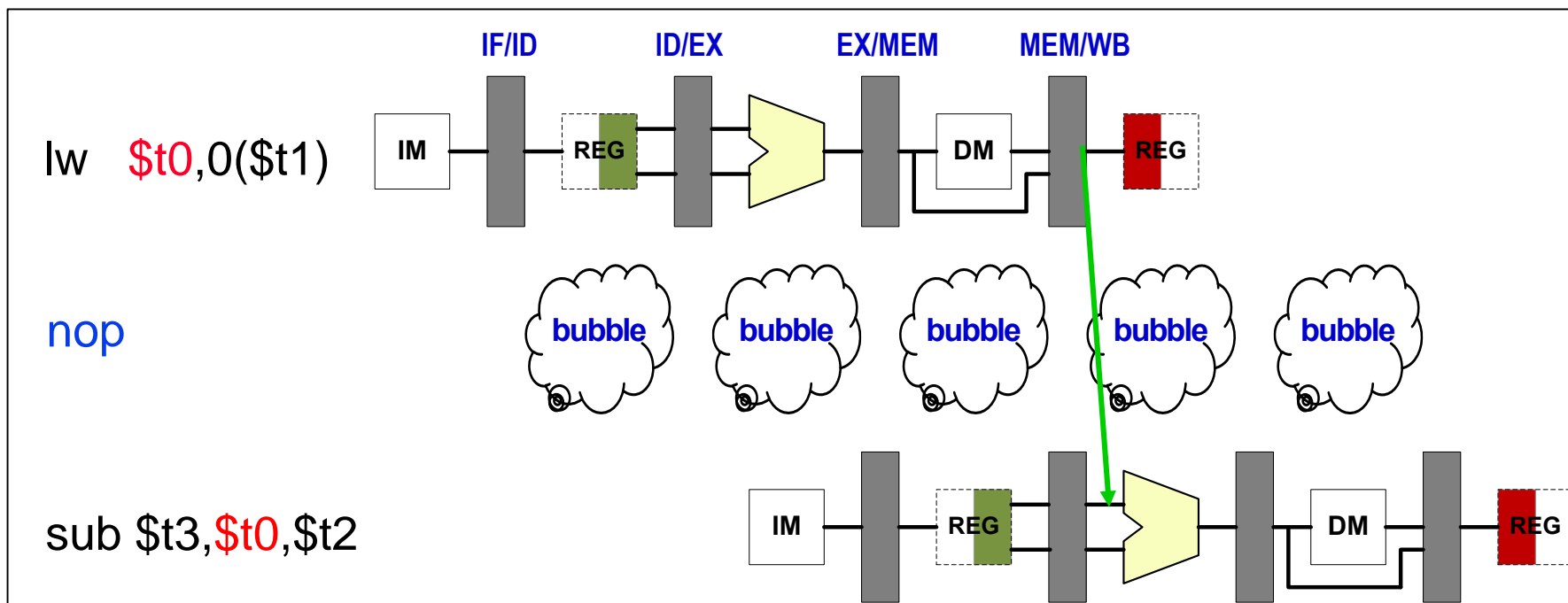
A: 在前半个时钟周期写入可以实现同一个时钟周期先写后读

# 数据冒险的处理：load导致的数据冒险

❖ 场景四：Load指令Rt（写），下一条指令Rs或Rt（读）



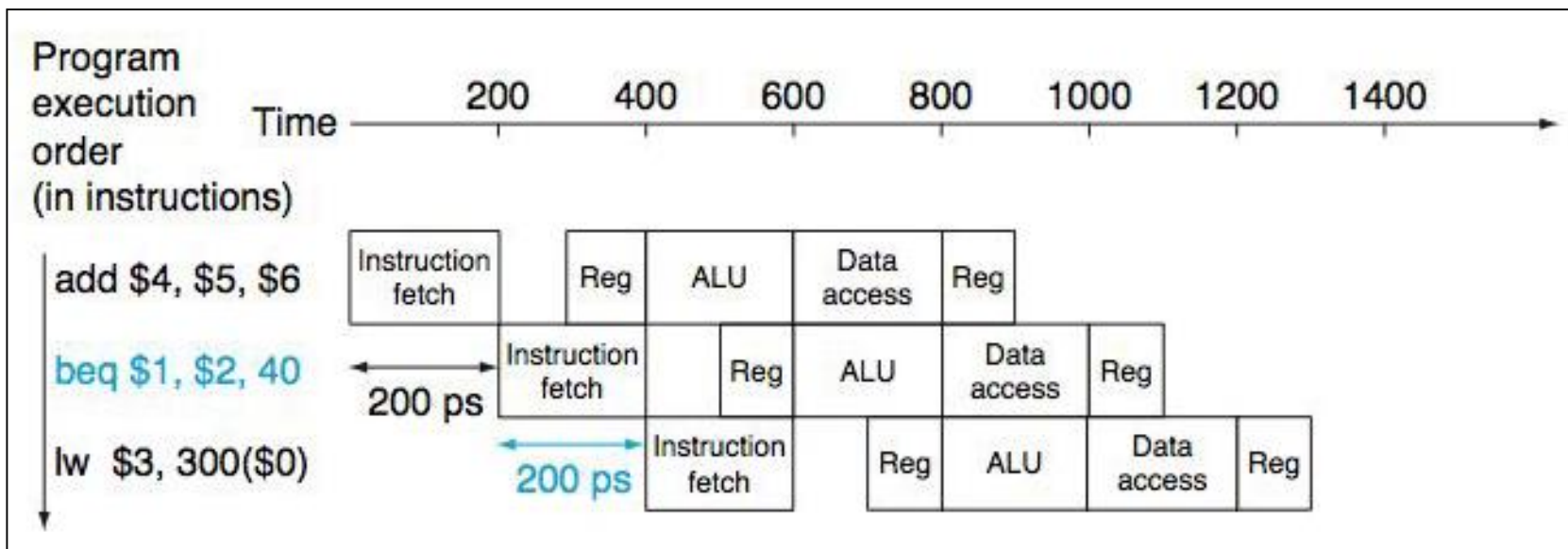
Q: 旁路转发能解决吗?



A: 须在lw指令后停顿一个周期（等价于插入一条NOP），然后再转发

## 流水线的冒险：控制冒险

- ❖ 控制冒险主要由条件转移指令引起，前面指令执行的结果可能会使程序执行发生转移，流水线中提前取来的指令可能不应该被执行。



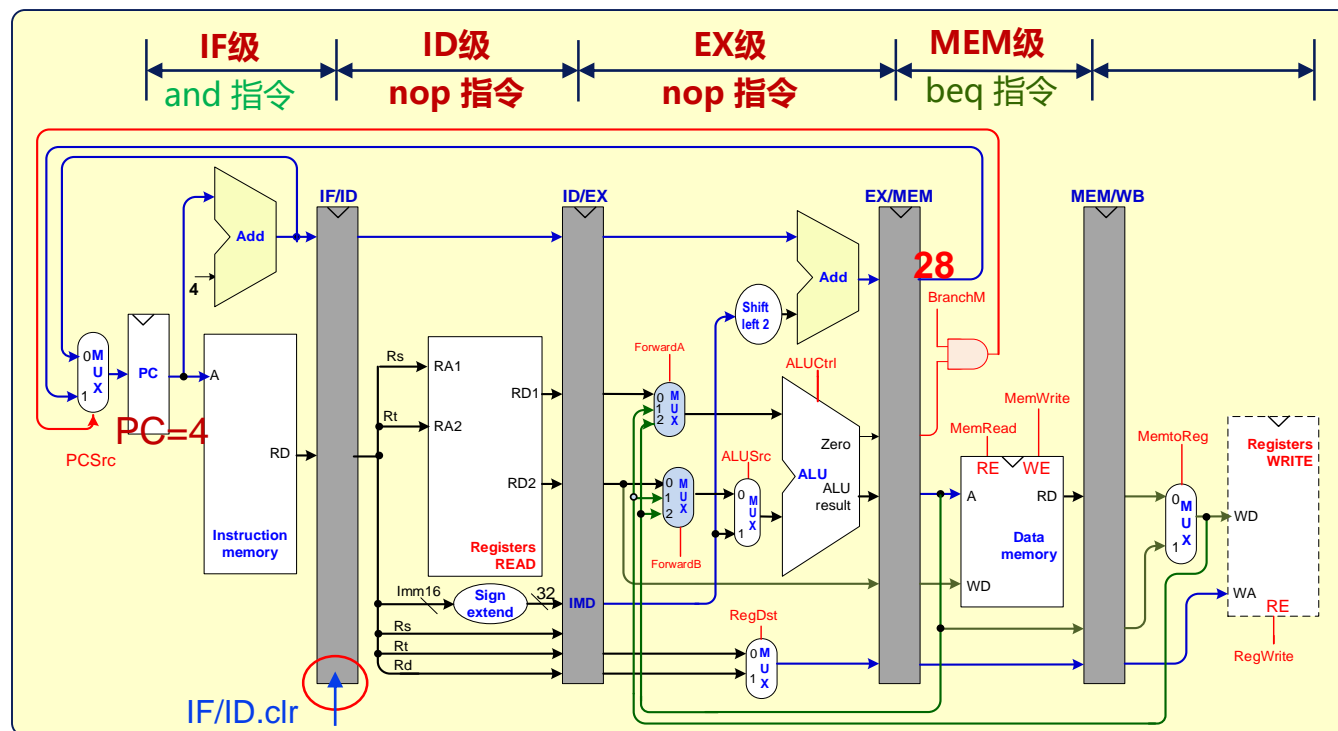
**beq \$1, \$2, 40** 执行时，如果发生条件转移，将不会执行**lw \$3, 300(\$0)**



# 控制冒险的处理：①停顿

		IF级		ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	beq \$1, \$3, lab	↑ 1	0→4	and	beq				
4	and \$12, \$2, \$5	↑ 2	4	and	nop	beq			
8	or \$13, \$6, \$2	↑ 3	4	and	nop	nop	beq结果		
12	add \$14, \$2, \$2	↑ 4	4→28	lw	nop	nop	nop		
		↑ 5	28→32	XXX	lw	nop	nop	nop	nop
28	lab:lw \$4, 50(\$7)								

如不对beq指令做任何处理，  
则**必须从IF/ID插入3个nop**  
**CLK3上升沿**：beq指令结果 (Zero) 及新PC值 (28) 写入EX/MEM  
**CLK4上升沿**：PC新值 (28) 写入PC，IM输出lw指令代码，IF/ID继续插入nop  
**CLK5上升沿**：lw指令代码写入IF/ID



CLK 3时钟周期时的指令流水

## 控制冒险的处理： ②假定分支不会发生

PC相对偏移	指令
0	beq \$1, \$3, lab
4	and \$12, \$2, \$5
8	or \$13, \$6, \$2
12	add \$14, \$2, \$2
28	lab: lw \$4, 50(\$7)

**假定分支不会发生**（50%概率）：ID级发现是beq指令时，不停顿（不插入nop），流水线顺序执行

**Q：结果产生时，若分支发生，怎么办？**

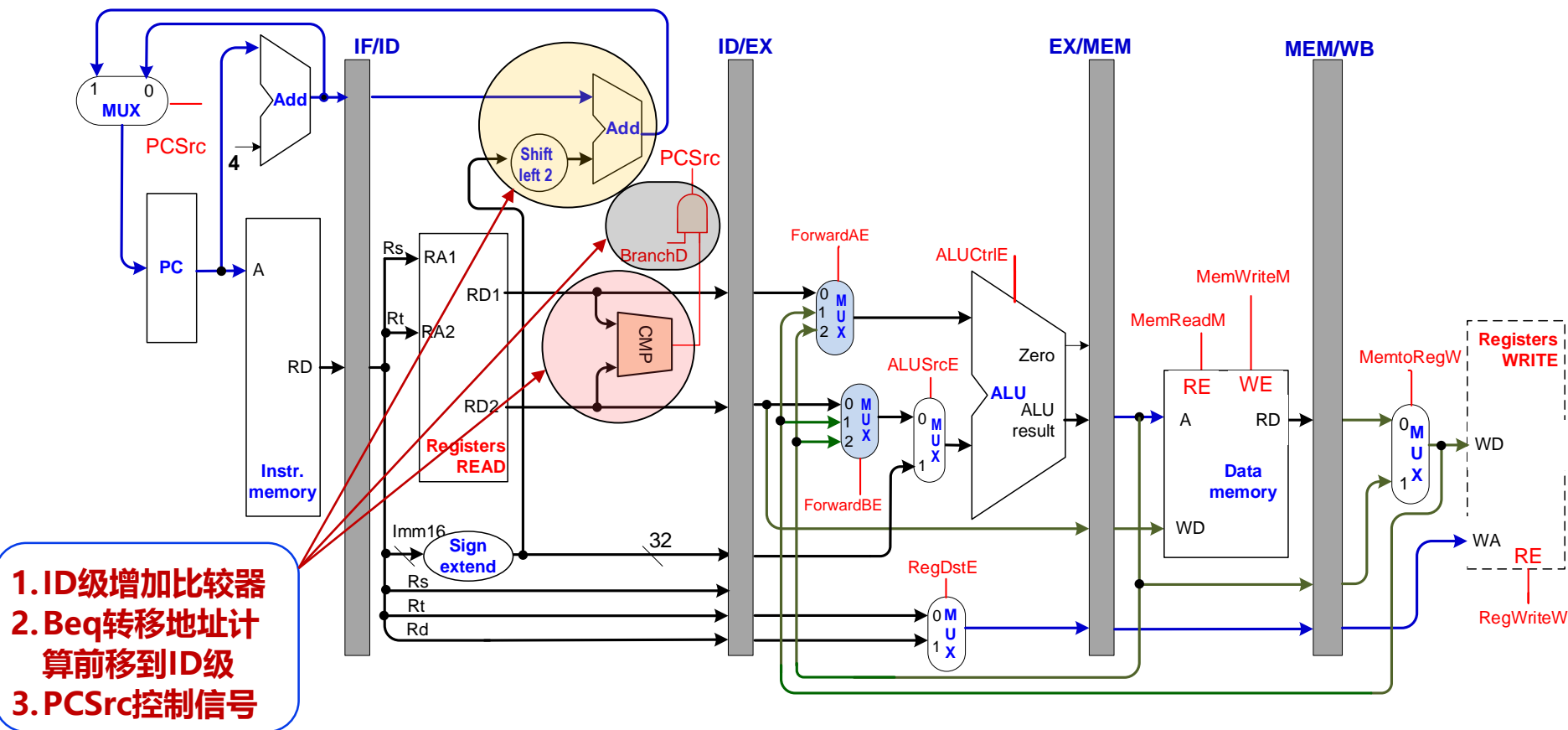
**若分支发生的处理：**后继3条指令已经进入流水线，**必须清除这3条后继指令**

- EX/MEM.clr=1 （清除and \$12, \$2, \$5）
- ID/EX.clr=1 （清除or \$13, \$6, \$2）
- IF/ID.clr=1 （清除add \$14, \$2, \$2）

# 控制冒险的处理：③缩短分支延迟

❖ 在ID级增加比较器，尽快得到beq指令结果

- RF两个数据输出端连接到比较器输入端，实现beq指令的比较功能，
- beq指令结果在ID级可以得到，提前2个clock
- beq指令后继可能被废弃的指令减少为1条，当需要转移时，清除IF/ID即可



Q: 比较器前置后，可能产生什么问题？

# 控制冒险的处理：④分支延迟槽技术

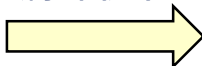
## ❖ Beq指令的处理 (branch delay slot)

### 原始程序

```
sub $4, $5, $6  
add $1, $3, $2  
or $7, $6, $3  
beq $5, $8, Exit  
xor $10, $9, $11
```

Exit: lw \$2, 100(\$2)

编译优化



### 支持延迟槽的程序

```
sub $4, $5, $6  
add $1, $3, $2  
beq $5, $8, Exit  
or $7, $6, $3  
xor $10, $9, $11
```

Exit: lw \$2, 100(\$2)

延迟槽

分支发生时的指令  
执行顺序

### 指令流

分支跳转指令  
↓  
目标地址的指令

### 指令流

分支跳转指令  
↓  
延迟槽指令  
↓  
目标地址的指令



# **第七部分**

## **输入输出系统**

## ❖ I/O方式

- 程序查询方式

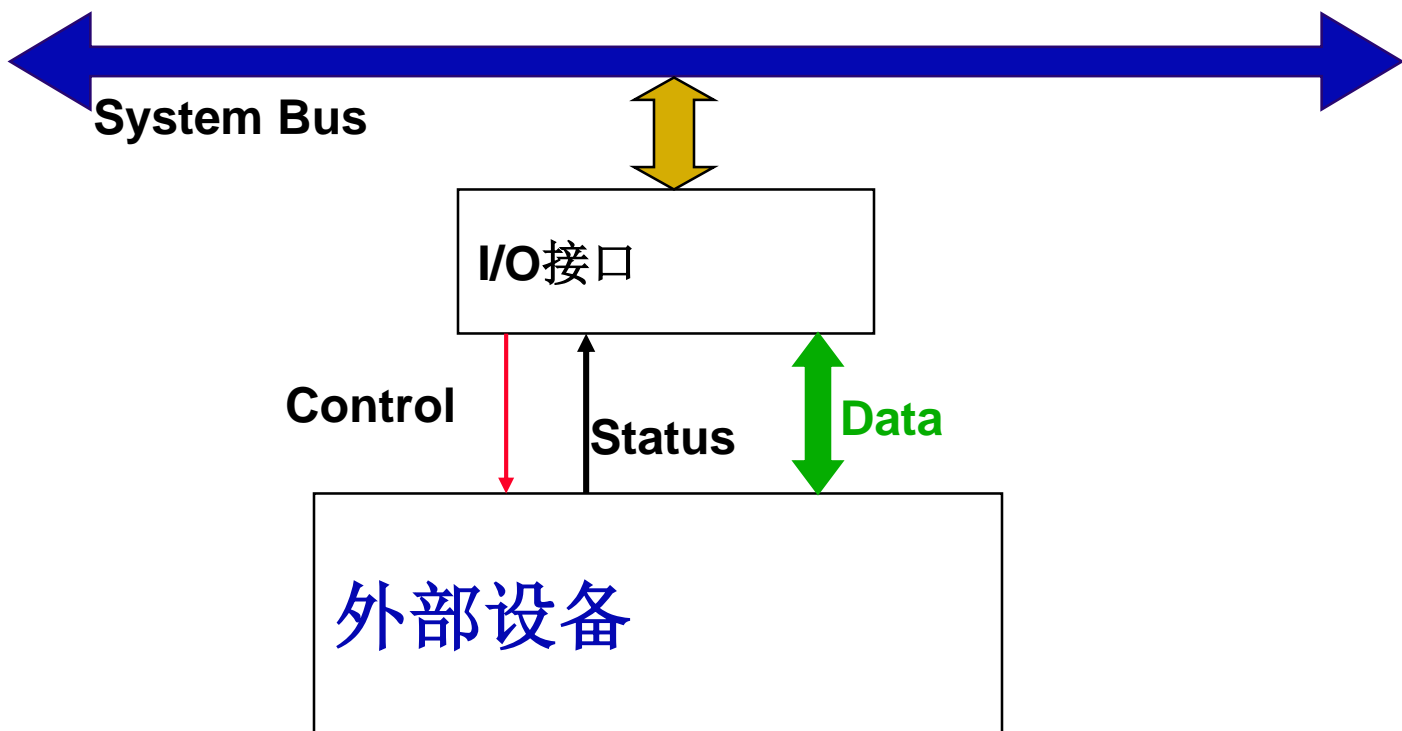
- 中断方式

- DMA方式

## ❖ 外部存储

# I/O接口

- ❖ 外部设备并不直接挂接在系统总线上,而是通过I/O接口为桥梁实现与系统总线的连接
  - 各种外设使用不同的操作方法, 由CPU来直接控制不同的外设不切实际。
  - 外设的数据传送速度比存储器和处理器的速度慢得多, 使用高速的系统总线与慢速的外设直接连接, 不切实际。
  - 外设经常使用与处理器不同的数据格式和字长度。



## ❖ I/O接口的功能

- **设备寻址**：识别I/O地址，即地址译码；
- **数据交互**：实现主机与I/O设备的数据交换；
- **设备控制**：传送CPU的控制命令，实现对外部设备的控制（启动、停止、复位、读、写等）；
- **状态检测**：检测外部设备的工作状态（就绪、忙、故障等状态）；
- **数据缓冲**：提供数据缓冲以匹配CPU与外部设备之间的速度差距；
- **格式转换**：进行数据格式、类型的转换（串并行转换，电平转换等）。



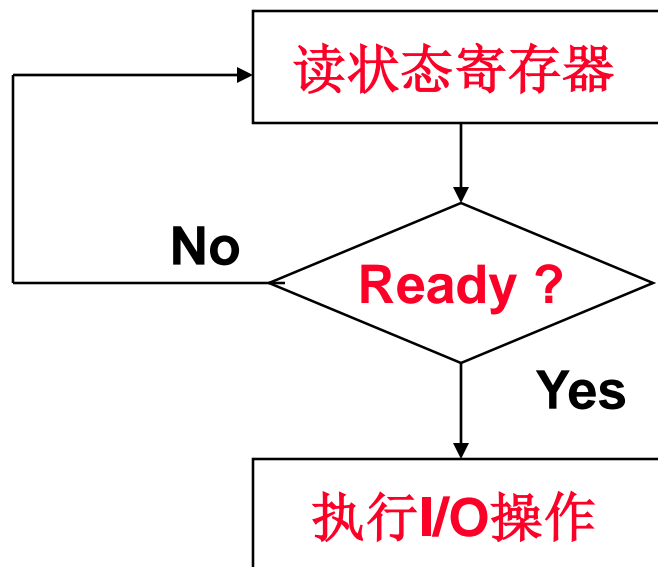
## ❖ I/O方式的演变（CPU从I/O事务中的解放）

- ① **直接控制方式**：**CPU**直接控制外设，主要用于简单的微处理器控制设备；
- ② **程序I/O方式**：增加控制器和I/O模块，处理器使用编程I/O，使处理器从外设的I/O细节中解脱出来；
- ③ **中断I/O方式**：增加控制器和I/O模块，采用中断I/O方式，处理器不需要浪费时间等待I/O操作完成，提高了处理器的效率；
- ④ **DMA方式**：I/O模块通过DMA直接存储存储器，除在传输开始和结束时，传输数据不需要处理器参与；
- ⑤ **I/O通道方式**：I/O模块成为有自主控制权的处理器，有处理I/O的专用指令集。**CPU**指示I/O处理器执行存储器中的I/O程序，I/O处理器不需要**CPU**干预就能获取并执行I/O指令。这允许**CPU**指派一系列的I/O活动，并只在整个活动执行完成后才中断**CPU**；
- ⑥ **I/O处理器方式**：I/O模块带局部存储器，成为自治的计算机。这种结构可以控制大量的I/O设备而最小化**CPU**的干预。

# 程序查询方式

- ❖ I/O接口设置状态寄存器以表示外部设备的工作状态
- ❖ CPU通过不断读取状态寄存器以查询外部设备的状态
- ❖ 在外部设备准备就绪的时候，CPU通过I/O接口中的数据寄存器与外设完成数据交换。

```
RdSta:  MOV DX,3FDH  
        IN  AL,DX  
        CMP AL,61H  
        JNE RdSta  
        MOV DX,3F8H  
        IN  AI,DX
```

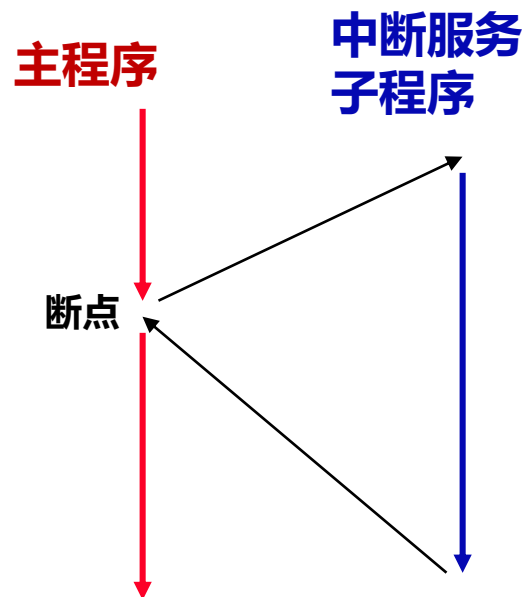


## ❖ 中断的概念

- 中断是指在计算机执行程序的过程中，出现某些急需处理的异常情况或特殊请求，CPU暂时停止执行现执行程序，而转去处理这些异常情况或特殊请求，处理完毕后，CPU自动返回**断点**继续执行被暂停的程序。
- **主程序**：被中断的程序；
- **中断服务子程序**：处理中断事务的程序。

## ❖ 中断系统

- 中断系统是计算机实现中断功能的软、硬件总称。在CPU一侧配置了中断机构，在设备一侧配置了中断控制接口，在软件上设计了相应的中断服务程序



# 中断的分类

## ❖ 硬件中断与软件中断

- **硬件中断**：由硬件引起的中断，外部中断和内部异常中的终止异常属于硬件中断；
- **软件中断**：由程序执行指令引起的中断，内部异常中的自陷和故障属于软件中断；

## ❖ 自愿中断和强迫中断

- **自愿中断**：不是随机事件，是程序中事先安排的，如内部异常中的自陷，X86中的INT调用，MIPS中的SYSCALL调用等；
- **强迫中断**：随机产生的，如外部中断。

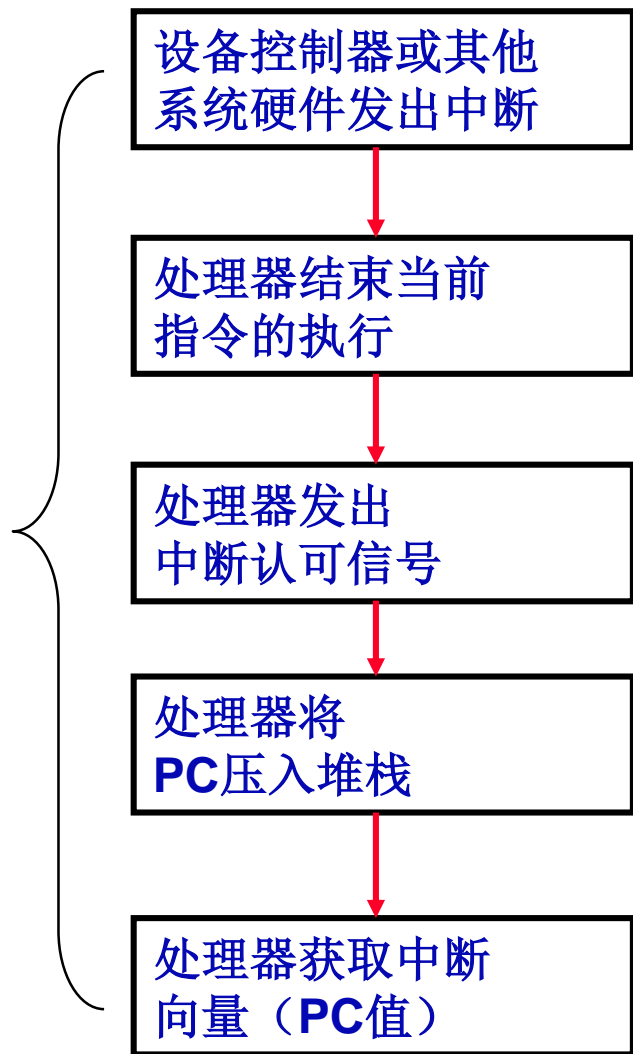


## ❖ 中断系统需要解决的问题

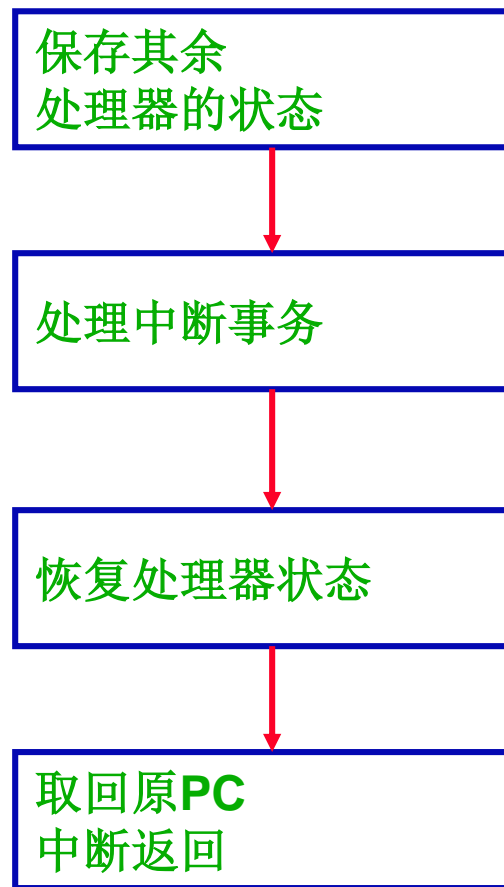
- 中断源如何向CPU提出中断申请;
- 多个中断同时申请时, 中断系统如何响应;
- CPU响应中断的时间、条件和方式;
- CPU响应中断后如何保护现场;
- CPU响应中断后, 如何转向中断服务子程序;
- 中断处理结束后, CPU如何恢复现场返回主程序断点位置;
- 中断处理过程中出现新的中断申请怎么处理。

# 中断响应与处理

硬件实现



中断服务子程序软件实现



## ❖ 程序I/O与中断I/O的不足

- I/O传送速度受处理器测试和服务设备速度的限制
- 处理器直接负责I/O，对于每一次I/O传送，处理器必须执行一些指令。
- 考虑批量（数据块）传送：
  - 程序I/O方式：处理器做不了其他工作；
  - 中断I/O方式：I/O传输效率较低。

## ❖ DMA (Direct Memory Access)

- CPU对总线的控制被临时禁止。
- DMA控制器接管总线控制权，控制数据直接在存储器与外设之间高速交换，CPU不再介入具体的I/O操作，由DMA控制器来负责提供存储器地址信号、读写控制信号等。
- CPU与I/O设备在更大的程度上并行工作，效率更高。
- **DMA方式适合高速批量的数据传输**，如视频显示刷新、磁盘存储系统的读写，存储器到存储器的传输等。

## ❖ DMA方式

### ➤ 周期窃取方式（单字传送方式）

- 每次DMA请求得到响应后，DMA控制器窃取一个总线周期完成一次数据传送，然后释放总线。
- 一般适应存储器速度远高于I/O设备速度的情况。

### ➤ 停止CPU访问内存（成组传送方式）

- 一次DMA请求得到响应后，DMA控制器完全占用总线，进行多次DMA传送，直到所有数据传送完毕才释放总线，这段时间完全停止CPU访问内存。
- 适应高速外设与存储器交换数据的情况。



## ❖ CPU的工作：初始化DMA控制器

- 设置数据传送方向：是请求读还是请求写（对存储器而言）
- 设置I/O接口地址：DMA操作所涉及的I/O接口的地址
- 设置存储器起始地址：读或写存储器的起始单元地址
- 设置传送的数据数量：传送数据的字数
- 有关中断方式的设置：DMA结束后通过中断方式请求CPU处理

## ❖ DMA请求

- 当接口做好数据传输的准备，通过有关逻辑向CPU发出DMA请求信号。

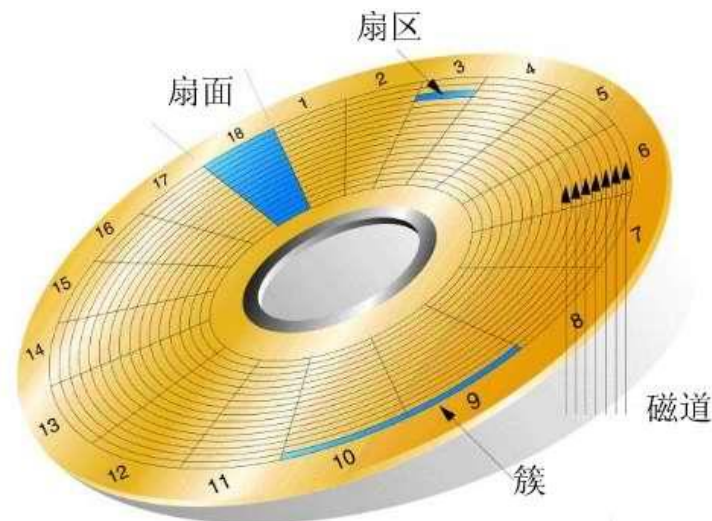
## ❖ DMA响应

- CPU接到DMA请求，在当前总线周期操作结束后，暂停CPU对系统总线的控制和使用，发出DMA响应信号，并交出系统总线的控制权。

# 硬盘基本结构

## ❖ 磁盘存储结构（恒定角速度，CAV）

- ▶ 盘面：一个磁盘包含若干盘片，每盘片分上下两个盘面，每个盘面由一磁头负责读写
- ▶ 磁道：磁盘表面的同心圆环
- ▶ 扇区：每个磁道包含若干扇区，扇区是磁盘数据读写的最小单位，扇区容量一般为512 ~ 4096 字节（默认为512字节，每个磁道包含相同数量扇区）
- ▶ 柱面（cylinder）：不同盘面同一半径上的磁道构成一个柱面
- ▶ CAV缺点：内外磁道存储数据量一致，外围磁道存储数据容量受内圈磁道存储密度限制



磁盘上的磁道、扇区和簇

## ❖ 扇区的地址表示：

扇区地址：

Cylinder #	Head #	Sector #
------------	--------	----------

## ❖ 性能指标

### ➤ 记录密度

- 道密度：磁盘沿半径方向单位长度的磁道数；
- 位密度：单位长度磁道记录二进制的位数。

### ➤ 存储容量：磁头数×磁道（柱面）数×每道扇区数×每扇区字节数

### ➤ 旋转速率： $r$ ，单位是转/秒

### ➤ 寻道时间 $T_S$ ：磁头从当前位置定位到目标磁道所需时间（用平均值表示）；

### ➤ 寻区时间 $T_w$ ：也称为旋转延迟，磁头定位到目标磁道后，等待目标扇区旋转到磁头下所需的时间（用平均值表示： $1/2r$ ）；

### ➤ 访问时间（也称寻址时间） $T_A$ ： $T_A = T_S + T_w = T_S + 1/2r$

### ➤ 数据传输率 $D_r$ ：单位时间内传输的数据位数（b/s）

### ➤ 传输b个字节所需的平均访问时间：平均寻道时间+平均旋转延迟+传输时

## ❖ 成绩评定

- 理论部分成绩: **A**, 期末考试卷面成绩 (**满分100分**)
- 实验部分成绩: **B**, 成绩评定办法详见实验课程 (**满分100分**)
- 平时作业成绩: **C**, 平时作业完成情况 (**满分10分**)
- **总成绩 =  $A*50\% + B*40\% + C$**

# 理论部分期末考试

## ❖ 总体原则：

- 难度相比往年：难度相对去年降低

## ❖ 试题类型

- 选择填空题
- 简单计算题、简单回答题、简单设计
- 汇编程序分析题
- 基础逻辑电路设计题
- MIPS处理器分析题

## ❖ 分数分布

- 基本概念选择填空：20（10小题各2分），单选
- 数字逻辑部分：25（设计分析15分+简答题5分+简答题5分）
- 指令系统与MIPS汇编语言：15（指令简答题5分+汇编题10分）
- 存储系统：25（主存题10分+ Cache题10分+虚存简答题5分）
- MIPS处理器分析：15分

## ❖ 考试题目类型

- 选择题：主要考察各部分内容最重要的知识点
- 简单题：主要考察对基本原理方法的理解与掌握
- 分析与计算题：主要考察对核心内容的理解、掌握与使用

## ❖ 不同程度的要求

- 有些内容需要简要了解即可
- 有些内容需要理解原理和过程，并掌握分析技巧
- 有些内容则需要融会贯通

