

计算机组成原理 (2014级)

计算机组成原理课程组

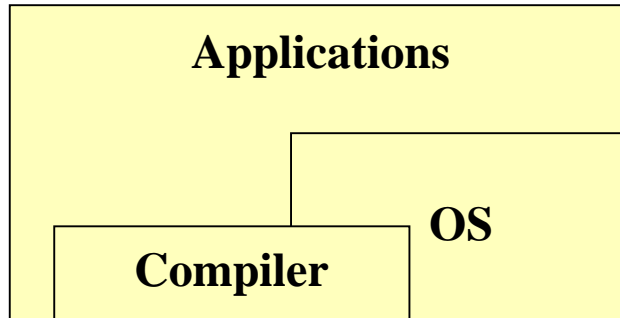
(刘旭东、高小鹏、肖利民、牛建伟、栾钟治)

Tel : 82316285

Mail: liuxd@buaa.edu.cn

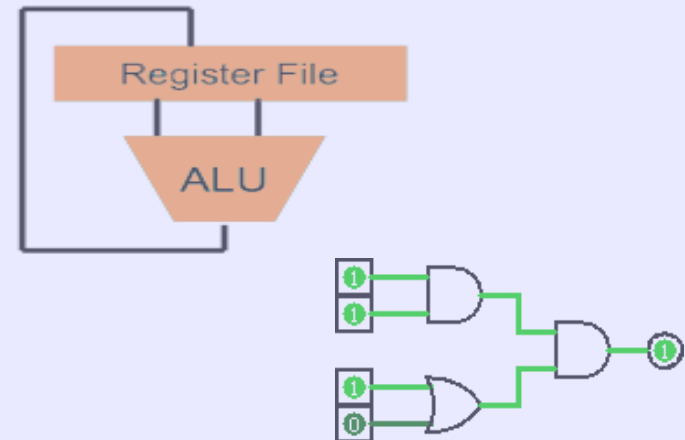
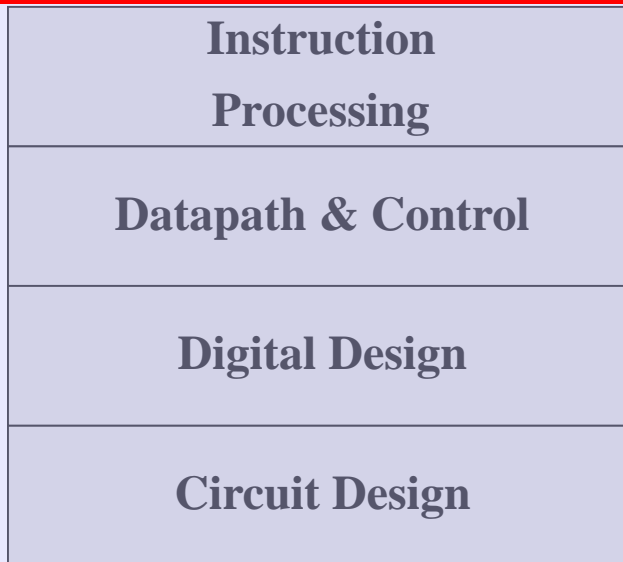
liuxd@act.buaa.edu.cn

Software
layers of
abstraction



Instruction Set Architecture (**ISA**)

Hardware
layers for
design
abstraction



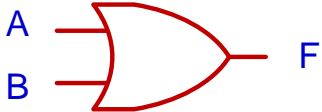
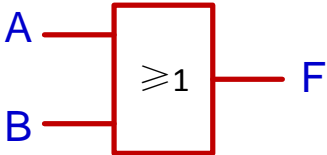
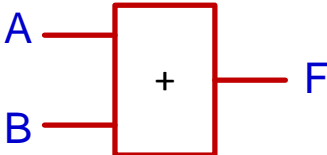

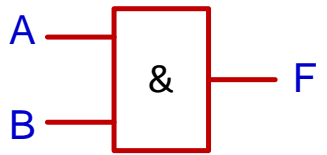
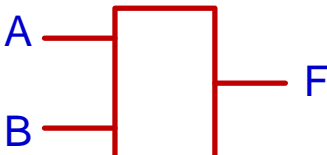
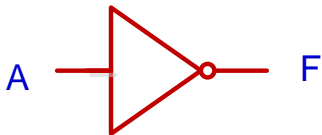
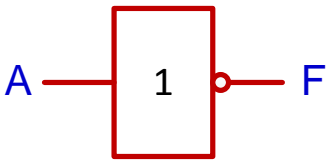
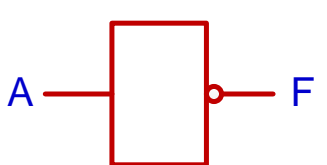
第一部分：基本逻辑电路

- ❖ 组合逻辑电路

- ❖ 时序逻辑电路

数字逻辑的数学基础：布尔逻辑代数

❖ 逻辑电路的符号表示

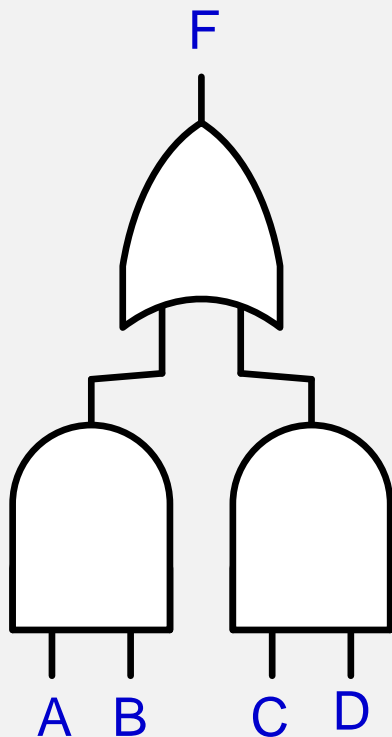
逻辑门	IEEE标准符号	国标符号	部标符号	逻辑表达式
或				$F = A + B$ $F = A \vee B$
与				$F = A \cdot B$ $F = A \wedge B$
非				$F = \overline{A}$ $F = \neg A$

逻辑函数的常用表达式

❖ 常用表达式包括：与或式、或与式、与或非式

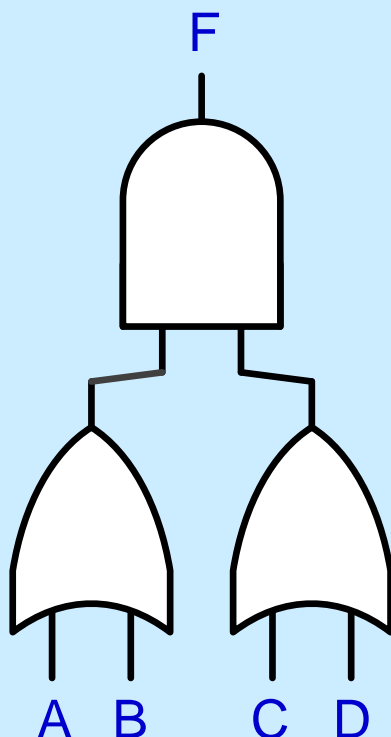
与或式

$$F = AB + CD$$



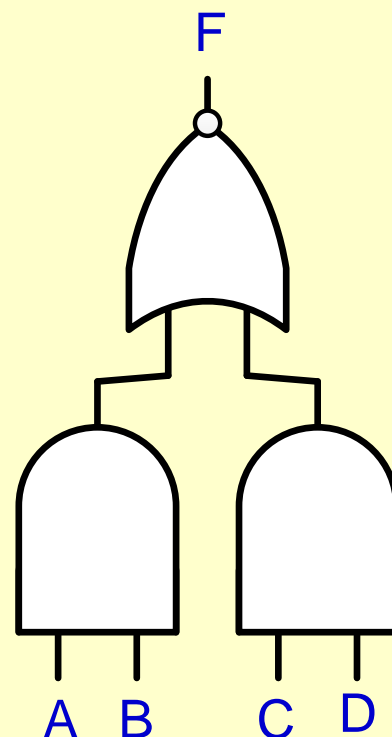
或与式

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



与或非式

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



逻辑函数的标准表达式：从真值表到逻辑表达式

❖ 最小项表达式 或 最大项表达式

例：三人表决器设计（表决原则：少数服从多数）

输入：A, B, C, 1表示赞成, 0表示反对

输出：F, 1表示通过, 0表示不通过

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

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

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

核心：

1. 从问题描述中提取输入输出变量；
2. 从规则描述中提取输出与输入之间的逻辑关系。

逻辑函数化简

❖ 最简与或式

- 1、或项个数最少（或门用的最少）；
- 2、在满足1的条件下，或项中变量数最少（或门的输入端最少）。

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

解：
$$\begin{aligned} F &= AB + \overline{A}\overline{B}C + BC(A + \overline{A}) \\ &= AB + \overline{A}\overline{B}C + ABC + \overline{A}BC \\ &= (AB + ABC) + (\overline{A}\overline{B}C + \overline{A}BC) \\ &= AB(1 + C) + \overline{A}C(B + \overline{B}) \\ &= AB + \overline{A}C \end{aligned}$$

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

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

逻辑函数化简

❖ 最简或与表达式

- 1、或项个数最少（或门用的最少）；
- 2、在满足1的条件下，或项中变量数最少（或门的输入端最少）。

➤ 化简方法

- 1、利用对偶规则，将“或与”表达式转换为“与或”表达式。
- 2、实际化简“与或”表达式。
- 3、利用对偶规则将最简“与或”表达式转为最简“或与”表达式。

【例】化简

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

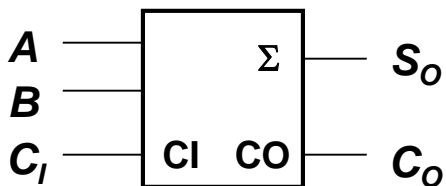
对偶规则

$$\begin{aligned} F' &= AB + \bar{A}C + BC + AC \\ &= AB + \bar{A}C + AC \\ &= AB + C \end{aligned}$$

则： $F=(A+B) \cdot C$

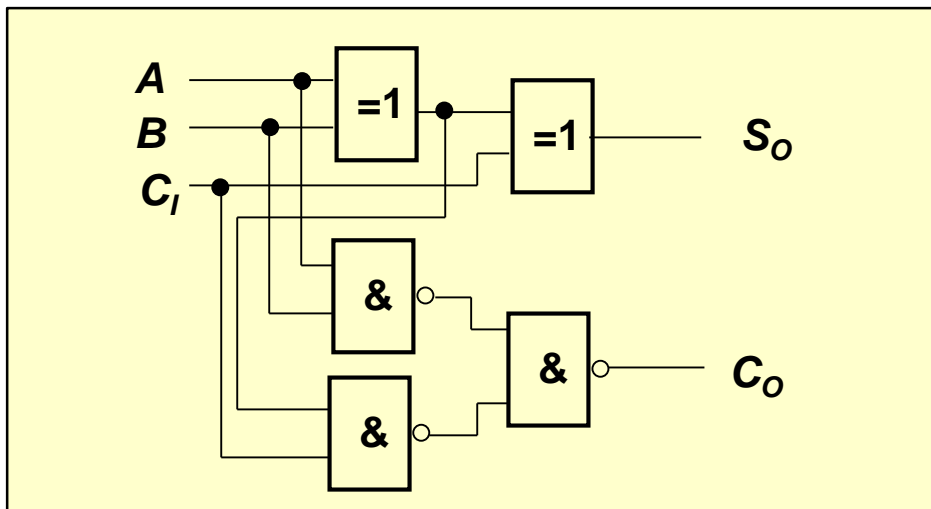
1. 基本组合逻辑电路：ALU

全加器



$$S_o = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i$$

$$C_o = \bar{A}BC_i + A\bar{B}C_i + AB\bar{C}_i + ABC_i$$



全加器真值表

$AB C_i$	S_o	C_o
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1
1 1 1	1	1

1. 基本组合逻辑电路：ALU

❖ 可执行1位与、或、或非、与非、加、减运算

➤ AND运算

- Ainvert=0
- Binvert=0
- Operation=0

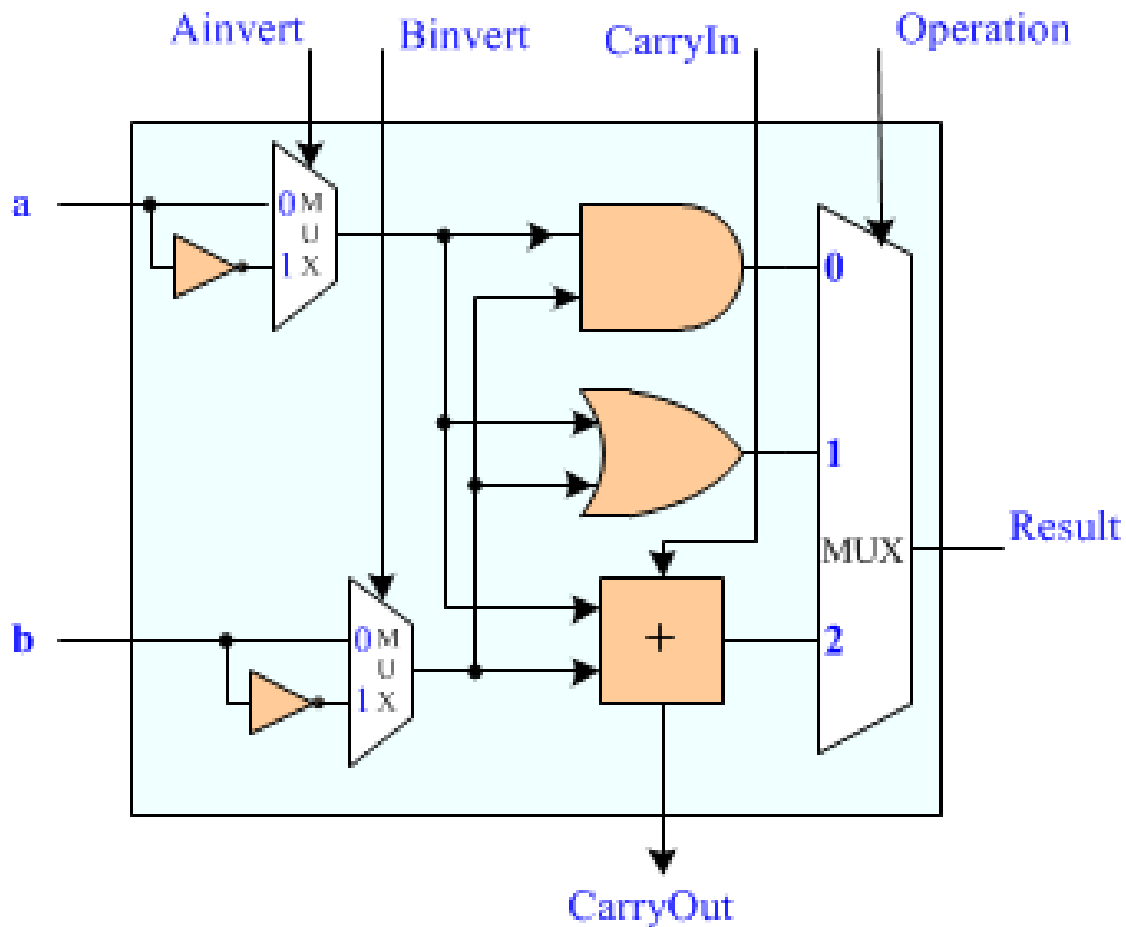
➤ OR运算

- Ainvert=1
- Binvert=1
- Operation=0

➤ SUB运算

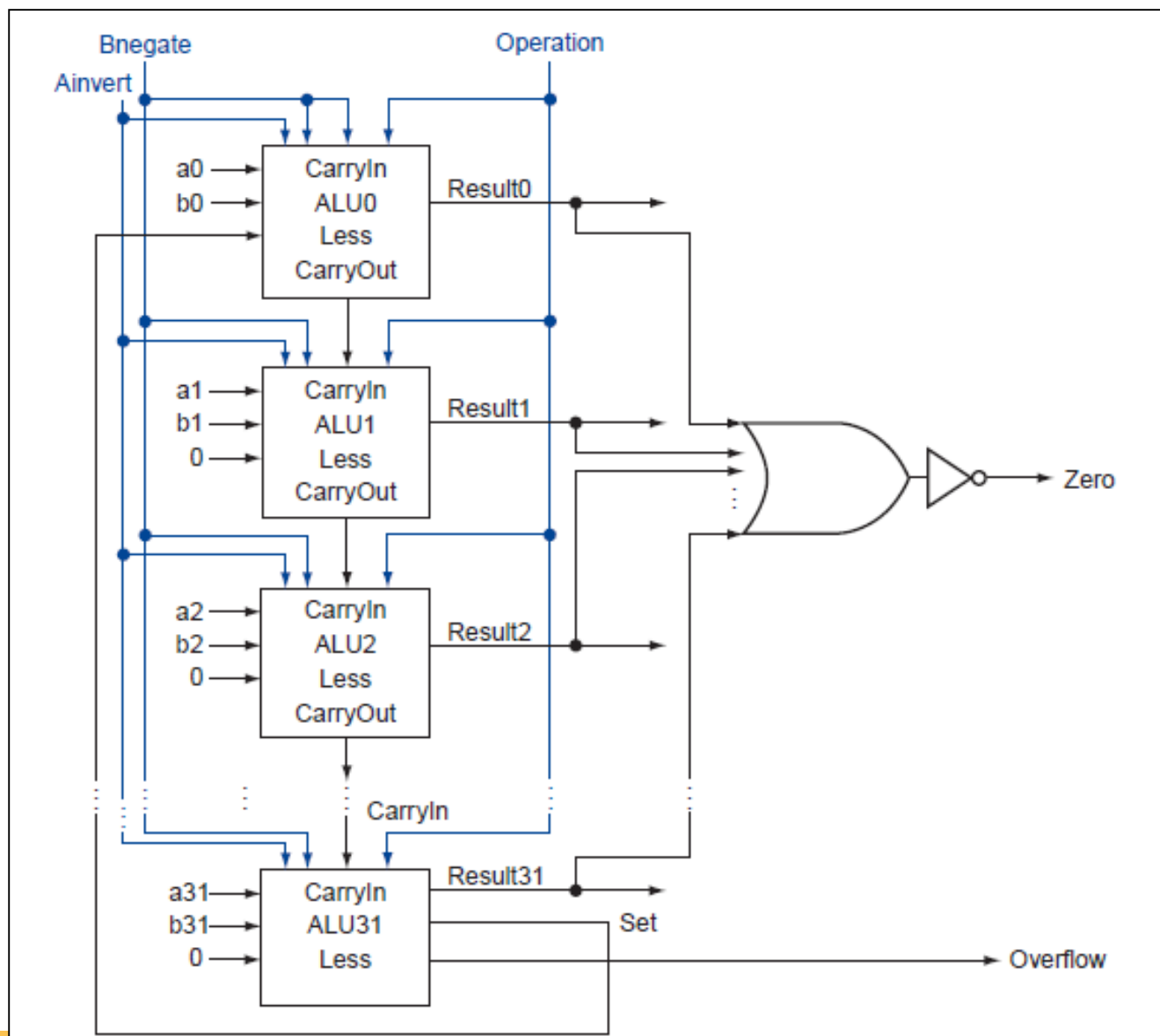
- Ainvert=0
- Binvert=1
- CarryIn=1
- Operation=2

➤ . . .



1. 基本组合逻辑电路：ALU

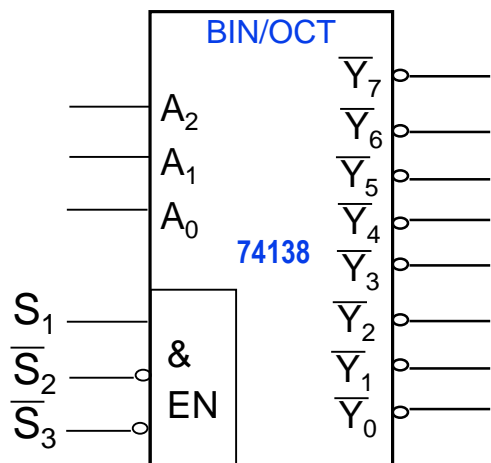
32位ALU



2. 基本组合逻辑电路：译码器

❖ 3线-8线译码器 (74138)

- 3个输入：A2, A1, A0；000~111共8种输入组合。
- 8个输出：Y7~Y0，**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，Y0输出有效；当输入为001时，Y1输出有效。
- 3个使能控制：S0, S1, S2 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。



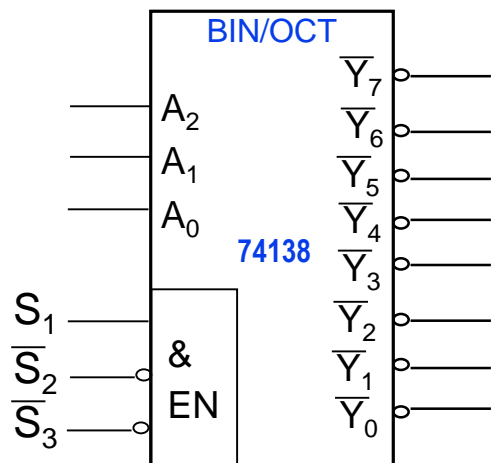
功能表

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

2. 基本组合逻辑电路：译码器

❖ 3线-8线译码器 (74138)

74138真值表



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

根据真值表推导出表达式 (最大项推导法)

$$\overline{Y_0} = A_2 + A_1 + A_0 = \overline{\overline{A_2} \overline{A_1} \overline{A_0}}$$

$$\overline{Y_1} = A_2 + A_1 + \overline{A_0} = \overline{\overline{A_2} \overline{A_1} A_0}$$

$$\overline{Y_2} = A_2 + \overline{A_1} + A_0 = \overline{\overline{A_2} A_1 \overline{A_0}}$$

$$\overline{Y_3} = A_2 + \overline{A_1} + \overline{A_0} = \overline{\overline{A_2} A_1 A_0}$$

$$\overline{Y_4} = \overline{A_2} + A_1 + A_0 = \overline{\overline{A_2} \overline{A_1} \overline{A_0}}$$

$$\overline{Y_5} = \overline{A_2} + A_1 + \overline{A_0} = \overline{\overline{A_2} \overline{A_1} A_0}$$

$$\overline{Y_6} = \overline{A_2} + \overline{A_1} + A_0 = \overline{\overline{A_2} A_1 \overline{A_0}}$$

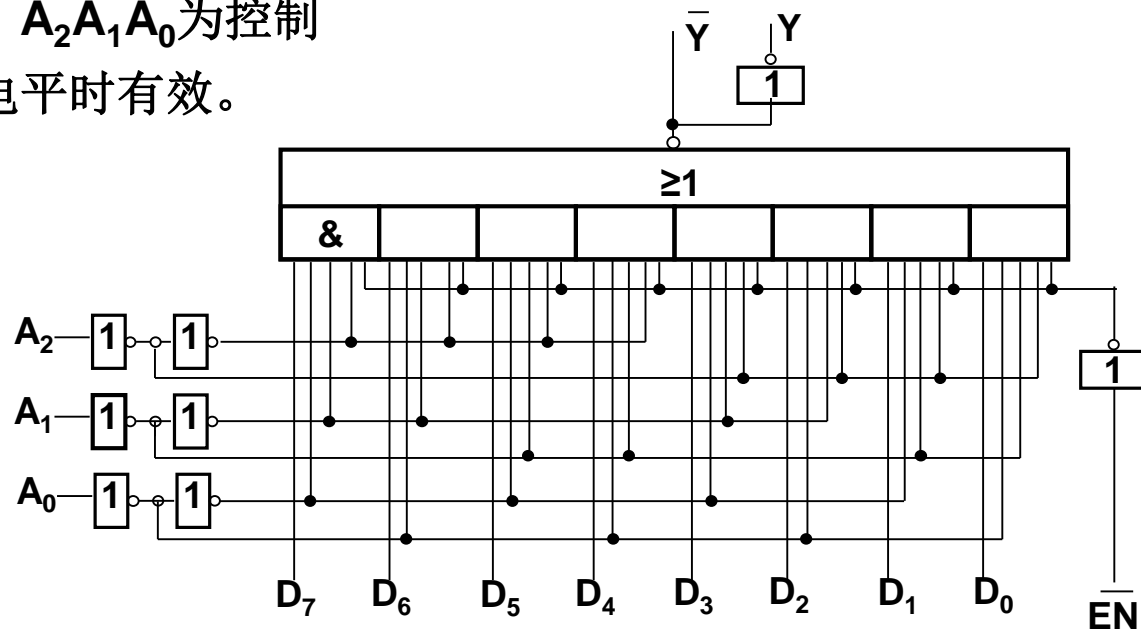
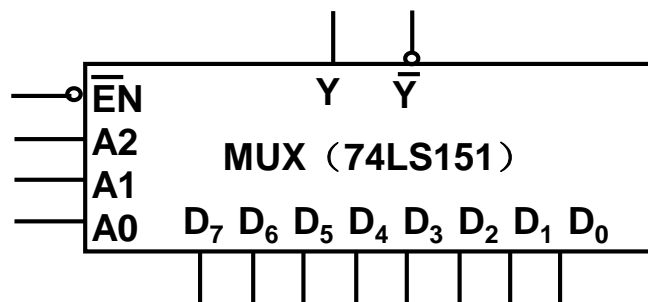
$$\overline{Y_7} = \overline{A_2} + \overline{A_1} + \overline{A_0} = \overline{\overline{A_2} A_1 A_0}$$

3. 基本组合逻辑电路：多路选择器

❖ 8选1多路选择器 (74151)

功能一： 8选1数据选择器， $A_2A_1A_0$ 为控制端，使能控制输入 \overline{EN} 为低电平时有效。

逻辑图与逻辑符号



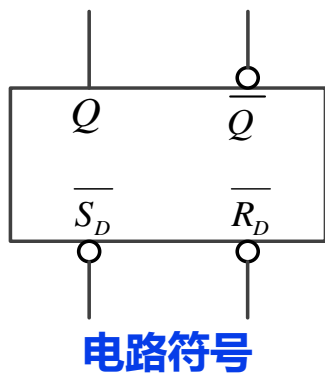
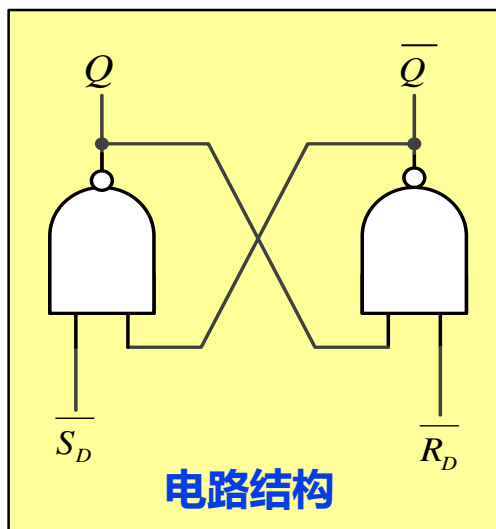
$$Y = \overline{A_2}\overline{A_1}\overline{A_0}D_0 + \overline{A_2}\overline{A_1}A_0D_1 + \overline{A_2}A_1\overline{A_0}D_2 + \overline{A_2}A_1A_0D_3 \\ + A_2\overline{A_1}\overline{A_0}D_4 + A_2\overline{A_1}A_0D_5 + A_2A_1\overline{A_0}D_6 + A_2A_1A_0D_7$$

4 基本RS锁存器

❖ **基本RS锁存器**：具有两个稳定状态，可自行保持输出状态，是各种触发器的基本构成。

➤ RS : Reset/Set

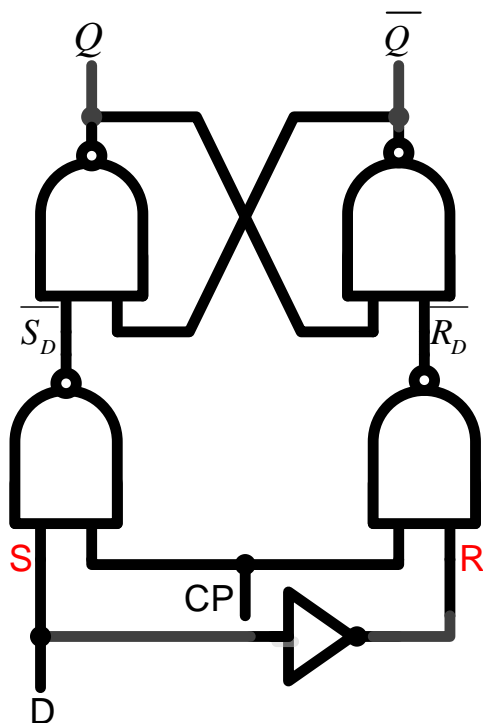
真值表(特性表)



$\overline{S_D}$	$\overline{R_D}$	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		

$$Q^{n+1} = S_D + \overline{R_D} Q^n$$

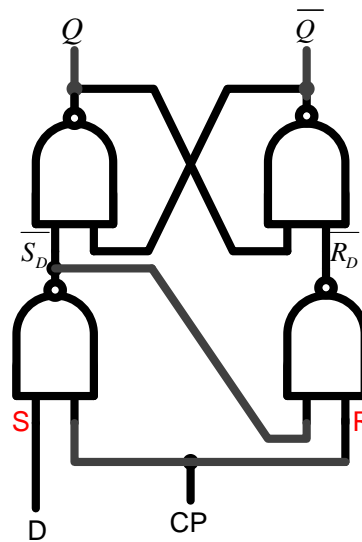
5. 钟控D锁存器



- 当CP = 0，保持原态。
- 当CP = 1时：
 - 若D=0，则S=0，R=1，锁存器置“0”
 - 若D=1，则S=1，R=0，锁存器置“1”

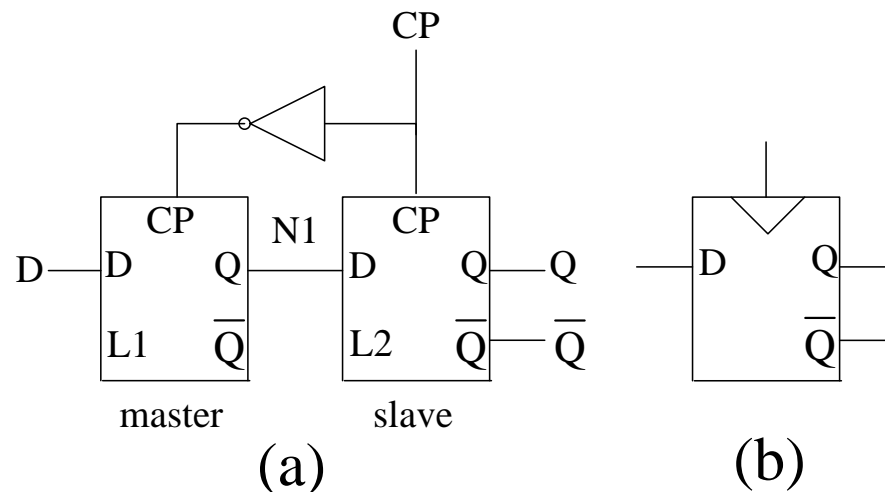
$$Q^{n+1} = S + \overline{R}Q^n = D + DQ^n = D$$

借用输入控制电路的一个与非门对D反相后反馈到另一个与非门的输入端，作为R信号



6. D触发器

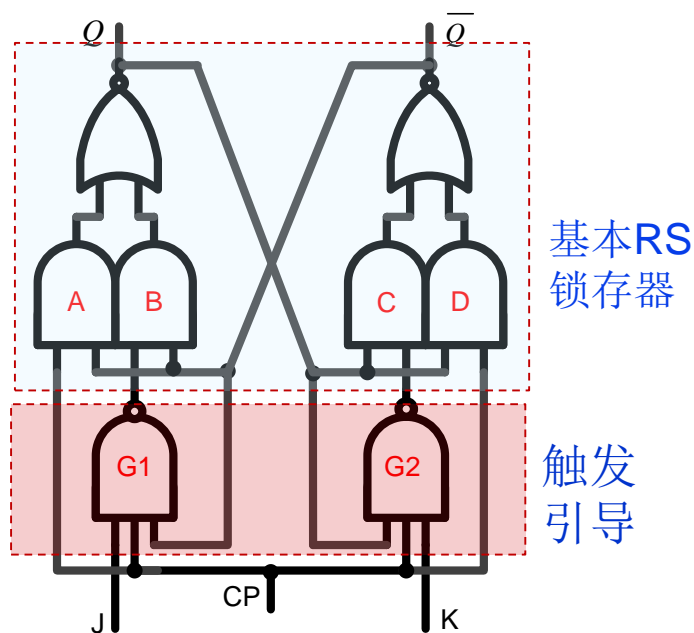
- ❖ 一个D触发器可以由两个反相的D锁存器构成，如下图(a)所示。图(b)为D触发器符号。
- ❖ 主从触发器：锁存器L1为主锁存器，L2为从锁存器
- ❖ 工作原理（CP从0到1）
 - ① CP=0: L1是通路， $Q1 \leftarrow D$ ；L2是断路，Q2值不变
 - ② CP从0上升到1: $Q2 \leftarrow Q1$ ，触发时刻
 - ③ CP=1: L1是断路，Q1值不变；L2是通路， $Q2 \leftarrow Q1$ （实际上Q2也保持不变）



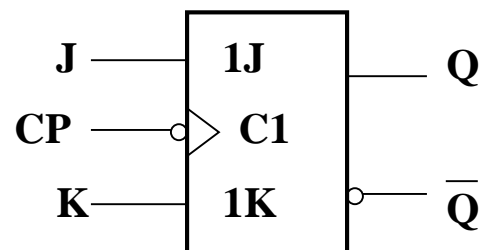
7. 负边沿触发的JK触发器

❖ 电路结构

- 一个基本RS锁存器
- 一个触发引导逻辑
- 利用门电路传输延迟差异而引导触发
- G1、G2传输延迟大于A、D的翻转时间



电路结构



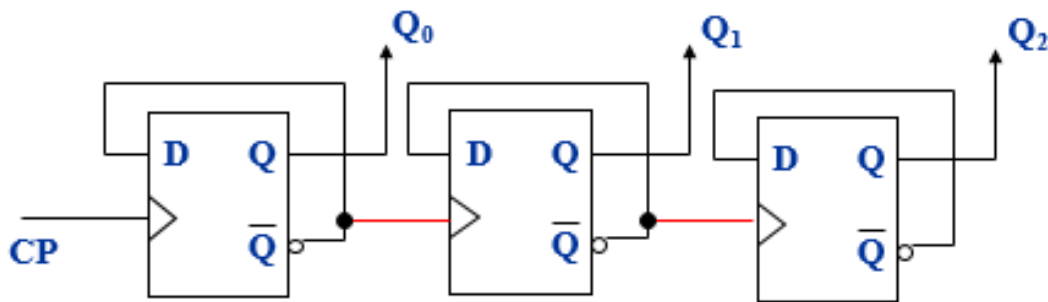
符号表示

$$CP \downarrow \text{ 时}$$
$$Q^{n+1} = J\bar{Q}^n + \bar{K}Q^n$$

时序电路的表示

示例：D触发器（上跳沿触发）构成的异步二进制（模8）加法计数器

（1）逻辑电路图



（2）状态方程

$$Q_0^{n+1} = \overline{Q_0}^n \cdot CP \uparrow$$

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

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

（3）驱动方程

$$D_0 = \overline{Q_0}^n \cdot CP \uparrow$$

$$D_1 = \overline{Q_1}^n \cdot Q_0 \downarrow$$

$$D_2 = \overline{Q_2}^n \cdot Q_1 \downarrow$$

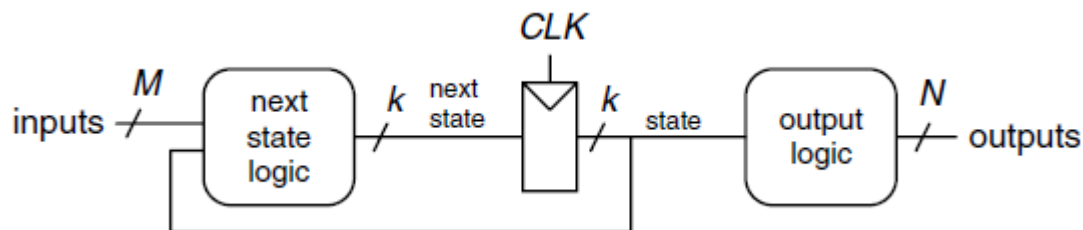
（4）状态转换表

（5）状态转换图

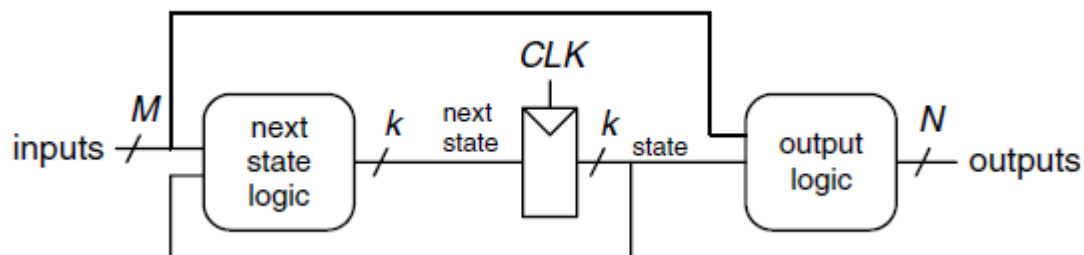
有限状态机模型

❖ **有限状态机 (Finite State Machine, FSM)**：描述有限个状态以及这些状态之间的转移及引起转移的动作等的离散数学模型。

- **次态逻辑**：组合逻辑，根据当前状态和输入计算下一状态；
- **状态寄存器**：在时钟沿到来之前保持现态，并为输出逻辑和次态逻辑提供稳定输入；在时钟沿到来时，锁存次态逻辑输出的状态值。
- **输出逻辑**：组合逻辑，根据现态形成输出信号。



Moore型有限状态机：输出信号仅与当前状态有关



Mealy型有限状态机：输出信号与当前状态及输入信号有关

7. Mealy型FSM设计

【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：

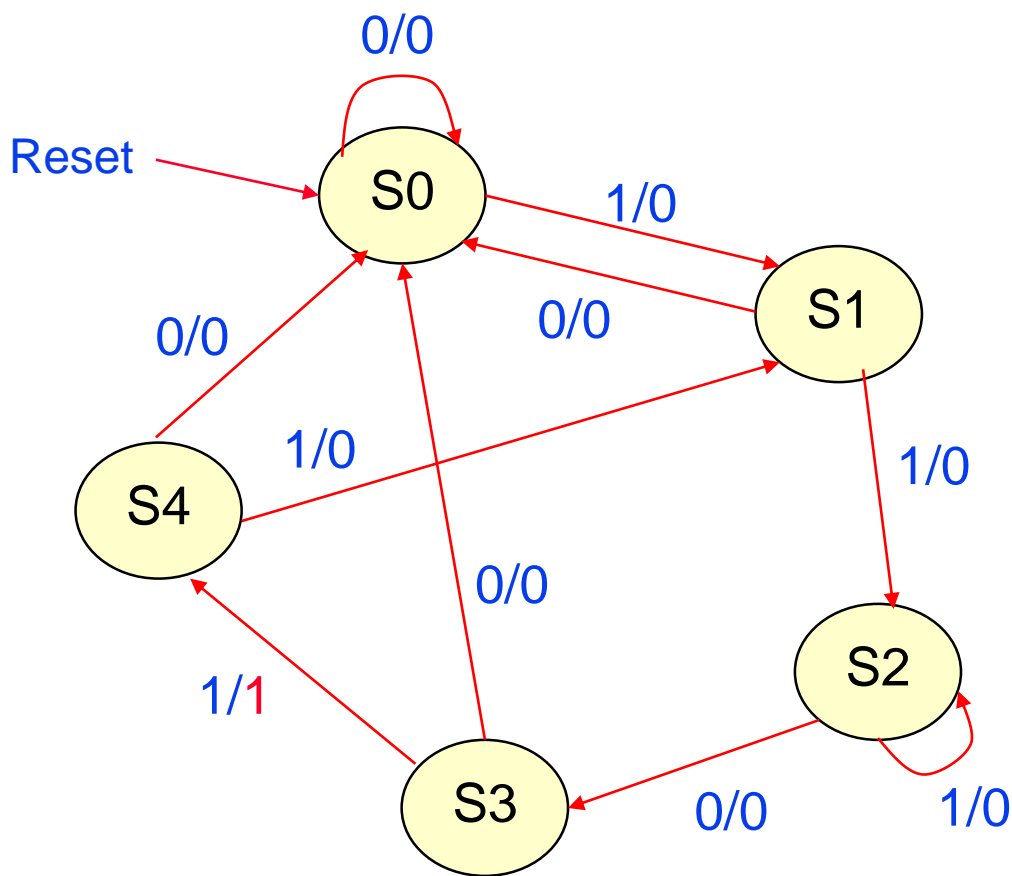
(1) 检测器FSM模型

- 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

6. Mealy型FSM设计

解（续1）：

（2）画出状态转换图

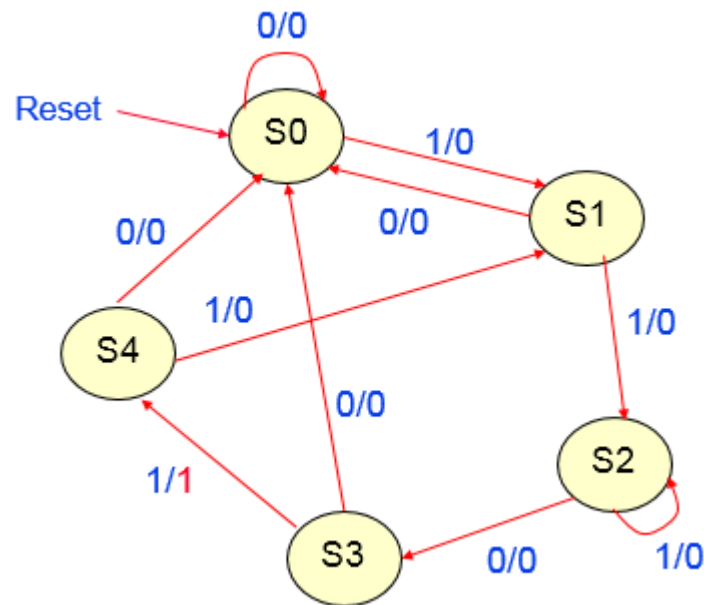


6. Mealy型FSM设计

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



6. 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。

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

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

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

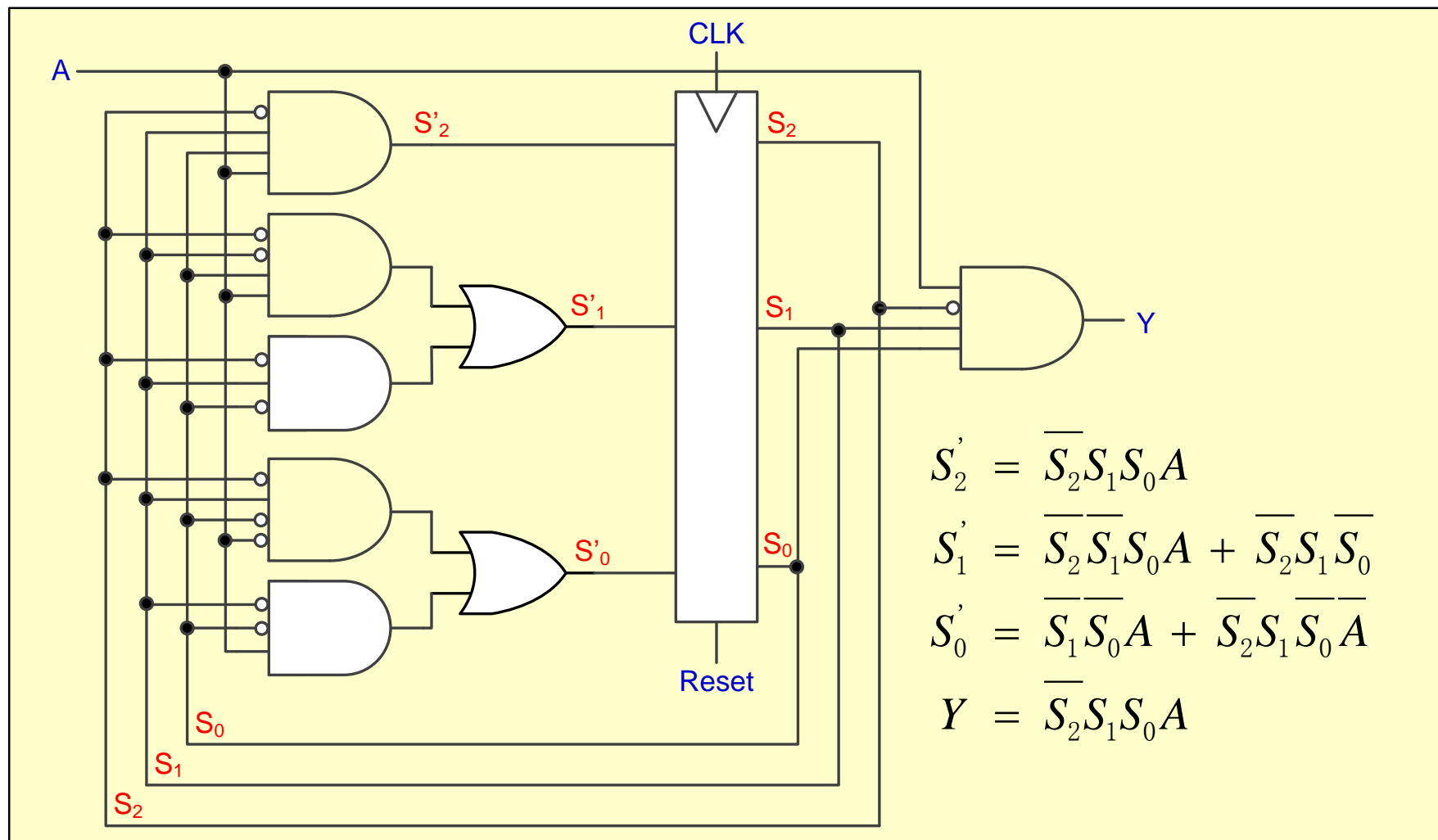
$$Y = \overline{S_2} S_1 S_0 A$$

当前状态 ($S_2 S_1 S_0$)	输入 (A)	下一状态 ($S_2' S_1' S_0'$)	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

6. Mealy型FSM设计

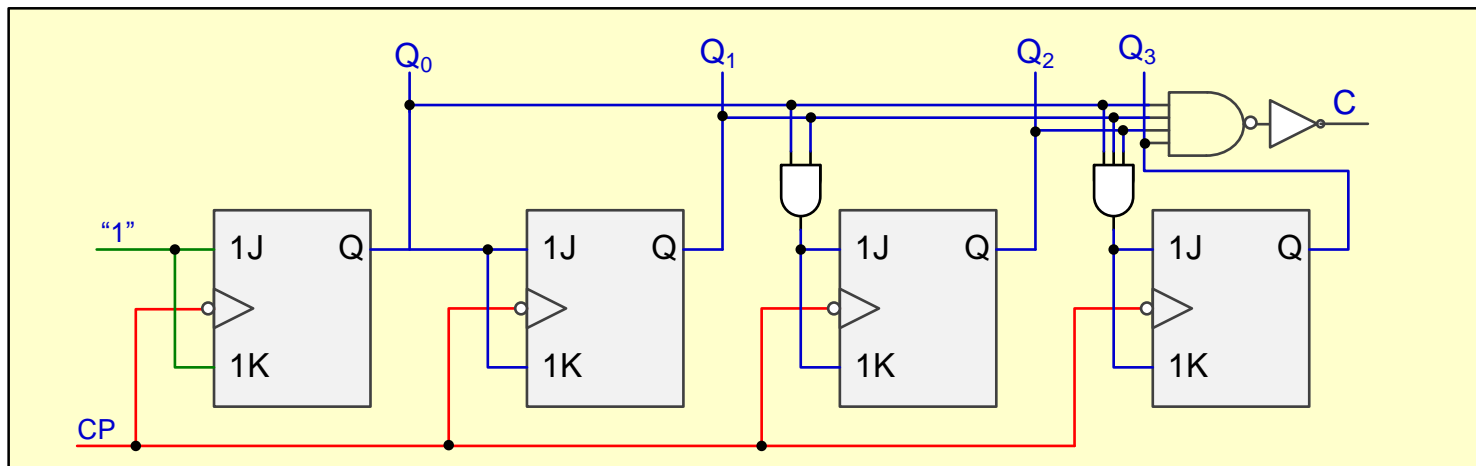
解（续4）：

（4）根据逻辑表达式画出逻辑图。

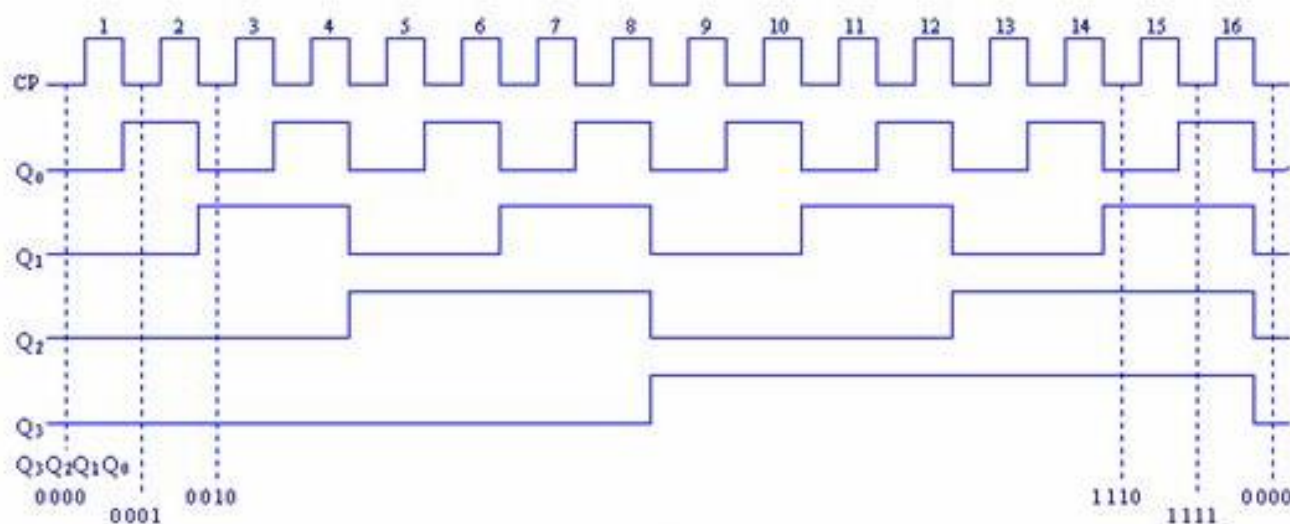


7. 同步计数器

► 同步十六进制数加法计数器



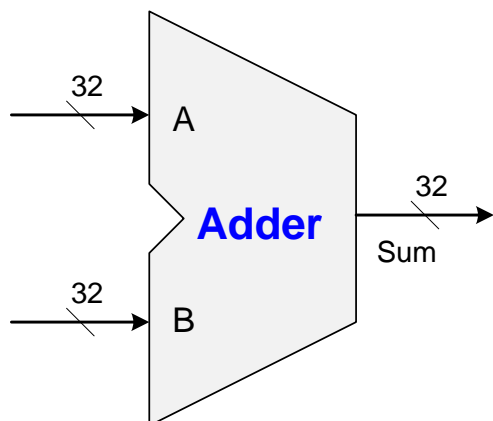
Q_0 、 Q_1 、 Q_2 、 Q_3 的周期分别是CP 周期的2、4、8、16倍，实际上是分别对CP进行了2、4、8、16分频，因此二进制计数器也称为分频器。



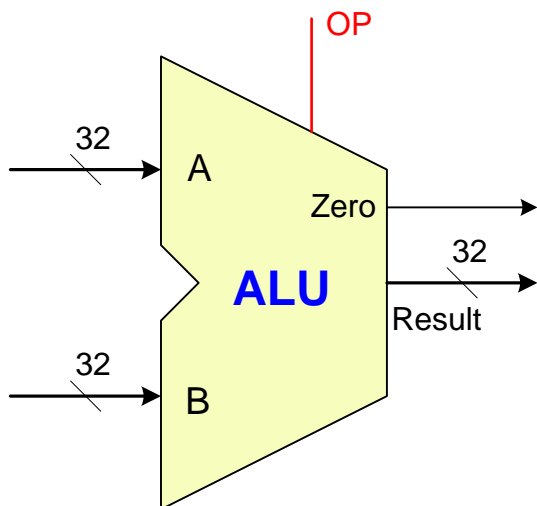
MIPS数据通路的组合逻辑部件

❖ 组合部件：

➤ 加法器 (Adder) 、 算术逻辑运算单元 (ALU)



输入 (32位)		输出 (32位)
A	B	Sum=A+B

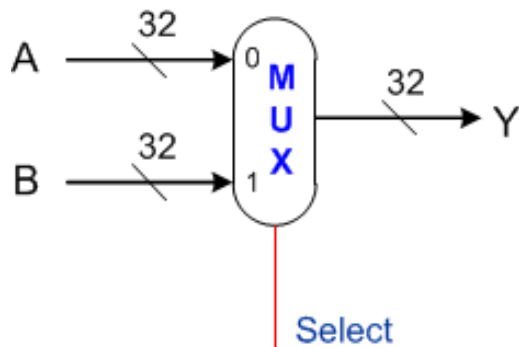


数据输入 (32位)		控制输入OP (4位)	输出
A	B	0000	Result = A&B
A	B	0001	Result = A B
A	B	0010	Result = A+B
A	B	0110	Result = A-B
A	B	0111	If A=B then Result=1

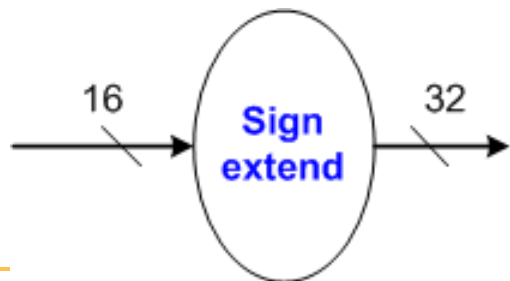
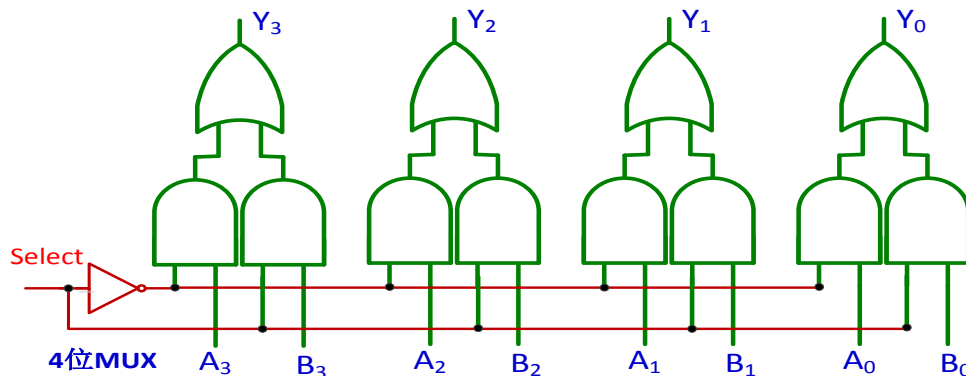
MIPS数据通路的组合逻辑部件

❖ 组合部件

- 多路选择器 (Mux)、符号扩展器 (Signext)
- 多路选择器有二选一，三选一，四选一等。



输入 (32位)		Select输入	输出 (32位)
A	B	无效 (低电平)	$Y=A$
A	B	有效 (高电平)	$Y=B$



16位二进制数按符号位扩展至32位

如: $\text{Signext}(8000\text{H}) = \text{FFFF}8000\text{H}$

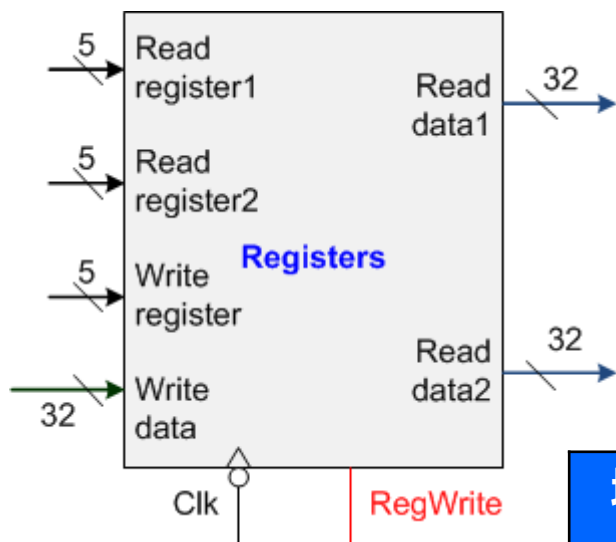
$\text{Signext}(7800\text{H}) = 00007800\text{H}$

MIPS数据通路的寄存器堆

❖ 寄存器堆（32个寄存器）

➤ 两个32位输出端口

➤ 一个32位输入端口



寄存器堆读操作（输出）

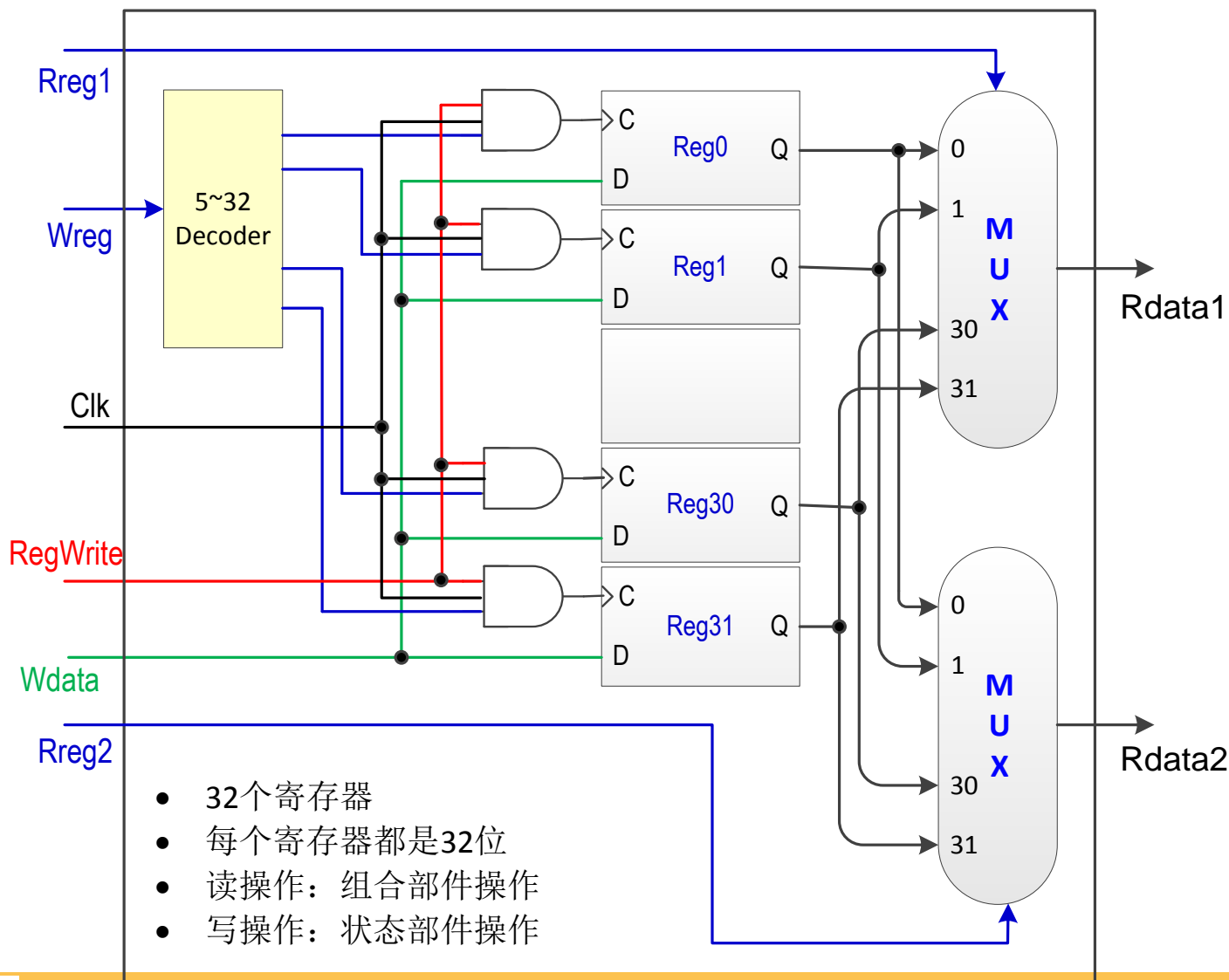
地址输入（5位）		数据输出
Read register1	Read register2	Read data1=R[Readregister1] Read data2=R[Readregister2]

寄存器堆写操作（输入）

地址输入（5位）	数据输入（32位）	控制输入 RegWrite	操作
Write register	Write data	无效	无
Write register	Write data	有效 (Clk时钟下跳沿)	$R[\text{Writeregister}] \leftarrow \text{Write data}$

MIPS数据通路的寄存器堆

❖ 寄存器堆内部结构

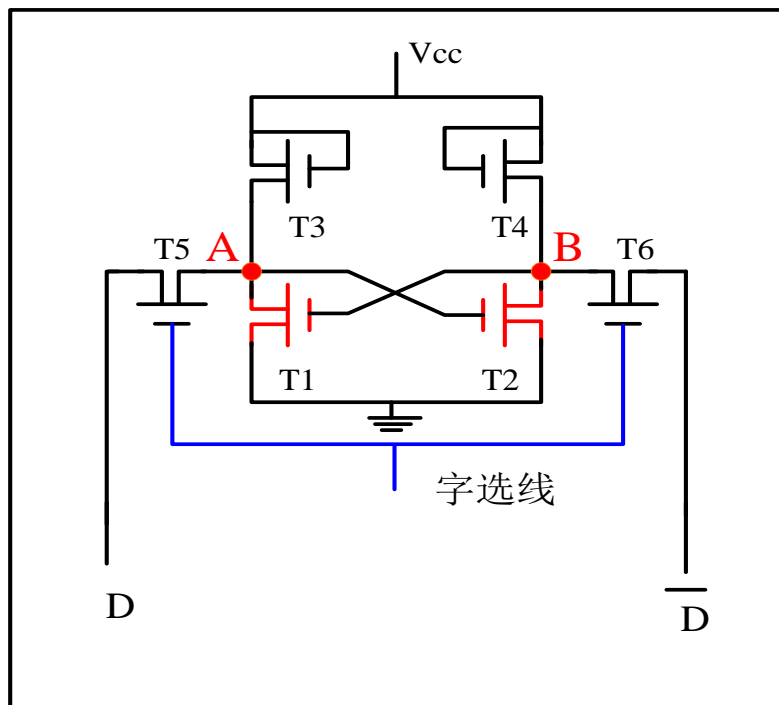


第二部分：存储系统

- ❖ 主存储器
- ❖ 高速缓存
- ❖ 虚拟存储
- ❖ 磁盘与I/O

存储单元电路

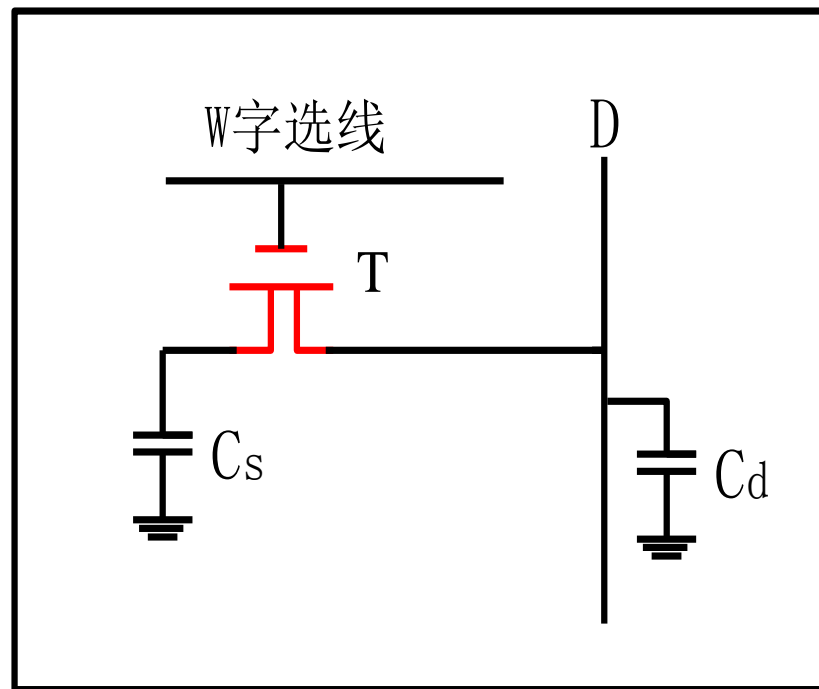
SRAM存储单元电路



SRAM的特点

- 集成度低
- 功耗大

DRAM存储单元电路

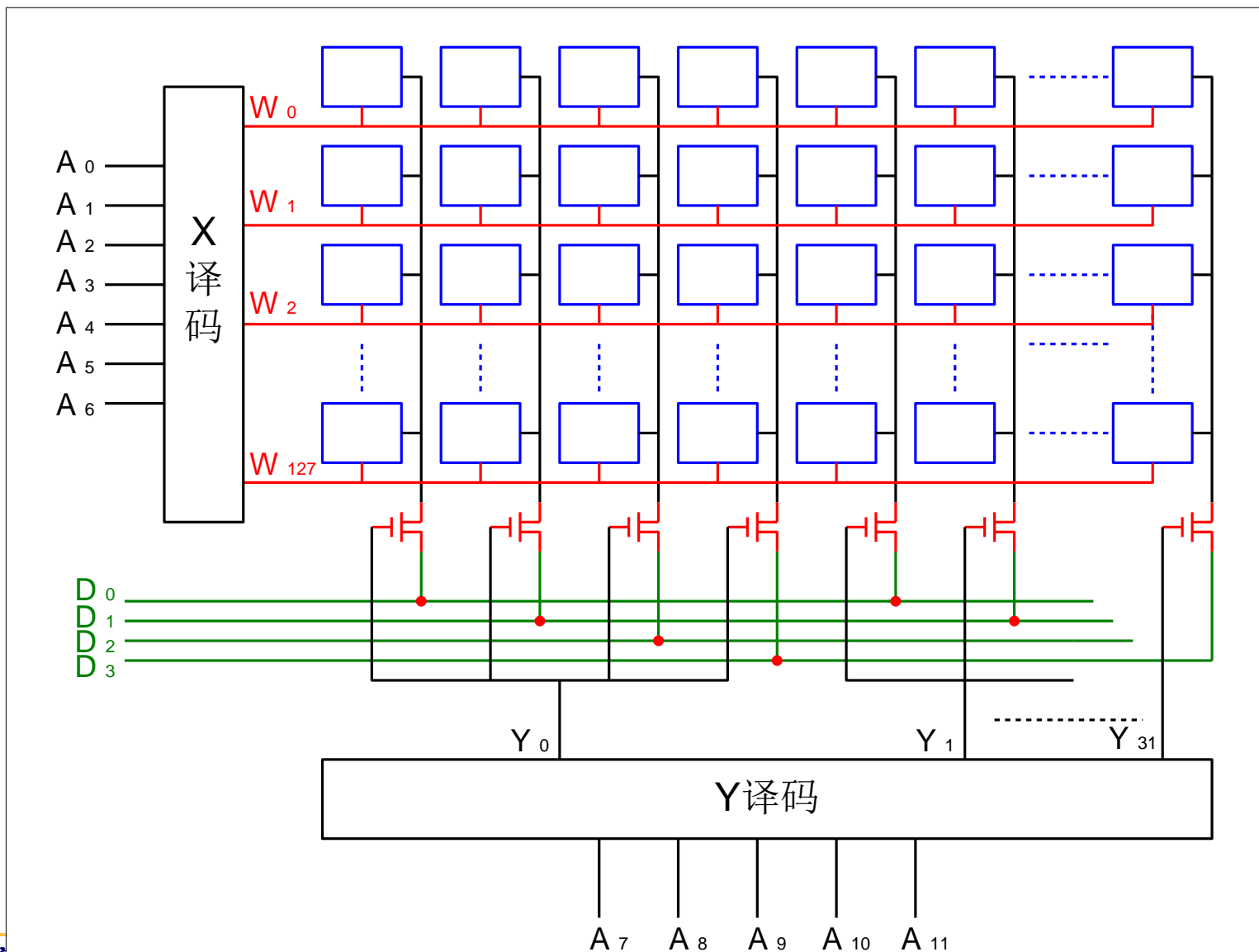


DRAM的特点

- 集成度高
- 功耗小
- 需要刷新

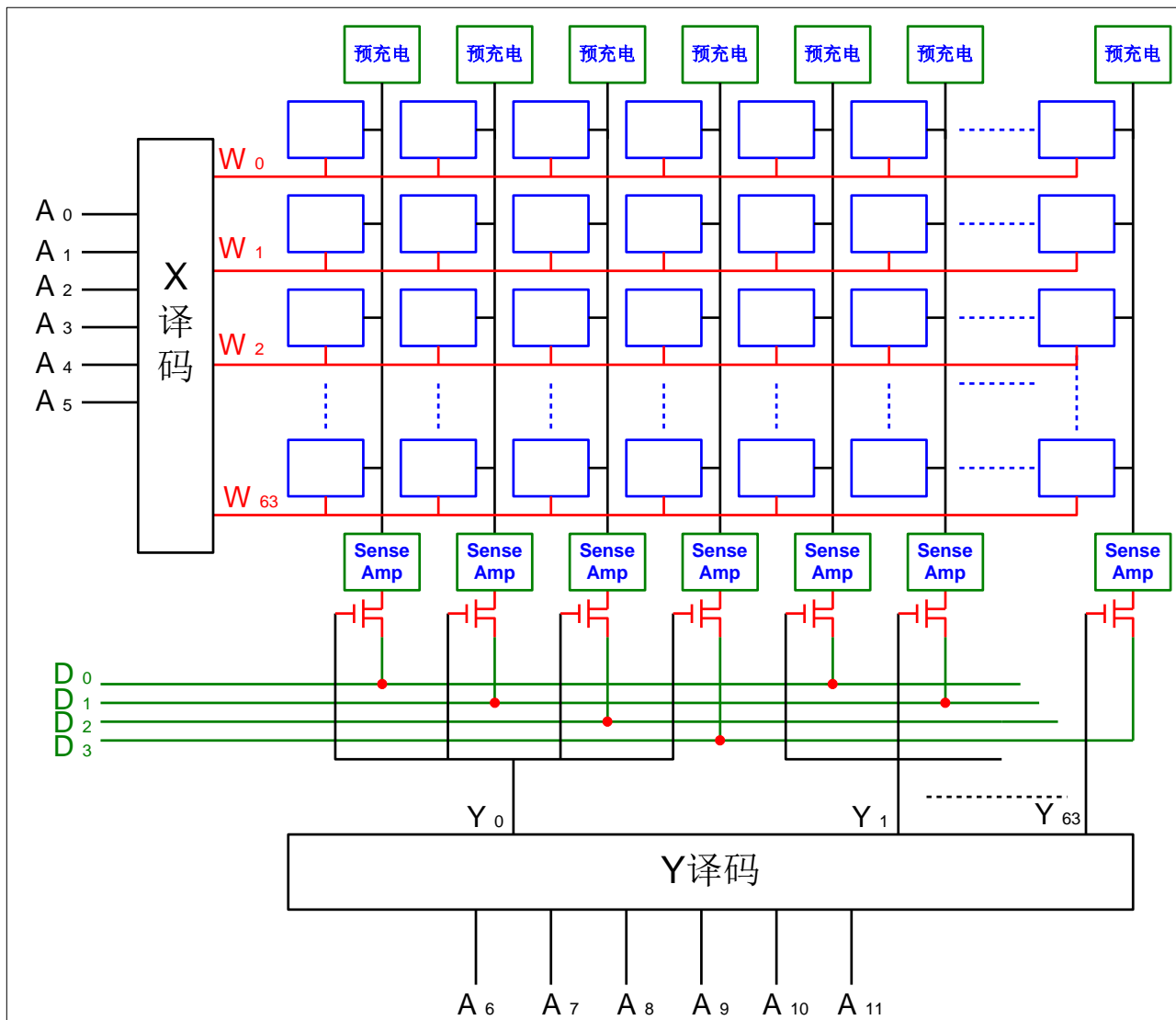
存储芯片内部结构

❖ 二维地址结构（SRAM）：4096*4：4096个字，每个字4位。



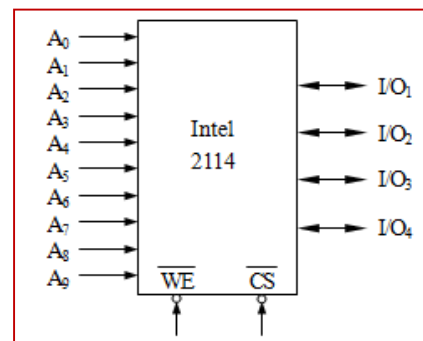
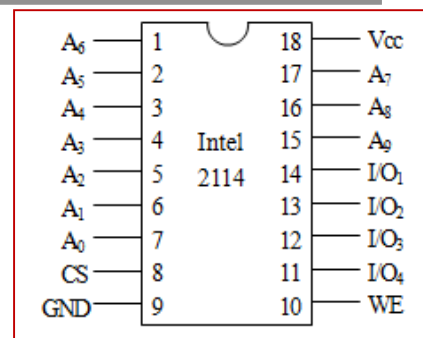
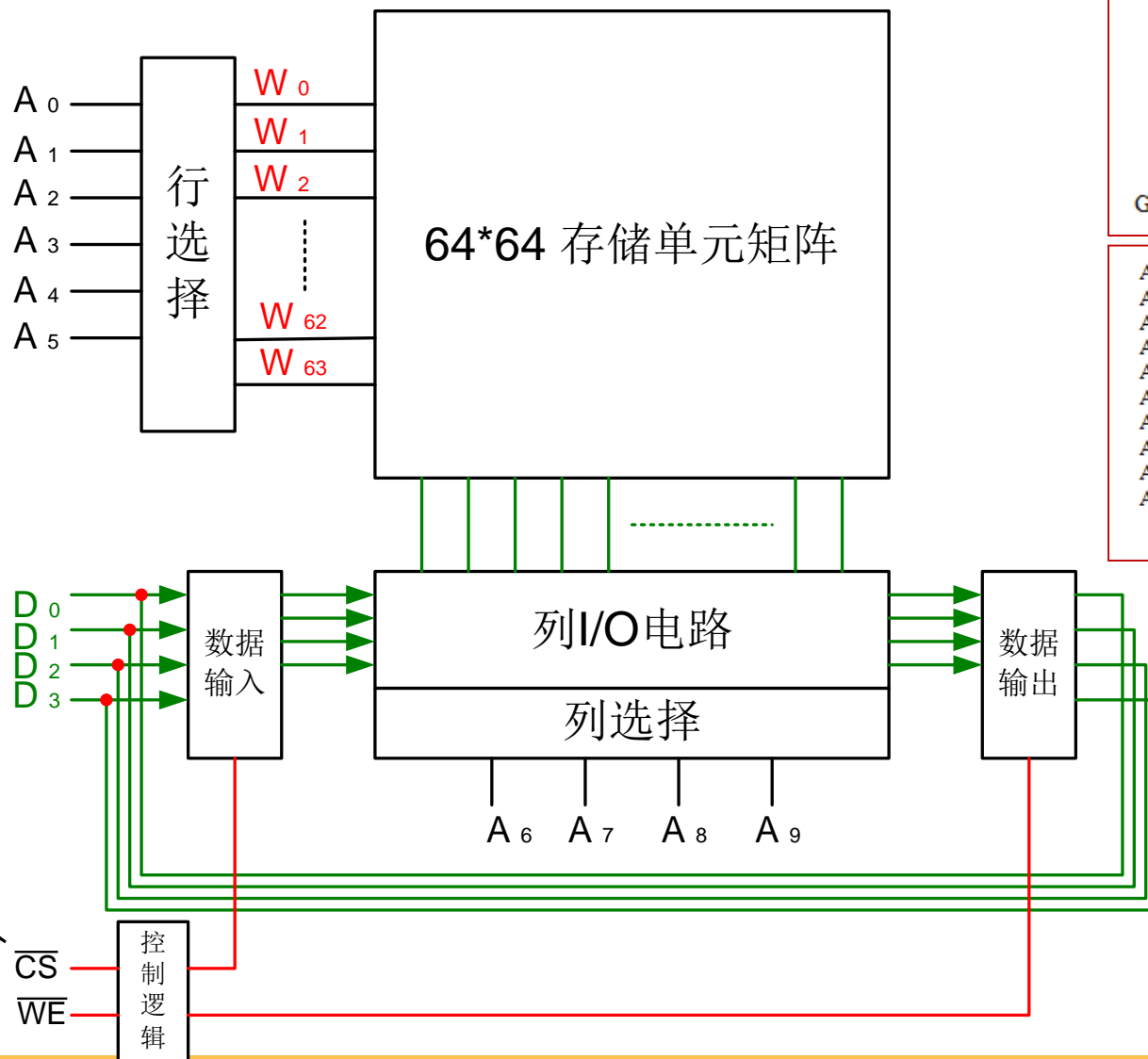
存储芯片内部结构

❖ 二维地址结构（DRAM）：4096*4：4096 个字，每个字 4 位。



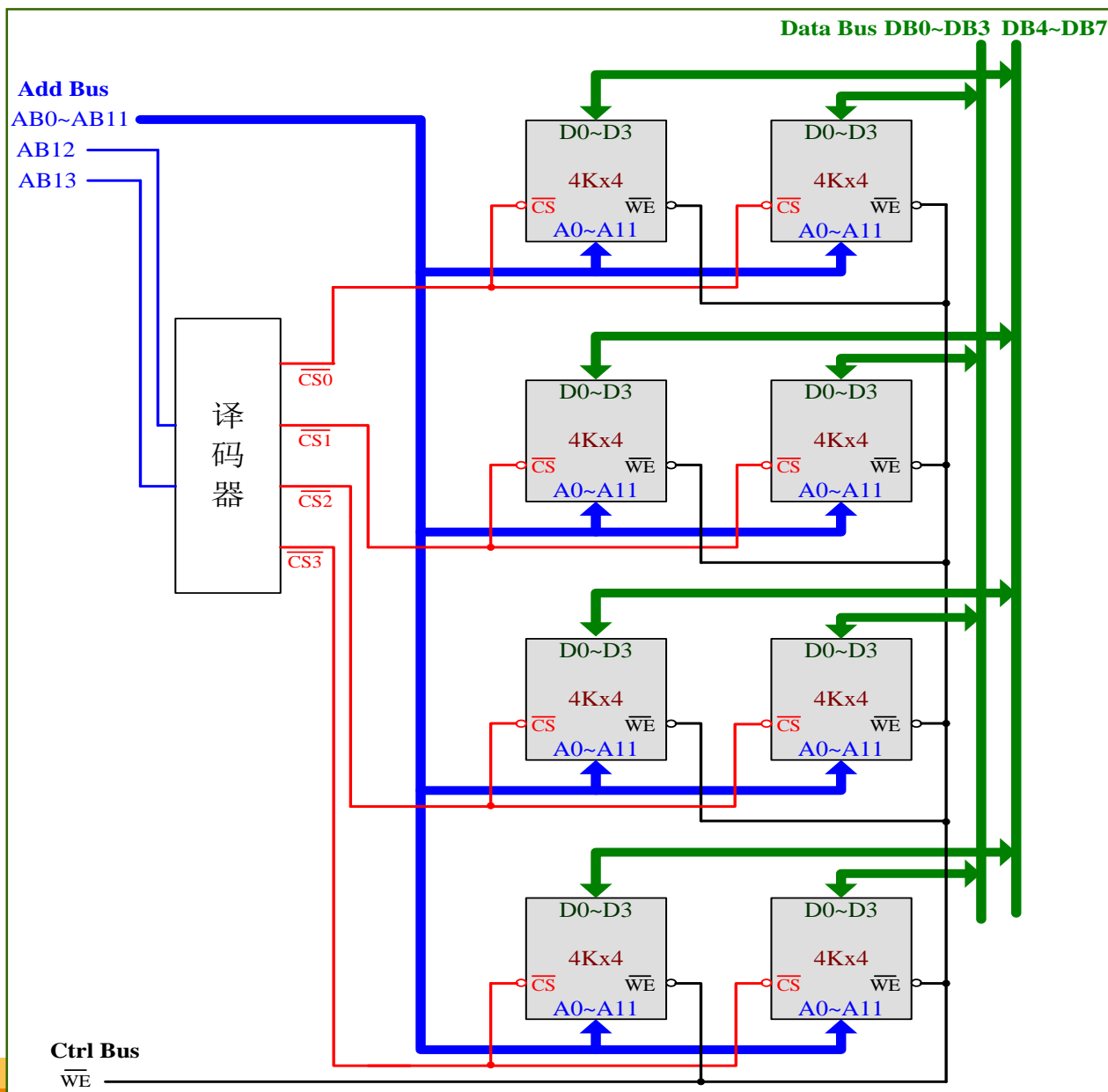
存储芯片结构示例

❖ SRAM 2114(1024×4)芯片结构



存储器芯片的扩展

4Kx4 SRAM存储芯片构成
16Kx8的存储器连接图



CPU与主存的连接（示例）

CPU地址线A15~A0，数据线D7~D0， \overline{WR} 为读/写信号， \overline{MREQ} 为访存请求信号。0000H~3FFFH为系统程序区，4000H~FFFFH为用户程序区。用8K×4位ROM芯片和16K×8位RAM芯片构成该存储器，要求说明地址译码方案，并将ROM芯片、RAM芯片与CPU连接。

解：因为0000H~3FFFH为系统程序区，ROM区高两位总是00，低14位为全译码。

ROM区大小为: $2^{14} \times 8\text{位} = 16\text{K} \times 8\text{位} = 16\text{KB}$

ROM芯片数为: $16K \times 8\text{位} / 8K \times 4\text{位} = 2 \times 2 = 8$, 字方向扩展2倍, 位方向扩展2倍

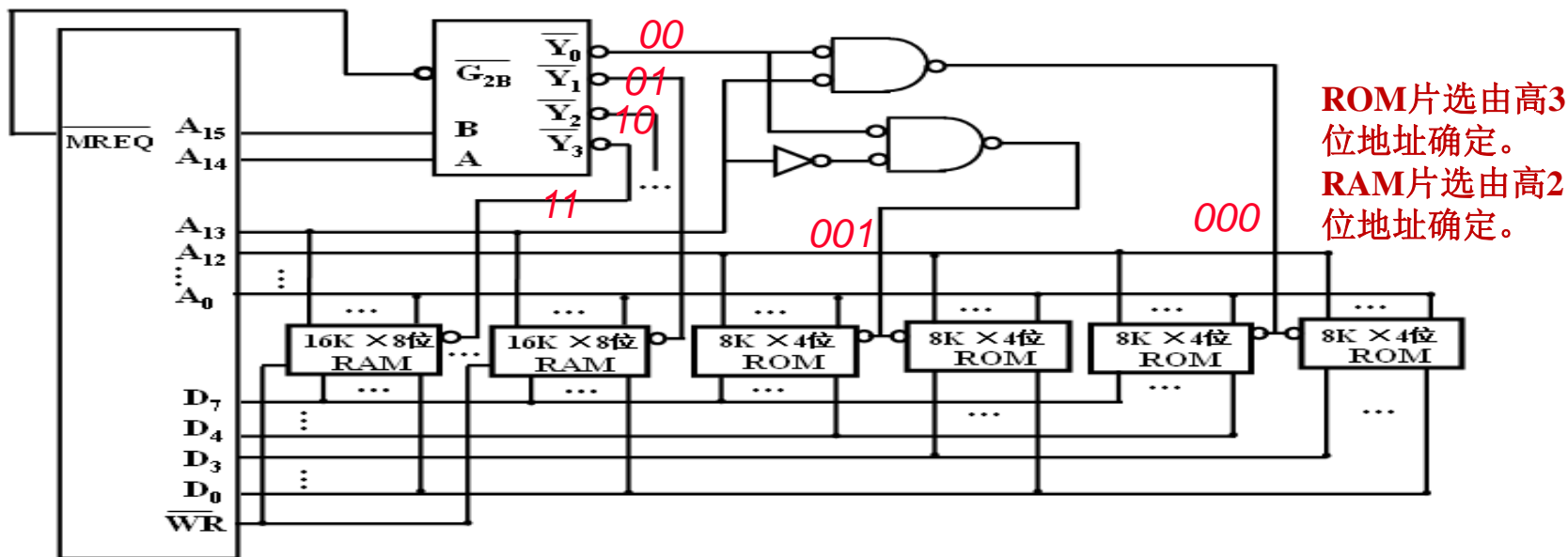
ROM芯片内地址位数为13位，连到CPU低13位地址线A12~A0

因为4000H~FFFFH为用户程序区，RAM区高两位是01、10、11，低14位为全译码。

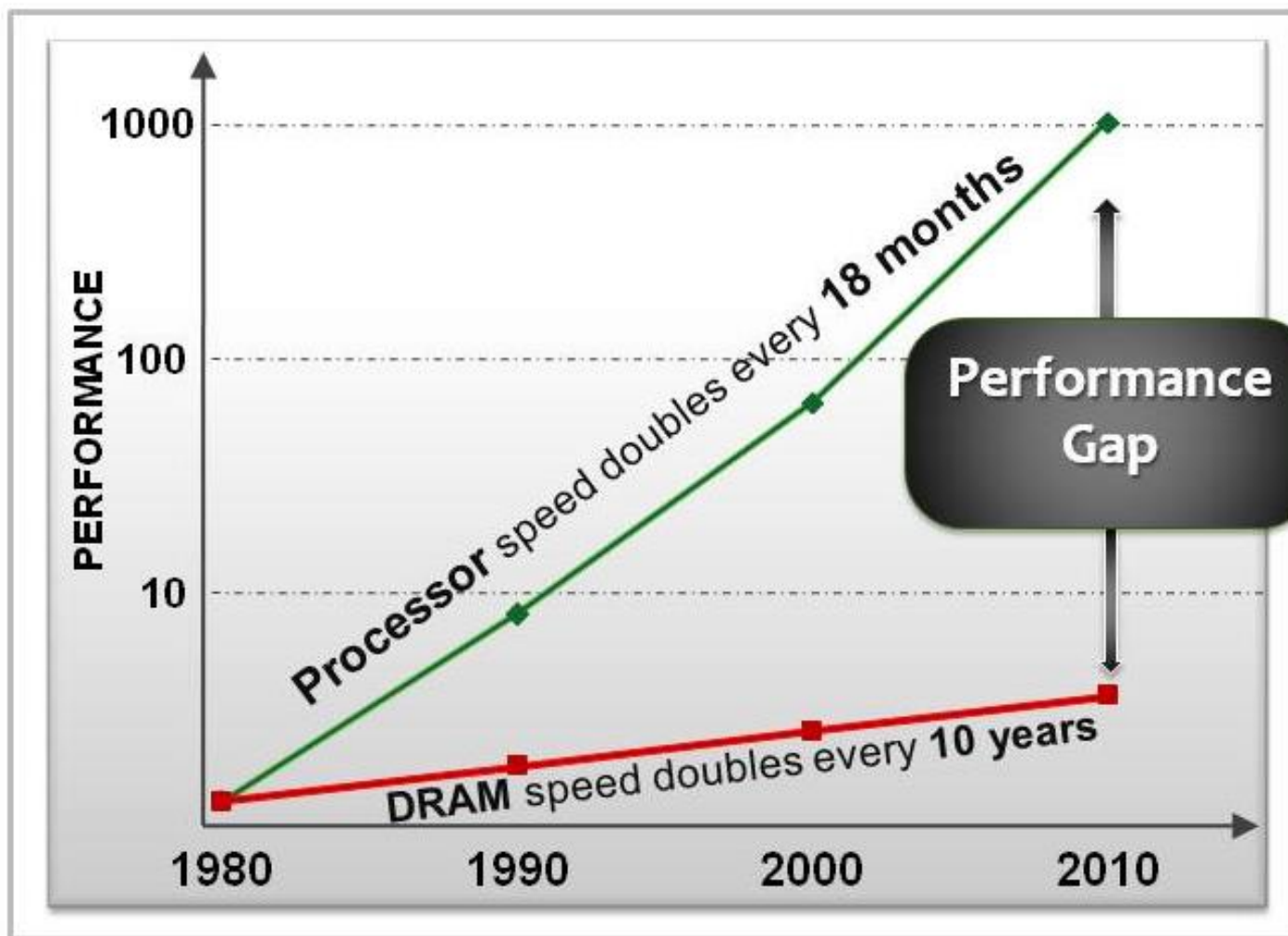
RAM区大小为: $3 \times 2^{14} \times 8\text{位} = 3 \times 16\text{K} \times 8\text{位} = 48\text{KB}$

RAM芯片数为： $48\text{K} \times 8\text{位} / 16\text{K} \times 8\text{位} = 3 \times 1 = 3$ ，字方向上扩展3倍，位方向上不扩展。

RAM芯片内地址位数为14位，连到CPU低14位地址线A13~A0。



处理器—DRAM存储器的性能差距



Source: acm.org & MoSys

©MoSys, Inc. 2010. All Rights Reserved.

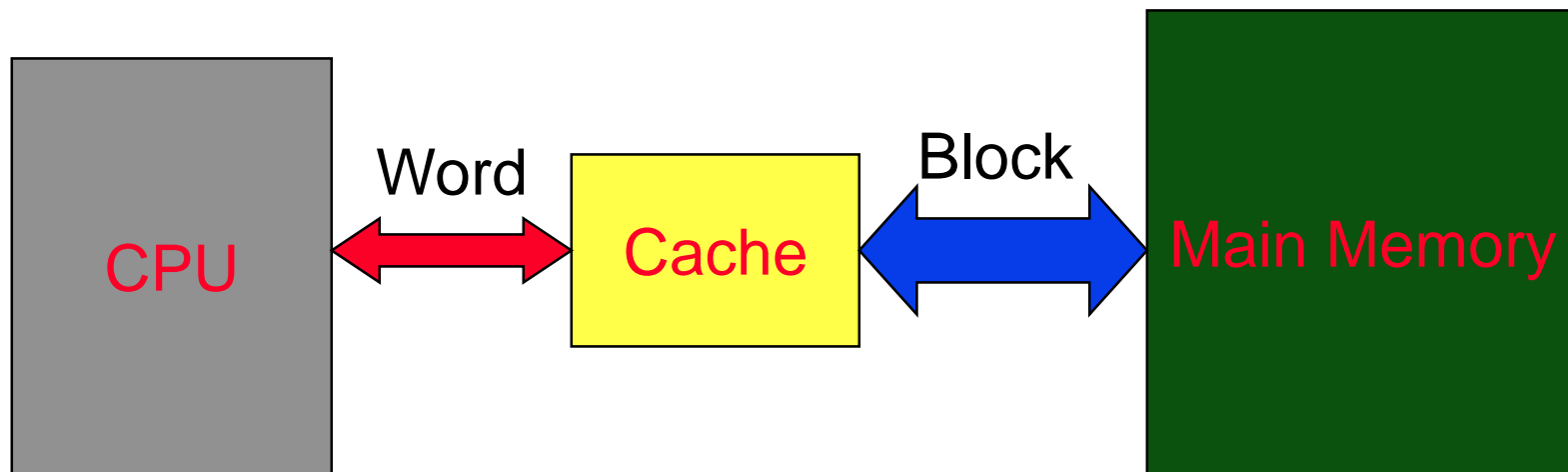
存储访问的局部性原理

- ◆ 局部性原理(**principle of locality**):在访问存储器时, 无论是存取指令或存取数据所访问的存储单元都趋于聚集在一个较小的连续单元区域中。
 - 时间局部性(**temporal locality**): 最近访问的存储单元在不久的将来仍将被访问
 - 空间局部性(**spatial locality**): 下次访问的存储单元很可能就在刚刚访问的存储单元附近

高速缓冲存储器 (CACHE) 的结构

❖ Cache产生的前提

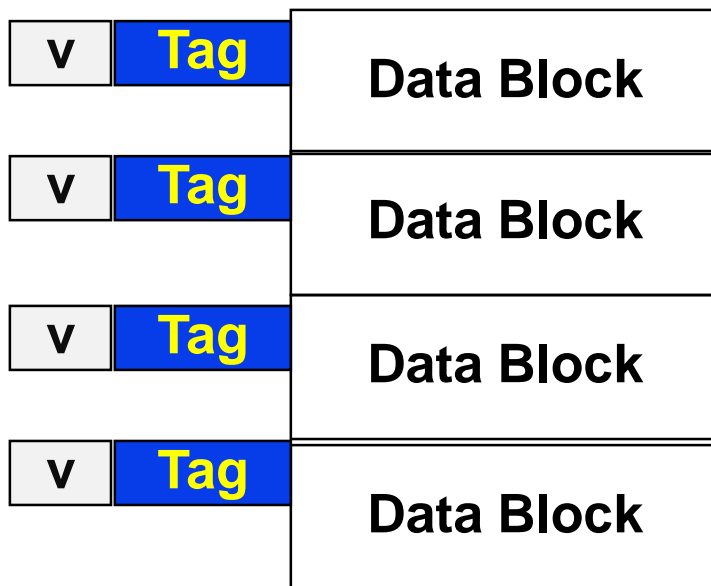
- ▶ 在CPU与内存之间设置一个高速的容量相对小的存储机构，把CPU正在执行的指令或数据附近一部分主存内容取来保存在这个存储机构中。在一段时间内CPU可以减少访问内存的频度，提高运行效率。这个存储机构就是高速缓冲存储器（CACHE）。



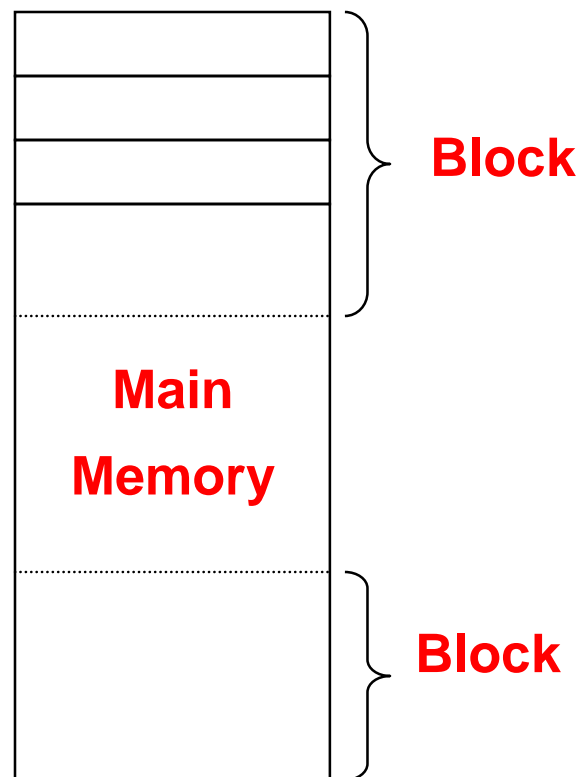
高速缓冲存储器 (Cache) 的原理

❖ 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，地址转换机制，地址标标记（Tag），一个Block具有一个Tag；
- 替换机制：记录Block的使用情况，替换策略，有效位（v）记录对应数据块中的数据是否有效。



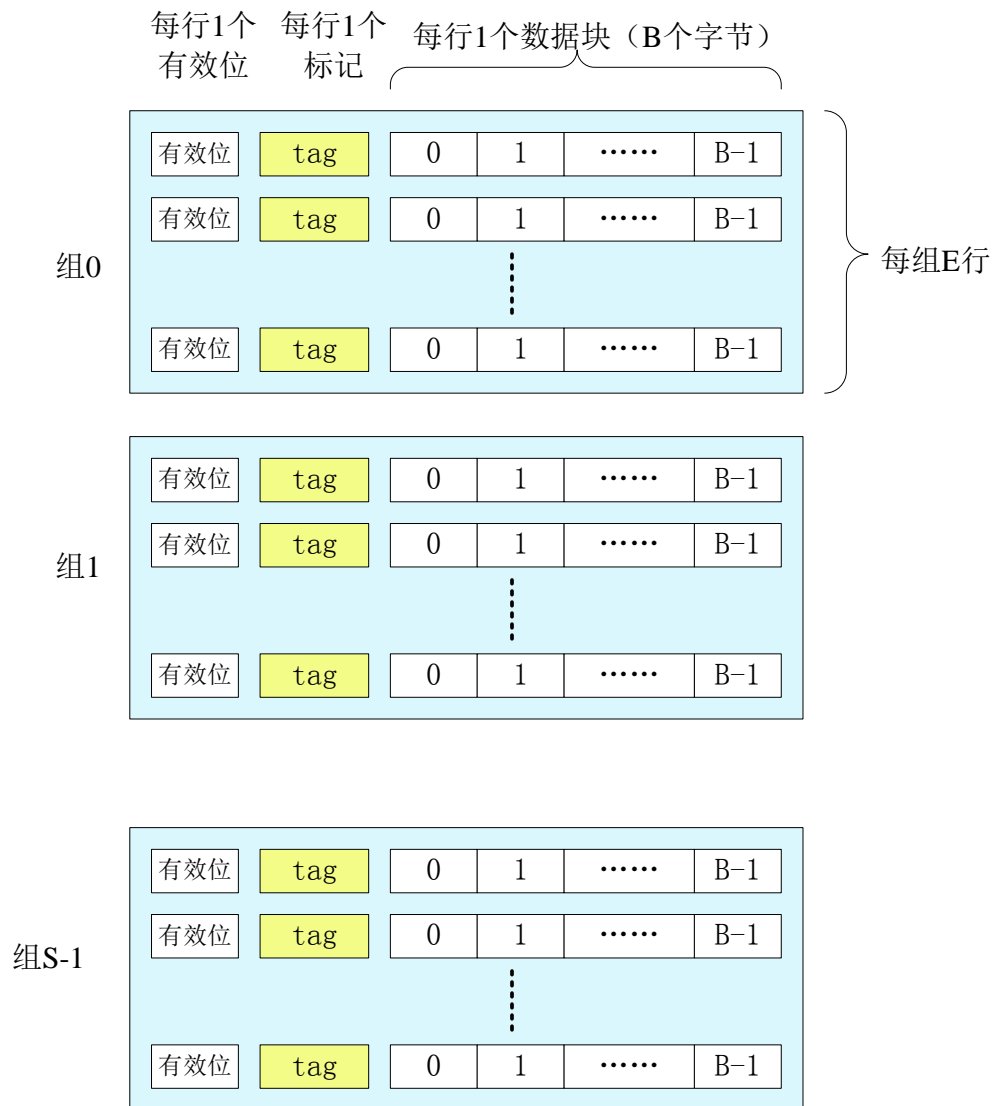
Cache 的基本结构



高速缓冲存储器 (Cache) 的原理

❖ Cache结构示意

- 分S组
- 每组E行
- 每数据块包含B个字节



Cache容量 (字节数) : $B \times E \times S$

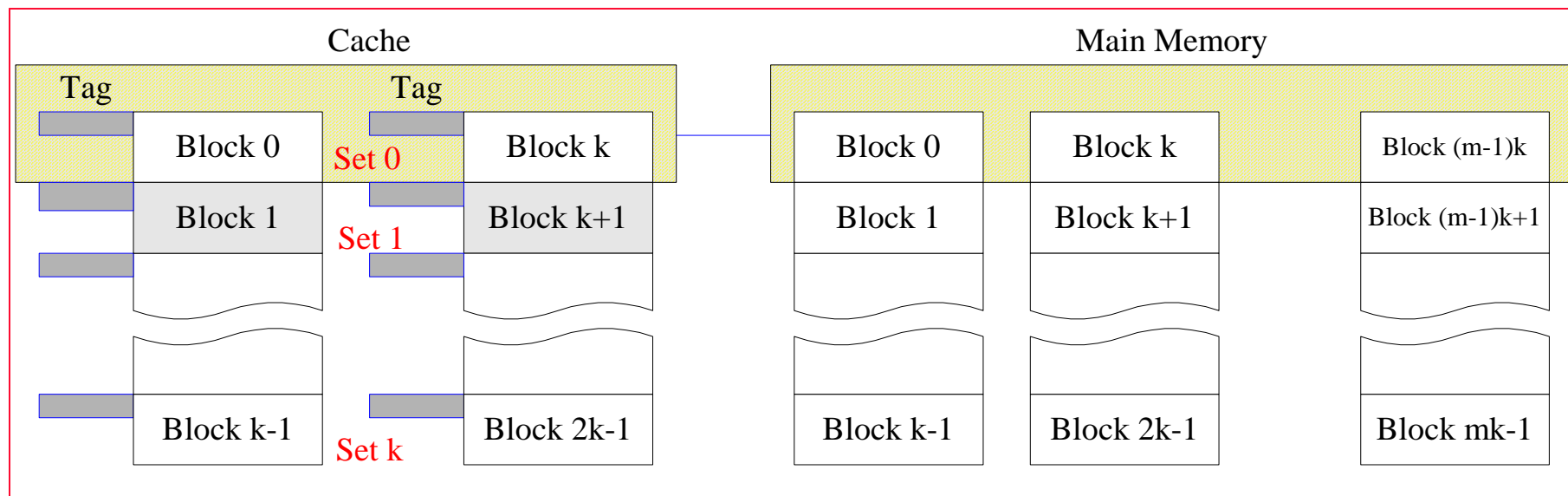
CACHE与主存之间的映射

❖ 组相联映射 (Set Associative Mapping)

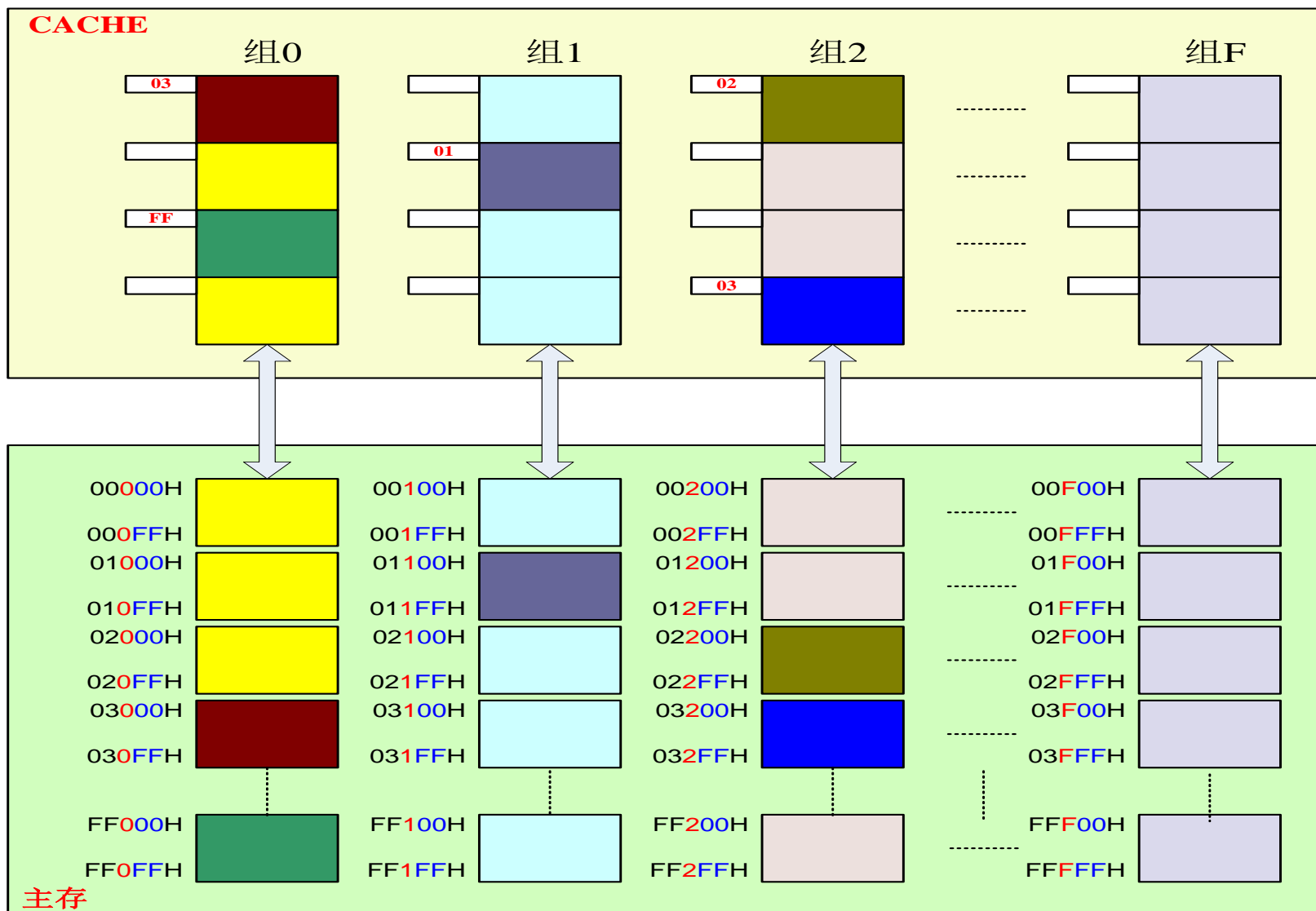
- 映射关系: Cache 分成 **K** 组, 每组分成分 **L** 块; 主存的块 **J** 以下列原则映射到 Cache 的组 **I** 中的任何一块。

$$I = J \bmod K$$

- 实际上主存与Cache都分成 **K** 组, 主存每一组内的块数与Cache一组内的块数不一致, 主存组**M**内的某一块只能映射到Cache组**M**内, 但可以是组**M**内的任意一块。

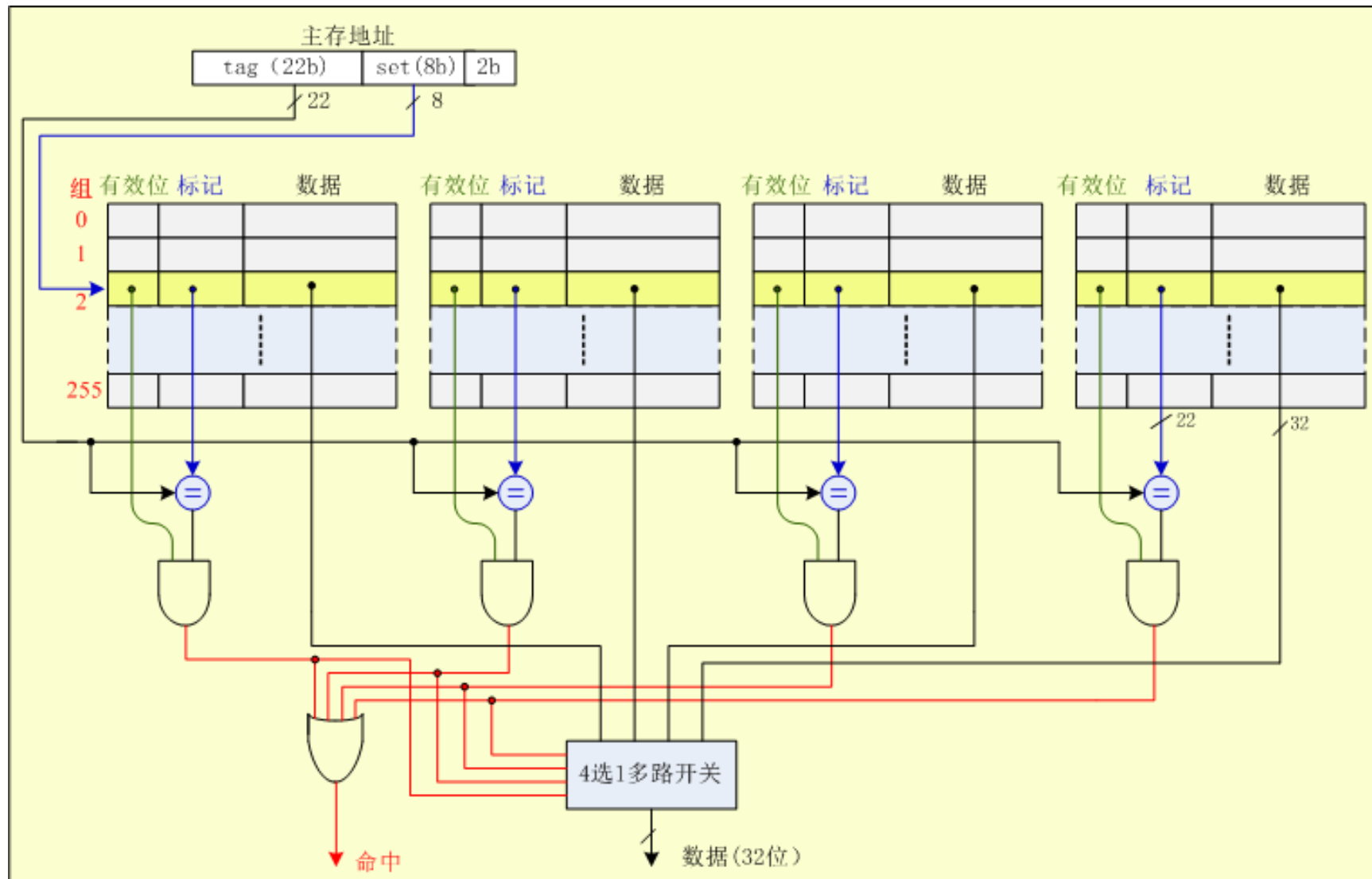


Cache与主存之间的映射—组相联



cache与主存之间的映射—组相联

- Cache示例：Cache容量4KB，4路组相联，数据块大小4B，主存地址32位。



Cache的替换策略

■ 替换策略

- 最近最少使用法（LRU, Least-Recently Used）：记录每一个数据块的相对使用情况，最近没有被使用的块被替换。
- 先进先出法（FIFO, First-In-First-Out）：最先装入数据的块被替换；
- 最小使用频率法（LFU, Least-Frequently Used）：记录每一个数据块的使用频率，使用次数最少的被替换。
- 随机法（RAND, Random）：随机选择一个数据块进行替换。

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

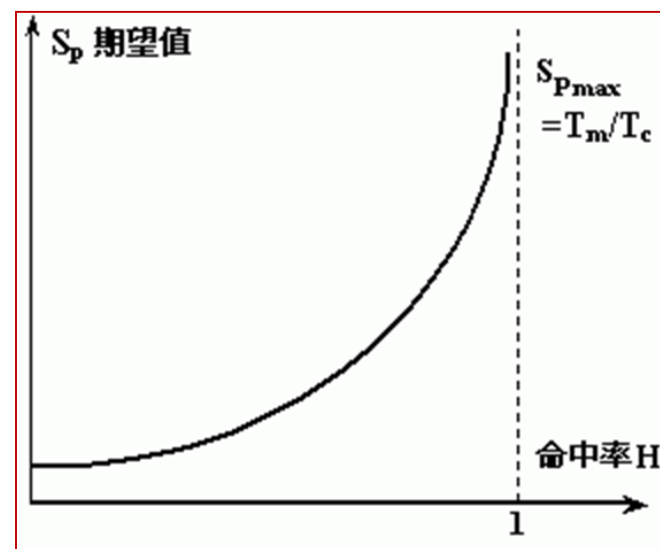
则存储系统的等效访问周期 T 为：

$$T = T_c \times H + T_m \times (1 - H)$$

■ 加速比SP (Speedup)

存储系统的加速比 S_p 为：

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



加速比与命中率的关系

Cache的容量

■ Cache的容量

不作特殊申明时，**Cache**的容量指**Cache**数据块的容量；
Cache实际总的存储容量实际上还包含**tag**和**valid bit**的位数。

- 例：假设一直接映射像**Cache**，有**16KB**数据，块大小为**4个字（32位字）**，主存地址**32位**，那么**Cache**总共有多少位？

Cache每数据块大小： $4 \times 32 = 128 \text{ bits} = 2^4 \text{ Bytes}$;

Cache块数： $16\text{K} \div 2^4 = 2^{10}$ 块；

tag位数： $32 - 10 - 1 = 21 \text{ bits}$

有效位：1位

Cache实际总容量： $2^{10} \times (128+21+1) = 147\text{K位} \approx 18.4\text{KB}$

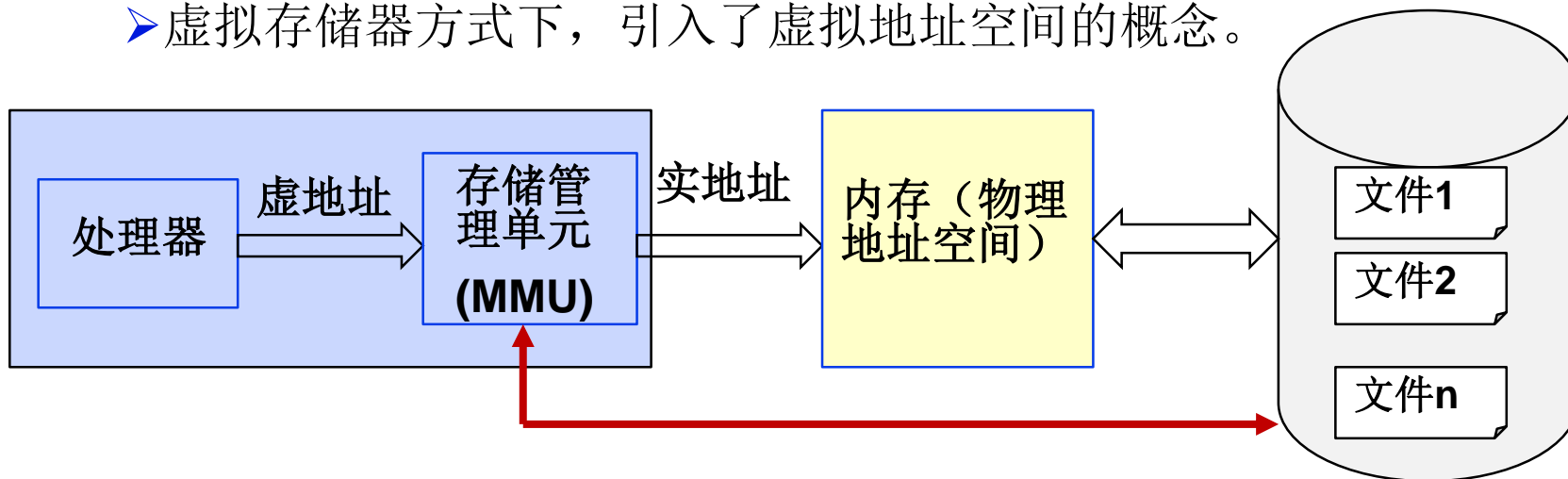
计算机系统包含32K字的主存，Cache容量4K字，每组4 Blocks，每Block 64个字。假设Cache开始是空的，CPU顺序从存储单元0，1，2到4351中读取字，然后再重复这样的取数9次，Cache速度是主存速度的10倍，采用LRU替换算法，假定块替换的时间忽略不计。

- 1) 计算上述取数过程的命中率；
- 2) 计算采用Cache后的加速比。

虚拟存储器

■ 基本原理

- 内存管理采用交换机制（硬件和操作系统实现），进程保存在辅存中，进程执行时，只将其**活跃部分**调入内存（局部性原理）。此时主存可以视为辅存的“高速缓存”。
- 这样一种**把主存当做辅助存储器的高速缓存技术**，称为虚拟存储技术，也称为虚拟存储器（**virtual memory**）。
- 虚拟存储器能从逻辑上为用户提供一个比物理存贮容量大得多、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。
- 虚拟存储器方式下，引入了虚拟地址空间的概念。

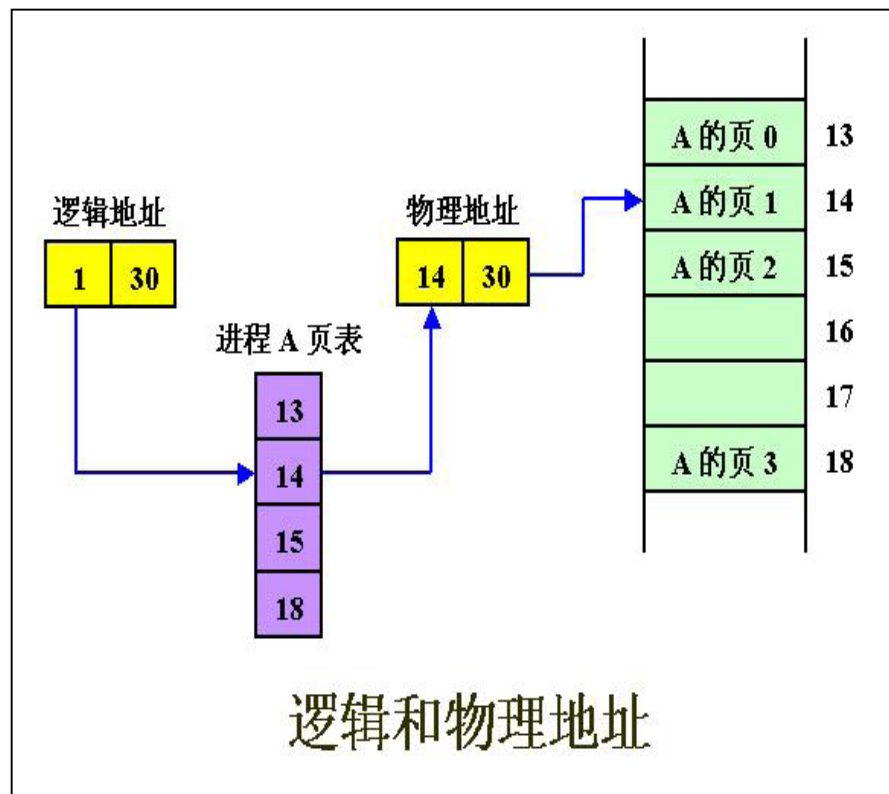


页式虚拟存储器

❖ 页式虚拟存储器

- 虚存（逻辑地址空间）和主存（物理地址空间）按固定大小分成若干页，虚存页称为**虚页**，主存页称为**实页**。辅存中程序按页调入内存；
- **页表**：记录虚页与实页的映射关系，实现虚实地址的转换，页表建立在内存中，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：
- 实地址格式：

虚页号	页内地址
实页号	页内地址



页式虚拟存储器

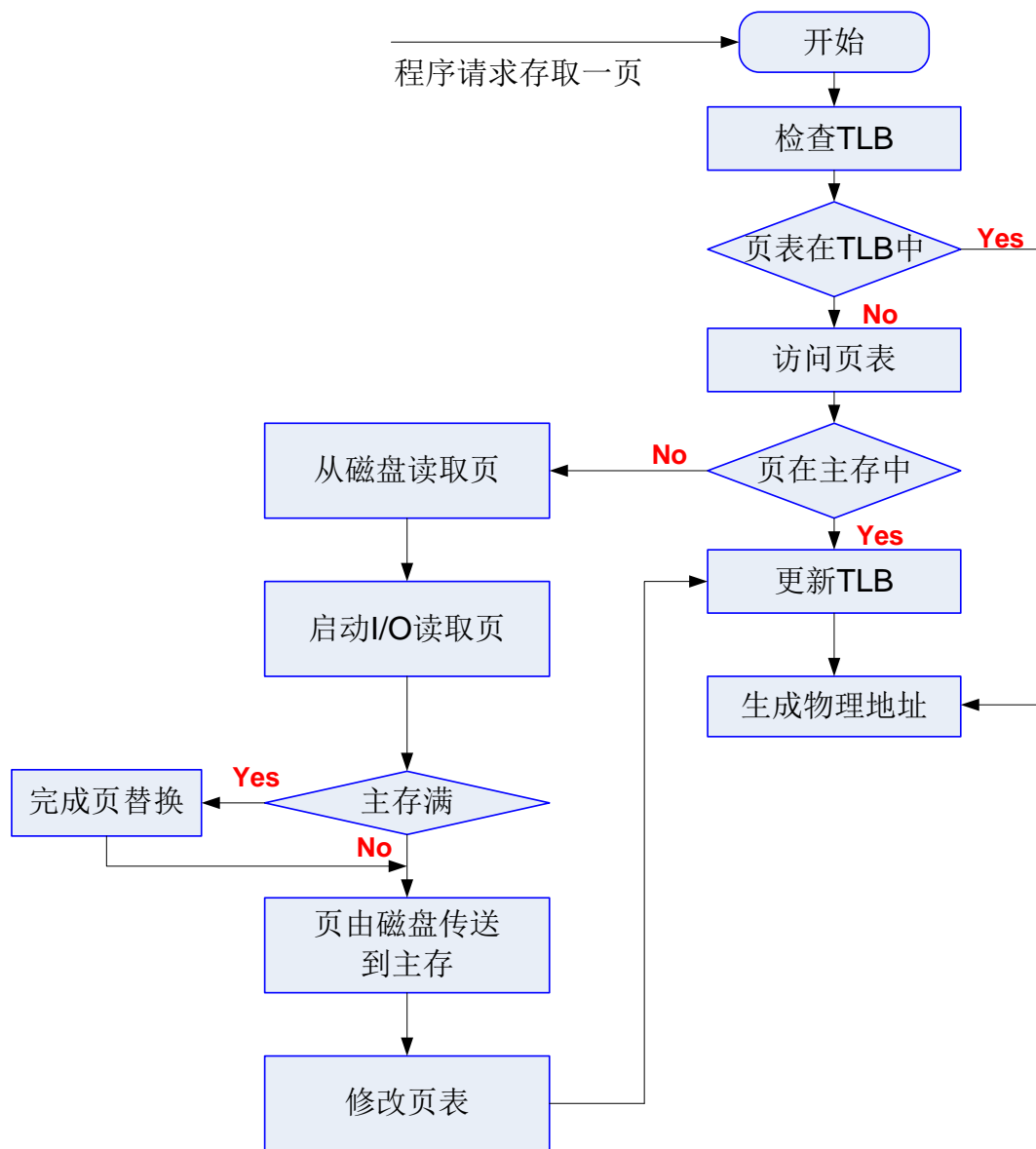
❖ 加快地址转换，采用快表**TLB**（**T**ranslation **L**ookaside **B**uffer，转换后备缓冲器）

- 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

有效位 修改位		标记（tag）	数据
		虚页号	实页号
		虚页号	实页号
		虚页号	实页号
		虚页号	实页号

快表（TLB）

页式虚拟存储器



页式虚拟存储器

假定页式虚拟存储系统按字节编址，逻辑地址36位，页大小16KB，物理地址32位，页表中包括有效位和修改位各1位、使用位和存期方式位各2位，且所有虚拟页都在使用中。请问：

- (1) 每个进程的页表大小至少为多少？
- (2) 如果所使用的快表（TLB）总表项数为256项，且采用2路组相联Cache实现，则快表大小至少为多少？

❖ 解答（1）

页面大小： **$16\text{KB}=2^{14}$** ，页内偏移**14位**

虚地址36位：**虚页号** **$=36-14=22$** 位

实地址32为：**实页号** **$=32-14=18$** 位

每个进程最多可有： **2^{22} 个虚页**

每个页表项： **$1+1+2+2+18=24$** 位

每个页表所占空间： **$2^{22} \times 24=12\text{MB}$**

(2)

TLB：**256**个表项，**2**路组相联，所以共有**128**组

22位虚页号：**7**位组地址，**15**位Tag

TLB每个表项： **$15+24=39$** 位

TLB容量： **$39 \times 256=9984$** 位=**1248**字节

辅助存储器：磁盘

❖ 数据结构与地址格式

➤ 数据结构：

- 磁道（柱面：Cylinder）
- 盘面（磁头：Head）
- 扇区（Sector）

➤ 扇区容量：512 Bytes

➤ 每个磁道包含的扇区数相同

➤ 最小访问单位：扇区

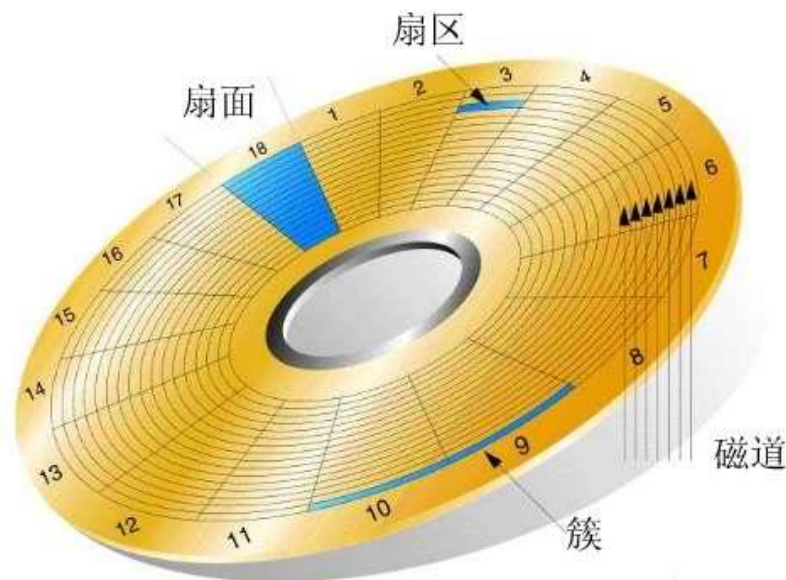
➤ 扇区的地址表示：

扇区地址：

Cylinder #

Head #

Sector #



磁盘上的磁道、扇区和簇

❖ 容量与数据传输率计算

某计算机字长为 32 位，CPU 主频为 500MHz，磁盘共有 16 个盘面，512 个柱面，每磁道包含 100 个扇区，每个扇区 512 字节，该磁盘旋转速度为 12000 RPM。

- (1) 计算该磁盘总容量？（2 分）
- (2) 计算该磁盘的数据传输率（bits/s）？（3 分）
- (3) 若磁盘采用 DMA 方式传送数据，每次 DMA 传送块大小为 10KB，DMA 预处理和后处理总开销为 500 个时钟周期，则 CPU 用于该磁盘 I/O 的时间占整个 CPU 时间的百分比是多少？（假设 DMA 与 CPU 之间没有访内冲突）。（5 分）

$$(1) \text{ 总容量} = 16 * 512 * 100 * 512 \text{B} = 400 \text{MB}$$

$$(3) \text{ 1 秒内 DMA 次数} = 200 * 50 \text{KB} / 10 \text{KB} = 1000 \text{ 次}$$

$$(2) \text{ 道容量} = 100 * 512 = 50 \text{KB}$$

$$\text{1 秒内 DMA 耗时} = 1000 * 500 = 5 * 10^5 \text{ 时钟周期}$$

$$\text{转速} = 12000 / 60 = 200 \text{r/s}$$

$$\text{百分比} = 5 * 10^5 / 500 * 10^6 = 0.1\%$$

$$\text{传输率} = 200 * 50 \text{KB/s}$$

$$= 200 * 50 * 2^{10} * 8 \text{bit/s}$$

$$= 81920000 \text{bit/s}$$

第三部分：MIPS数据通路

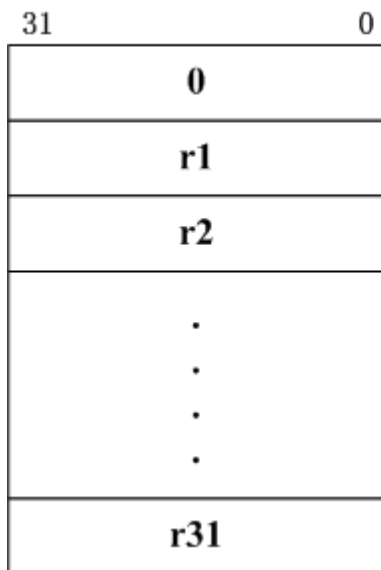
- ❖ MIPS 指令系统
- ❖ 单周期数据通路
- ❖ 流水线数据通路

MIPS 指令集架构

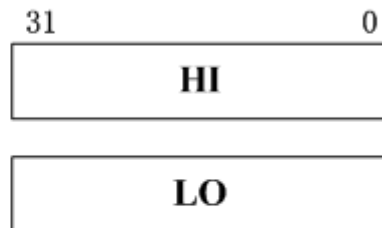
❖ MIPS R2000/R3000 寄存器结构

- 32位虚拟地址空间
- 32个32位GPRs（通用寄存器）
- 32个32位FPRs（浮点数寄存器）
- HI, LO, PC

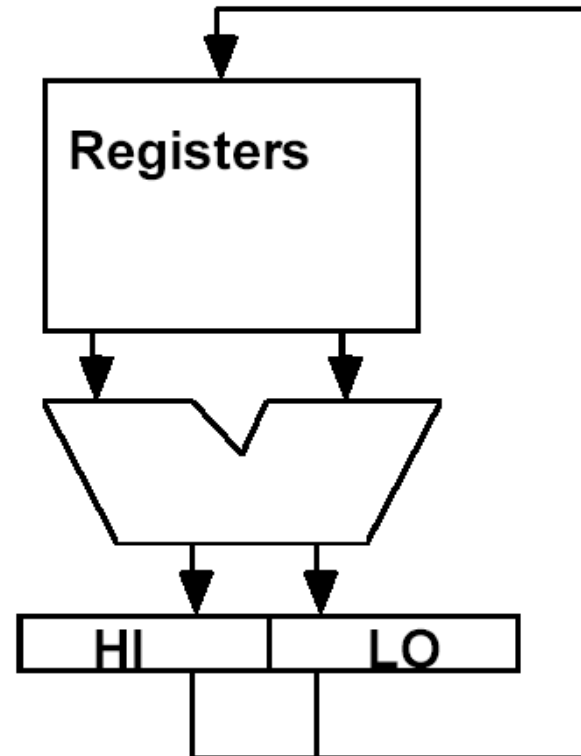
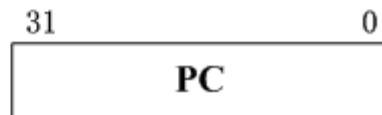
通用寄存器



乘/除寄存器



程序计数器



MIPS模型机指令集

❖ 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)

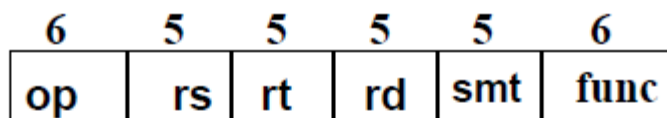
	6	5	5	5	5	6
R类型	Op (31-26)	Rs (25-21)	Rt (20-16)	Rd (15-11)	Shamt (10-6)	Func (5-0)
I类型	Op (31-26)	Rs (25-21)	Rt (20-16)	16 bit Address or Immediate (15-0)		
J类型	Op (31-26)	26 bit Address (for Jump Instruction) (25-0)				

MIPS模型机指令集

❖ MIPS 寻址方式

R-format:

Register (direct)



register

I-format:

Immediate



Base或index



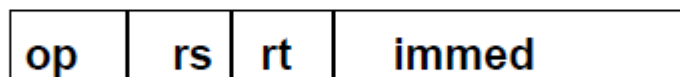
register

+

Memory

B/HW/W

PC-relative



PC + 4

+

Memory

J-format:

Pseudodirect



Memory

MIPS模型机指令集

模型机指令集（8条指令）

指令格式	指令	功能	说明
R 类型	add rd, rs, rt	$R[rd] \leftarrow R[rs] + R[rt]$	加运算 : 寄存器 rs 和寄存器 rt 相加, 结果送寄存器 rd
	sub rd, rs, rt	$R[rd] \leftarrow R[rs] - R[rt]$	减运算 : 寄存器 rs 和寄存器 rt 相减, 结果送寄存器 rd
	and rd, rs, rt	$R[rd] \leftarrow R[rs] \& R[rt]$	与运算 : 寄存器 rs 和寄存器 rt 按位与, 结果送寄存器 rd
	or rd, rs, rt	$R[rd] \leftarrow R[rs] R[rt]$	或运算 : 寄存器 rs 和寄存器 rt 按位或, 结果送寄存器 rd
I 类型	lw rt, rs, imm16	$Add = R[rs] + \text{Signext}(imm16)$ $R[rt] \leftarrow M[Add]$	取字 : 寄存器 rs 和立即数 imm16 (符号扩展至 32 位) 相加得到内存地址, 从内存该地址单元读取数据送 rt
	sw rt, rs, imm16	$Add = R[rs] + \text{Signext}(imm16)$ $M[Add] \leftarrow R[rt]$	存字 : 寄存器 rs 和立即数 imm16 (符号扩展至 32 位) 相加得到内存地址, 寄存器 rt 数据写入内存该地址单元
	beq rs, rt, imm16	If ($R[rs] - R[rt] = 0$) then $PC \leftarrow PC + \text{Signext}(imm16) \ll 2$	分支 : 如果寄存器 rs 与 rt 相等, 则转移 (imm16 符号扩展至 32 位), 否则顺序执行。(取指令后, PC+4)
J 类型	j target	$PC(31:2) \leftarrow PC(31:28) \parallel \text{target}(25:0)$	跳转 : 当前 PC 的高 4 位与 target (26 位) 拼接成 30 位目标地址送 PC (31:2)。(取指令后, PC+4)。

MIPS模型机指令集

模型机指令编码

R 类型格式	OP (31 ~ 26)	Rs (25 ~ 21)	Rt (20 ~ 16)	Rd (15 ~ 11)	Shamt (10 ~ 6)	Funct (5 ~ 0)
add rd, rs, rt	000000	rs	rt	rd	XXXXXX	100000
sub rd, rs, rt	000000	rs	rt	rd	XXXXXX	100010
and rd, rs, rt	000000	rs	rt	rd	XXXXXX	100100
or rd, rs, rt	000000	rs	rt	rd	XXXXXX	100101

I 类型格式	OP (31 ~ 26)	Rs (25 ~ 21)	Rt (20 ~ 16)	16 bits immediate or address (15 ~ 0)
lw rt, rs, imm16	100011	rs	rt	imm16
sw rt, rs, imm16	101011	rs	rt	imm16
beq rs, rt, imm16	000100	rs	rt	imm16

J 类型格式	OP (31 ~ 26)	26 bits address
j target	000010	target

汇编语言语句

MIPS汇编中的3类语句

❑ 可执行指令：为处理器生成在运行时执行的机器码

➤ 如add、addi、beq、jal

❑ 伪指令和宏：便于编程人员理解，由汇编程序翻译成真正的指令，可执行

➤ 如la \$s0, table(取地址)、li \$v0,10(赋值)、move \$t8, \$sp(寄存器传送)

❑ 汇编伪指令：为汇编翻译程序提供信息，用来定义段、分配内存变量等，不可执行

➤ **.DATA** 伪指令：定义程序的**数据段**

- 程序的变量需要在该伪指令下定义
- 汇编程序会分配和初始化变量的存储空间

➤ **.TEXT** 伪指令：定义程序的**代码段**

➤ **.GLOBL** 伪指令：声明一个符号为**全局的**

- 全局符号可以被其它的文件引用
- 用该伪指令声明一个程序的 *main* 过程

汇编程序模板

```
# Sum of three integers
# Objective: Computes the sum of three integers.
#   Input  : Requests three numbers.
#   Output : Outputs the sum.

##### Data segment #####
.data
prompt:      .asciiz  "Please enter three numbers: \n"
sum_msg:     .asciiz  "The sum is: "

##### Code segment #####
.text
.globl main
main:
    la  $a0,prompt    # display prompt string
    li  $v0,4
    syscall
    li  $v0,5          # read 1st integer into $t0
    syscall

    .....
    move $a0,$t0      # output sum
    li  $v0,1
    syscall
    li  $v0,10        # exit
    syscall
```

程序说明

数据段：定义程序中的变量，编译程序将为这些变量分配存储空间

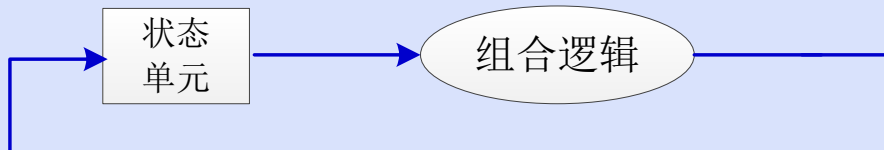
代码段：程序代码部分，包含一个全局main过程。

数据通路的时钟同步方法

❖ 时钟同步方法

- 以时钟周期信号为基准，确定数据读出和写入的时刻。
- 采用边沿触发的时钟同步方法，如下跳沿触发，意味着所有状态元件（寄存器、存储器）的数据写入都发生在时钟周期的下跳沿时刻。

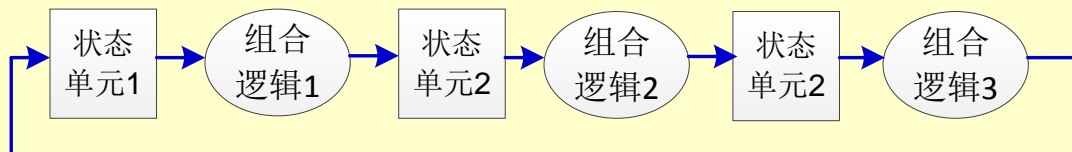
时钟



单周期通路同步

- 组合逻辑操作时钟周期内完成；
- 信号在时钟周期内从状态单元输出经组合逻辑再回到状态单元输入。
- 时钟信号上跳沿同步

时钟



多周期通路同步

- 组合逻辑操作时钟周期内完成；
- 所有信号在时钟周期内从状态单元1经组合逻辑传送到状态单元2。
- 时钟信号上跳沿同步

单周期数据通路设计

❖ 单周期

- 所有指令执行周期固定为单一时钟周期，**CPI=1**。

❖ 通路设计考虑

- **哈佛体系结构**：使用指令存储区（IM）和数据存储区（DM）分别保存指令和数据
- 先为每类指令设计独立的数据通路，然后再考虑数据通路合并。

❖ 指令执行的共性

- 根据PC从指令存储器读取指令，取指令后，**PC+4**；
- 模型机7条指令在读取寄存器后，都要使用ALU
 - **LW/SW**（存储访问）指令用ALU计算数据地址
 - **ADD/SUB/AND/OR**(算术逻辑)指令用ALU完成算术逻辑运算
 - **BEQ**（分支）指令用ALU进行比较（减法运算）

单周期数据通路设计

❖ 分析指令执行步骤，确定数据通路所需部件和部件间连接

➤ 模型机指令执行过程一般会分为如下几个步骤：

- 取指令：根据PC访问指令存储器获得指令，然后PC+4；
- 读寄存器：根据指令格式读取相应寄存器操作数
- ALU运算：在ALU完成相应的算术逻辑运算
- 数据存取：LW/SW指令的数据存储器访问
- 写寄存器：运算类指令和LW指令要把数据写入寄存器

❖ 使用数据通路设计表格

- 表格记录数据通路部件的
- 暂不考虑控制信号

指令	Adder		PC	IM Add.	Registers				ALU		DM	
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata

单周期数据通路设计的一般性方法

单指令 数据通路 构造

- for each 指令
 - for each 新增需求
 - case 可以合并至已有部件：
修改部件设计描述、HDL建模：{F', I', O'}
 - case 需要新增部件：
建立新部件设计描述、HDL建模：{F, I, O}
增加新部件
 - for each 部件
 - 设置输入来源（使用设计表格）

多数据通 路综合

- 按垂直方向合并数据通路，并去除相同项
- for each 输入来源多余1个的输入端
 - 部署1个MUX (MUX的输入规模为输入来源数)
- MUX设计定义、HDL建模

系统实现

- HDL建模：连接所有的部件及所有的MUX

单周期数据通路设计——LW指令数据通路

3. 取数指令（lw）数据通路

➤ lw rt, rs, imm16

➤ 功能描述：

▪ $R[rt] \leftarrow DM[R[rs] + \text{Signext}(imm16)]$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

Op (31-26)	Rs (25-21)	Rt (20-16)	16 bit Address or Immediate (15-0)
---------------	---------------	---------------	---------------------------------------

指令	Adder		PC	IM Add.	Registers				ALU		DM		Sign-ext
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata	
R型指令	PC	4	Adder	PC	Rs	Rt	Rd	ALU	Rdata1	Rdata2			
Lw	PC	4	Adder	PC	Rs		Rt	DM	Rdata1	Sign-ext	ALU		imm16
Sw	PC	4	Adder	PC									
Beq	PC	4	Adder	PC									

单周期数据通路设计——LW指令数据通路

3. 取数指令（lw）数据通路

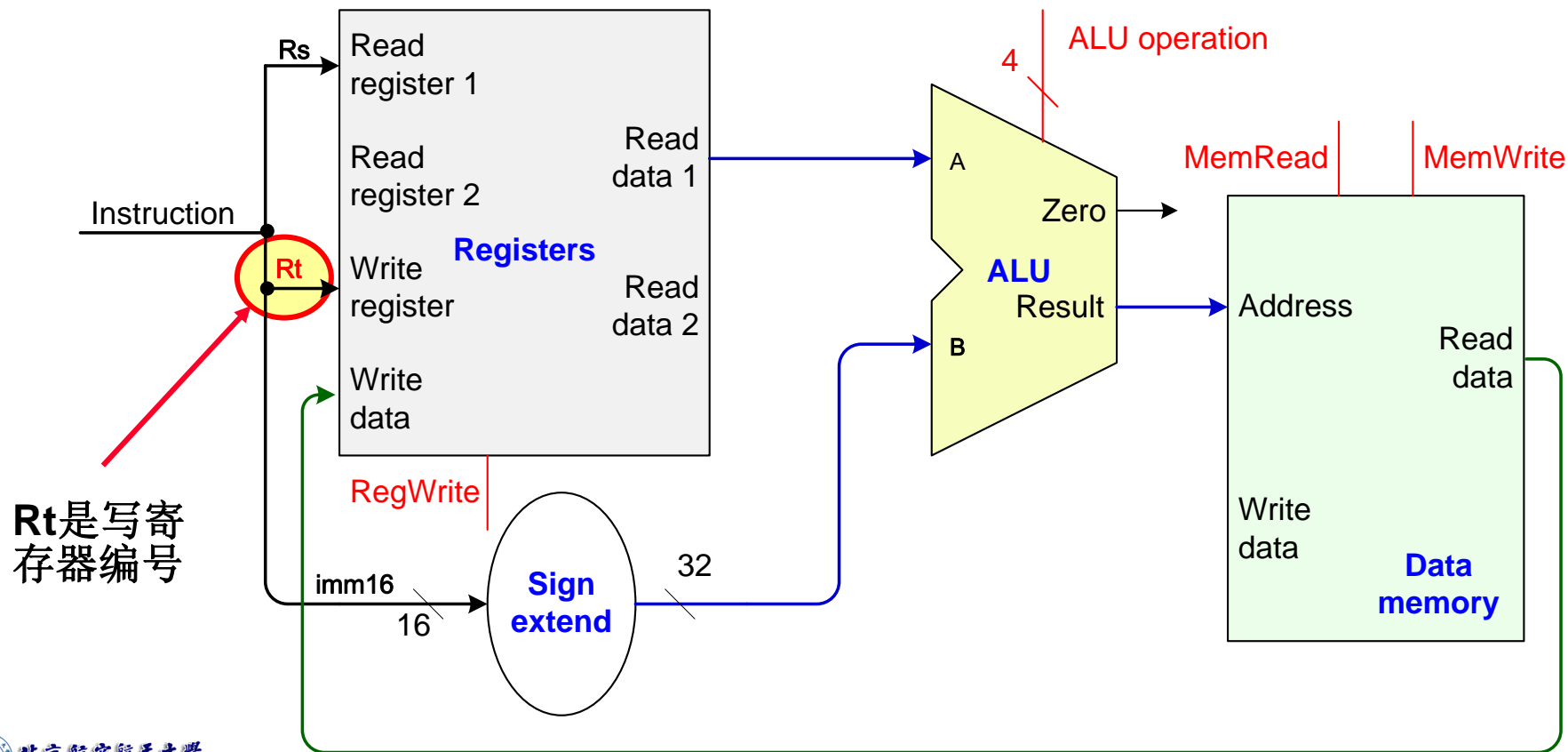
➤ lw rt, rs, imm16

➤ 功能描述：

$$R[rt] \leftarrow DM[R[rs] + \text{Signext}(imm16)]$$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

Op (31-26)	Rs (25-21)	Rt (20-16)	16 bit Address or Immediate (15-0)
---------------	---------------	---------------	---------------------------------------



单周期数据通路设计

7. MIPS数据通路再合并

➤支持：R类型指令、内存访问指令（lw/sw）、beq指令

指令	Adder		PC	IM Add.	Registers				ALU		DM		Sign- ext	Nadd	
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata			
R型与访存	PC	4	Adder	PC	Rs	Rt	Rd Rt	ALU DM	Rdata1	Rdata2 Sign-ext	ALU	Rdata2	imm16		
Beq	PC	4	Adder Nadd	PC	Rs	Rt			Rdata1	Rdata2			imm16	Adder	Shift
合并	PC	4	Adder Nadd	PC	Rs	Rt	Rd Rt	ALU DM	Rdata1	Rdata2 Sign-ext	ALU	Rdata2	imm16	Adder	Shift

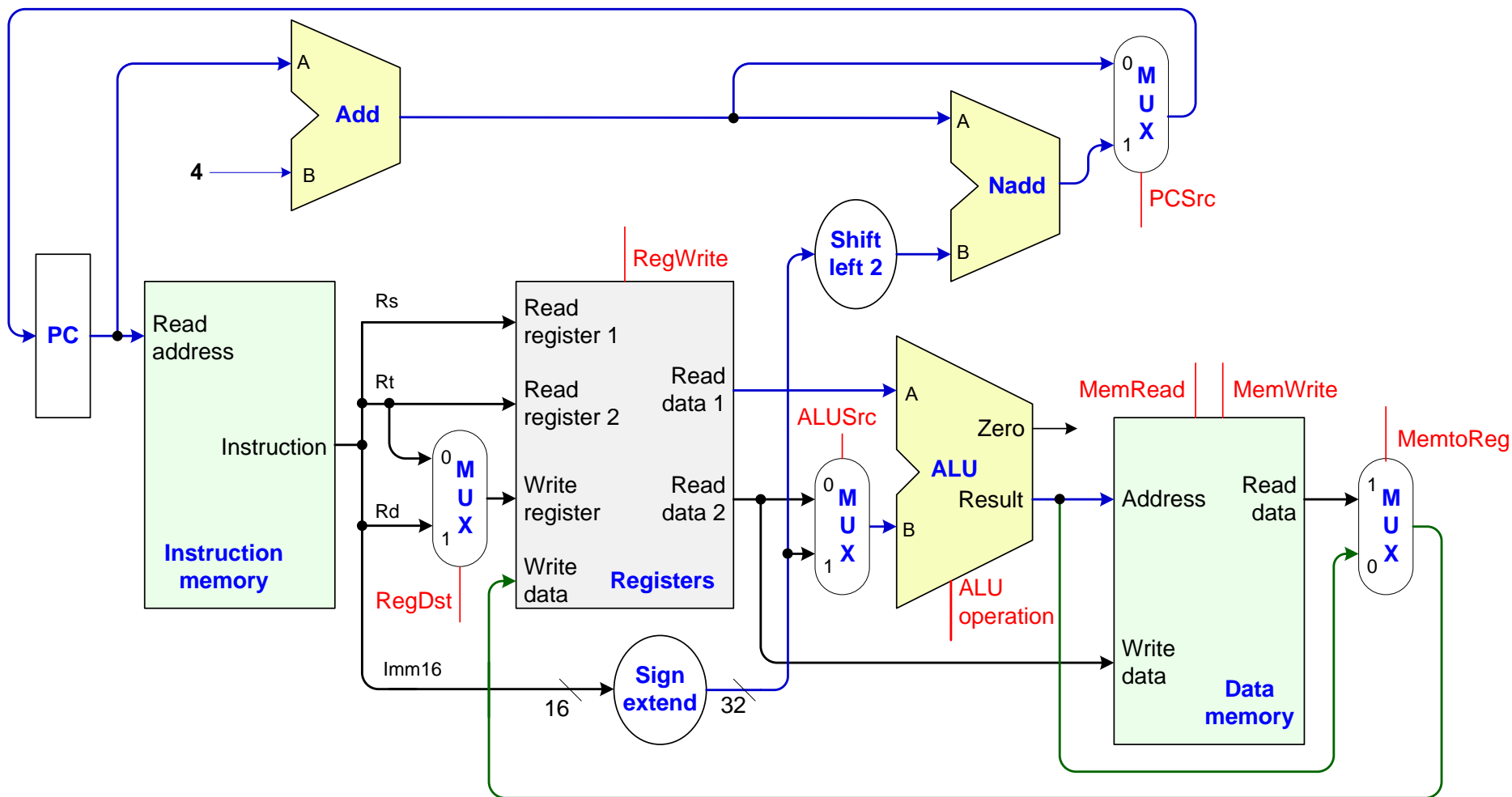
需要4个二选一多路选择器MUX

- PC输入端数据源选择MUX，选择控制信号 PCSrc
- 寄存器堆写入端地址选择MUX，选择控制信号 RegDst
- ALU输入端B数据源选择MUX，选择控制信号 ALUSrc
- 寄存器堆写入端数据源选择MUX，选择控制信号 MemtoReg

单周期数据通路设计

7. MIPS数据通路合并

➤ 支持：R类型指令、内存访问指令（lw/sw）、beq指令



单周期控制器设计

❖ 单周期通路所需控制信号

- **ALU控制 (ALU Operation)** : 4位
- 其他控制信号: 7个

ALU 控制

输入		ALU operation	ALU运算
A	B	0000	A & B
A	B	0001	A B
A	B	0010	A + B
A	B	0110	A - B

7个控制信号

控制信号	失效时作用	有效时作用
RegDst	寄存器堆写入端地址来选择Rt字段	寄存器堆写入端地址选择 Rd字段
RegWrite	无	把数据写入寄存器堆中对应寄存器
ALUSrc	ALU输入端B选择寄存器堆输出R[rt]	ALU输入端B选择Signext输出
PCSrc	PC输入源选择 PC+4	PC输入选择beq指令的目的地址
MemRead	无	数据存储器DM读数据 (输出)
MemWrite	无	数据存储器DM写数据 (输入)
MemtoReg	寄存器堆写入端数据来自ALU输出	寄存器堆写入端数据来自DM输出

单周期控制器设计

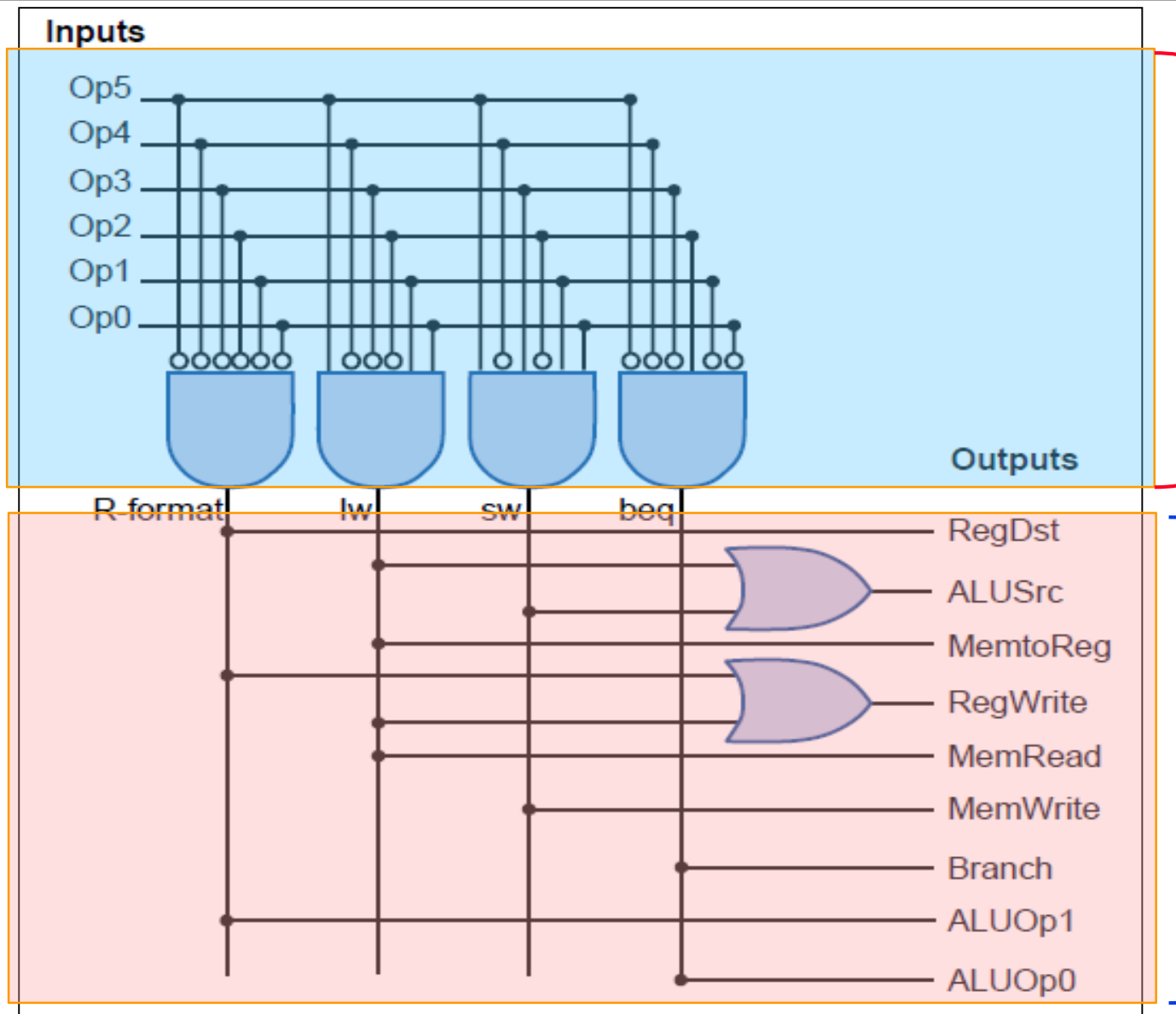
主控单元真值表

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



提供给ALU控制单元

单周期控制器设计

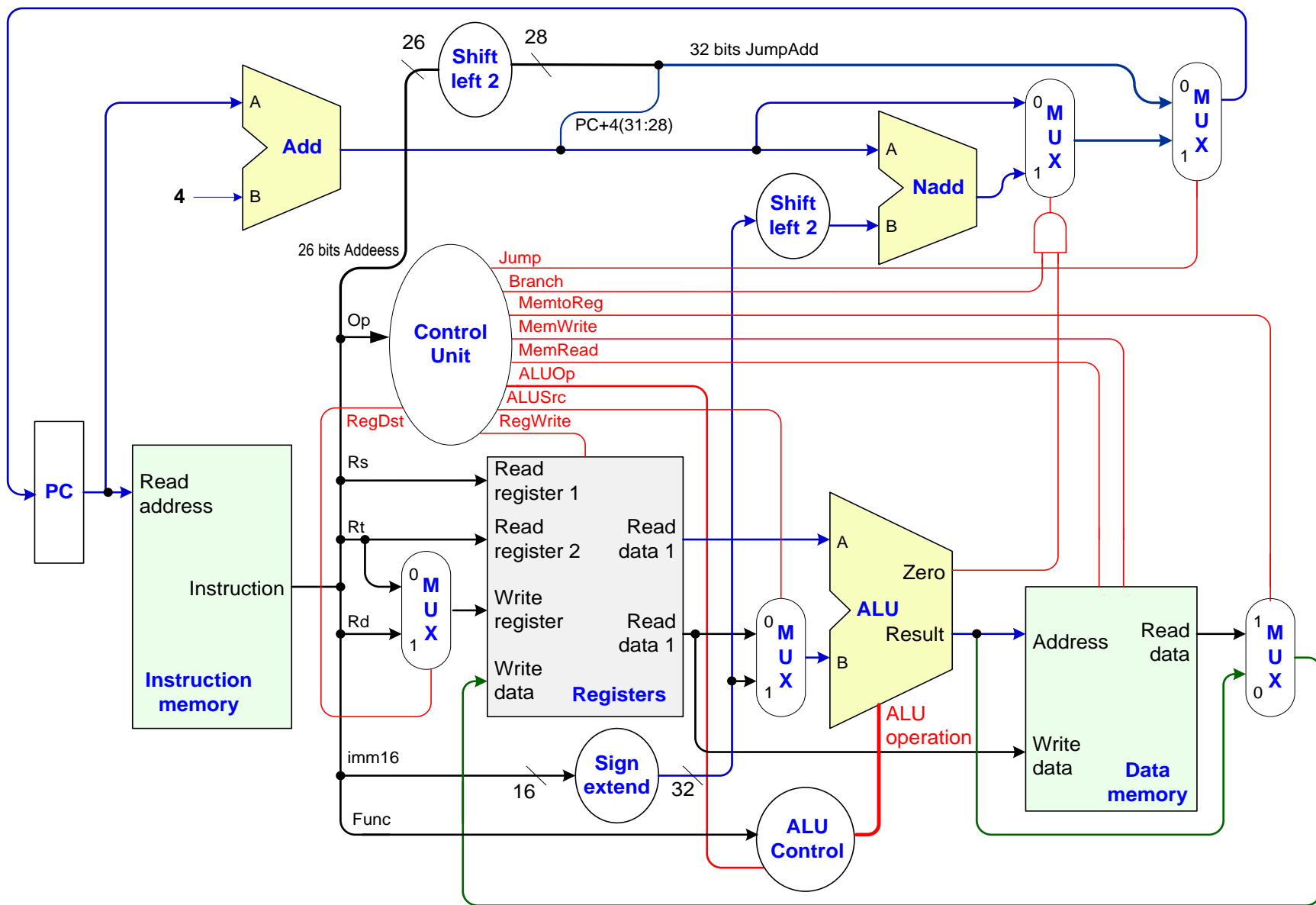


AND逻辑
实现指令
译码

OR逻辑
实现控制
信号

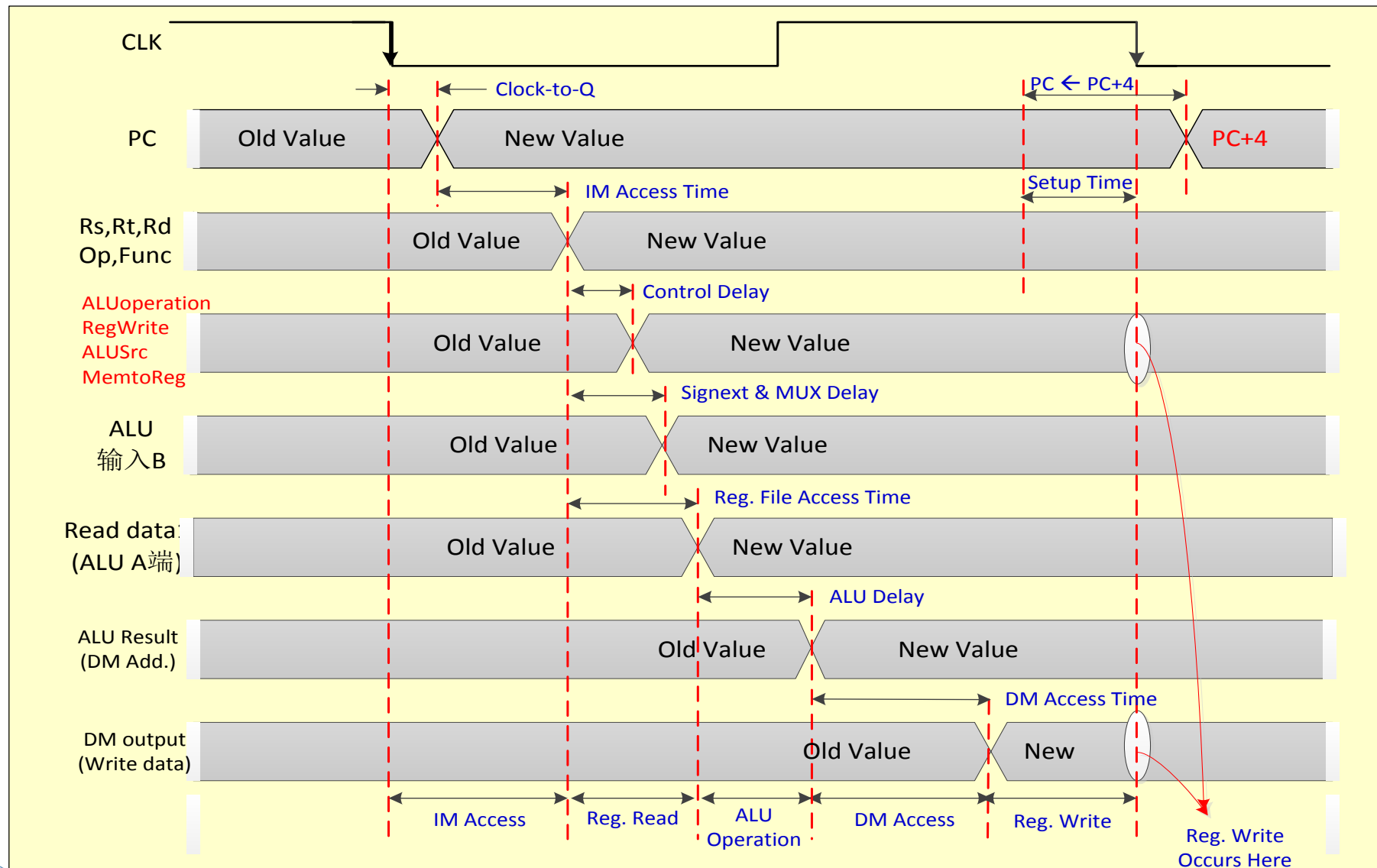
主控
单元
逻辑
实现

单周期控制器设计（包含跳转指令的数据通路）



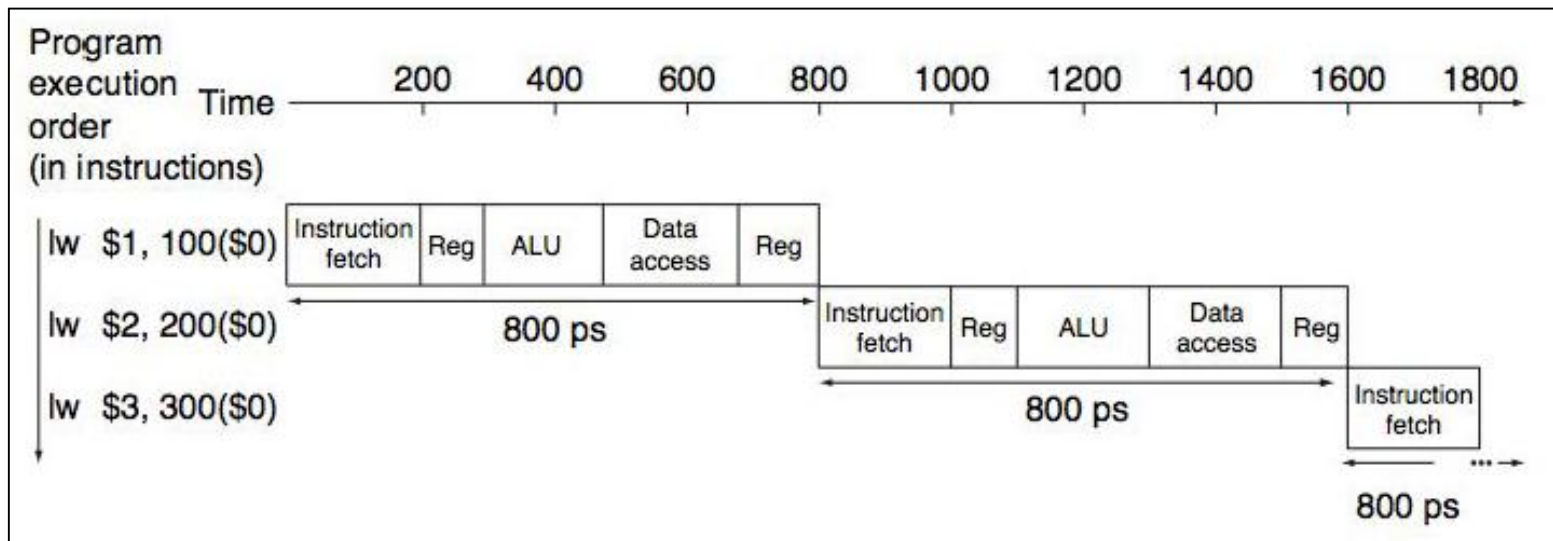
单周期数据通路性能分析

❖ LW指令的指令周期

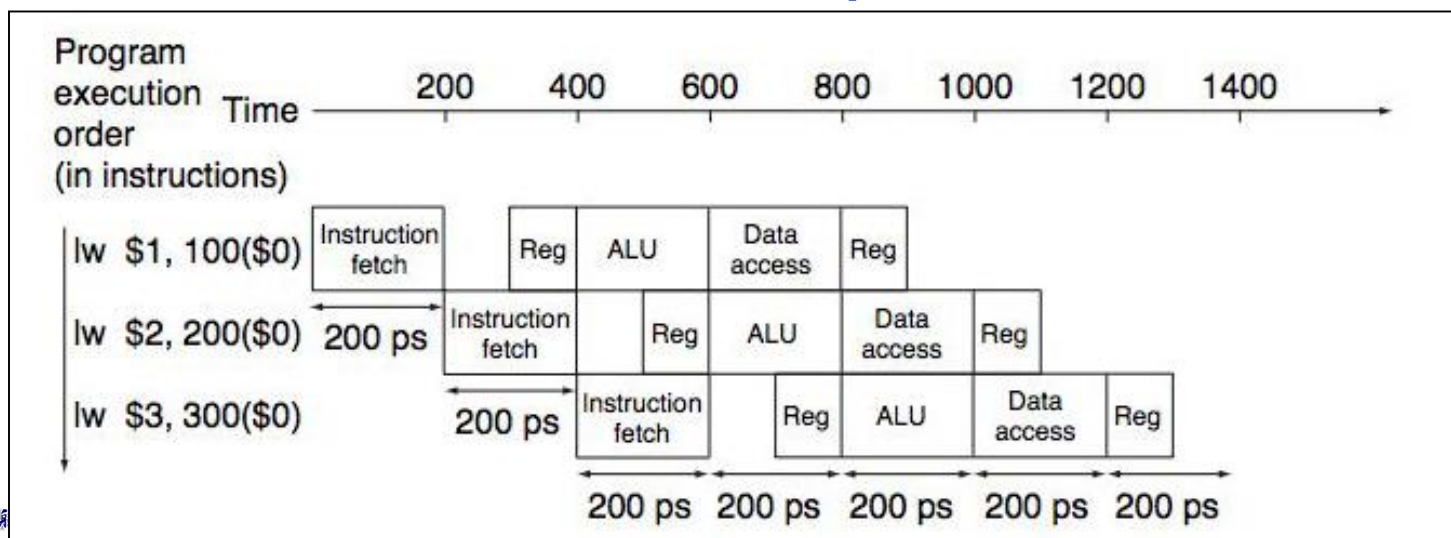


流水线原理

❖ 单周期模型



❖ 流水线模型（5个步骤，每个步骤200ps）



流水线原理

❖ 单周期模型

➤ 指令周期（时钟周期）：**800ps**

➤ **CPI = 1**

❖ 流水线模型

➤ 时钟周期等于最长步骤所花时间为：**200ps**

➤ 指令执行分**5步（5级流水）**，每步一个时钟周期，共 **1000 ps**

➤ **N**条指令的执行时间为： **$(1000+200*(N-1))ps$**

➤ 在指令数**N**很大时，比单周期方式提高约 **4 倍**

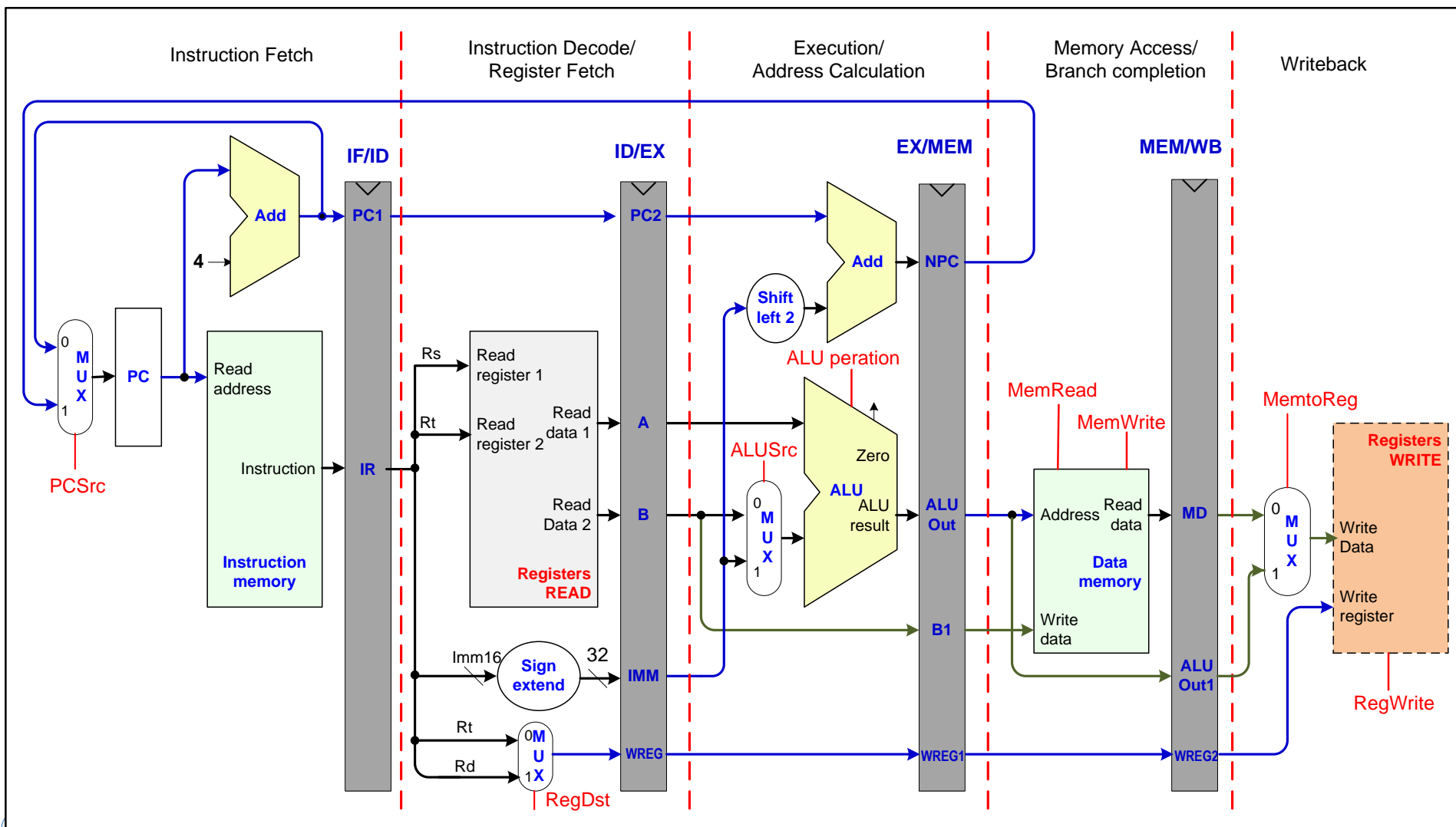
➤ 指令数**N**很大时，**CPI \approx 1**

- 流水线不改善单个任务处理**延迟**，但改善了整体工作负载的**吞吐率**
- 流水线速率受限于**最慢**的流水段
- **多个**任务同时工作，但占用**不同**的资源
- 潜在加速比 = **流水线级数**

MIPS流水线数据通路

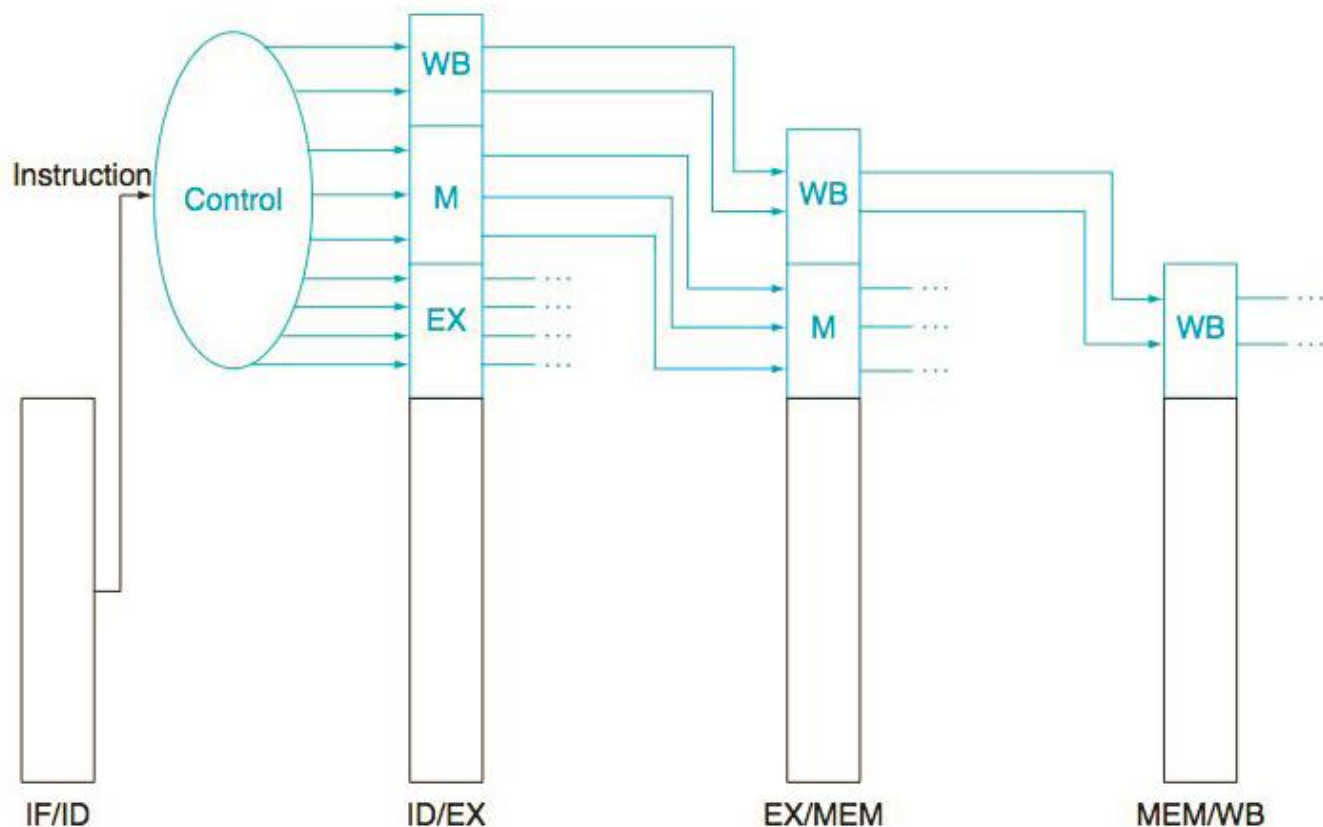
❖ 寄存器整合成流水线寄存器：IF/ID, ID/EX, EX/MEM, MEM/WB

➤ 每个时钟周期指令流和数据流都会从一个流水线寄存器传递到下一个流水线寄存器

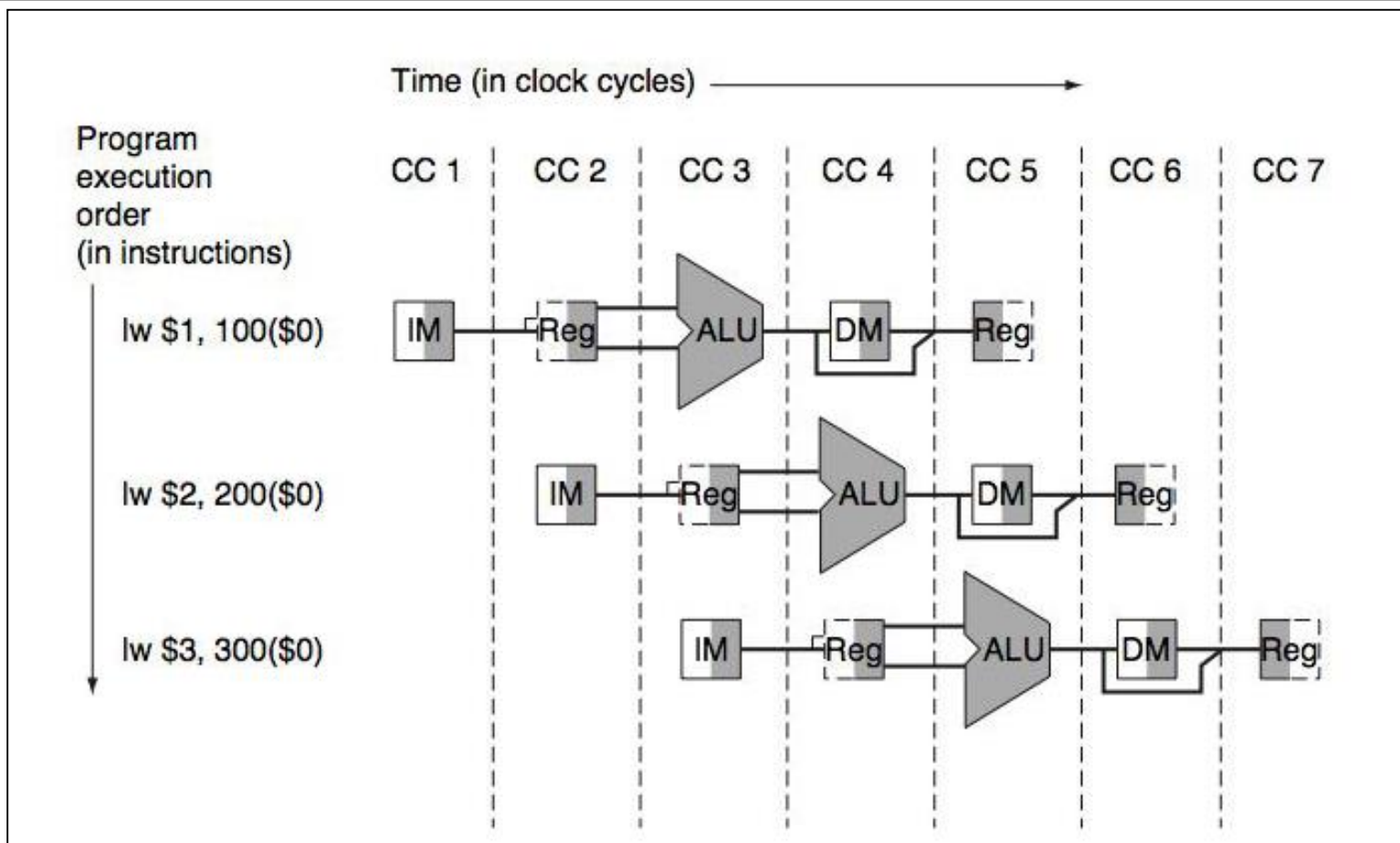


MIPS流水线数据通路——控制信号

- ❖ 控制器：译码产生指令执行所有控制信号，与单周期完全相同；每个控制信号只在所需要的流水段发生作用，这一点与单周期不同
- ❖ 控制信号流水寄存器：控制信号在寄存器中传递，直至不再需要



MIPS流水线数据通路----流水线模型



- 每一级都以该级使用的部件表示。如：IM表示IF阶段的指令存储器；
- 寄存器堆Reg和数据存储器DM右侧阴影表示读，左侧阴影表示写。
- 行：表示同一条指令在不同周期所经过的流水部件
- 列：表示同一周期不同指令所经过的流水部件

流水线的冒险

❖ **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, 指令流水线必须出现停顿。**

➤ **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。

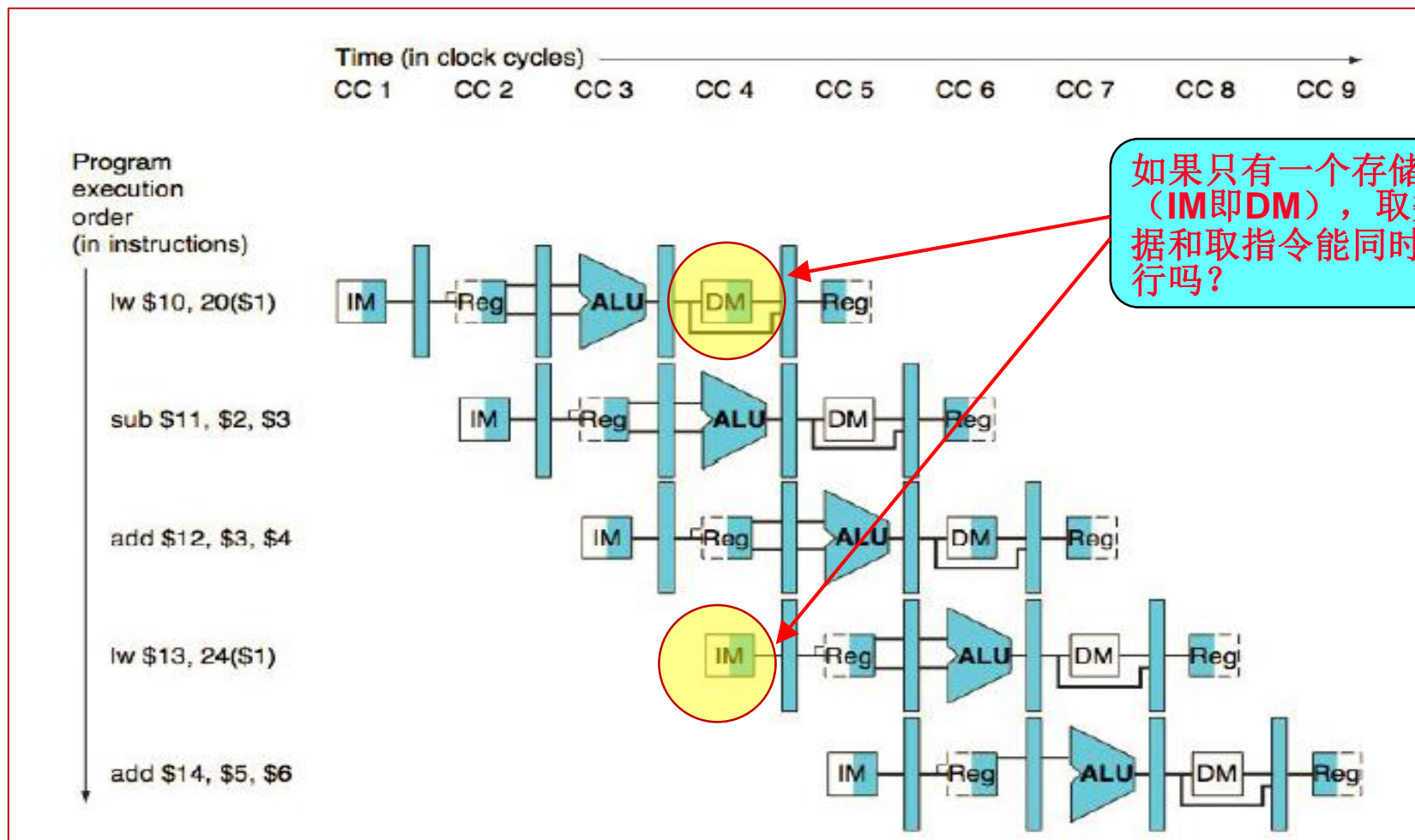
- lw/sw指令执行需要访问存储器, 指令取指阶段需要访问存储器, 将出现存储器使用冲突

➤ **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。

➤ **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



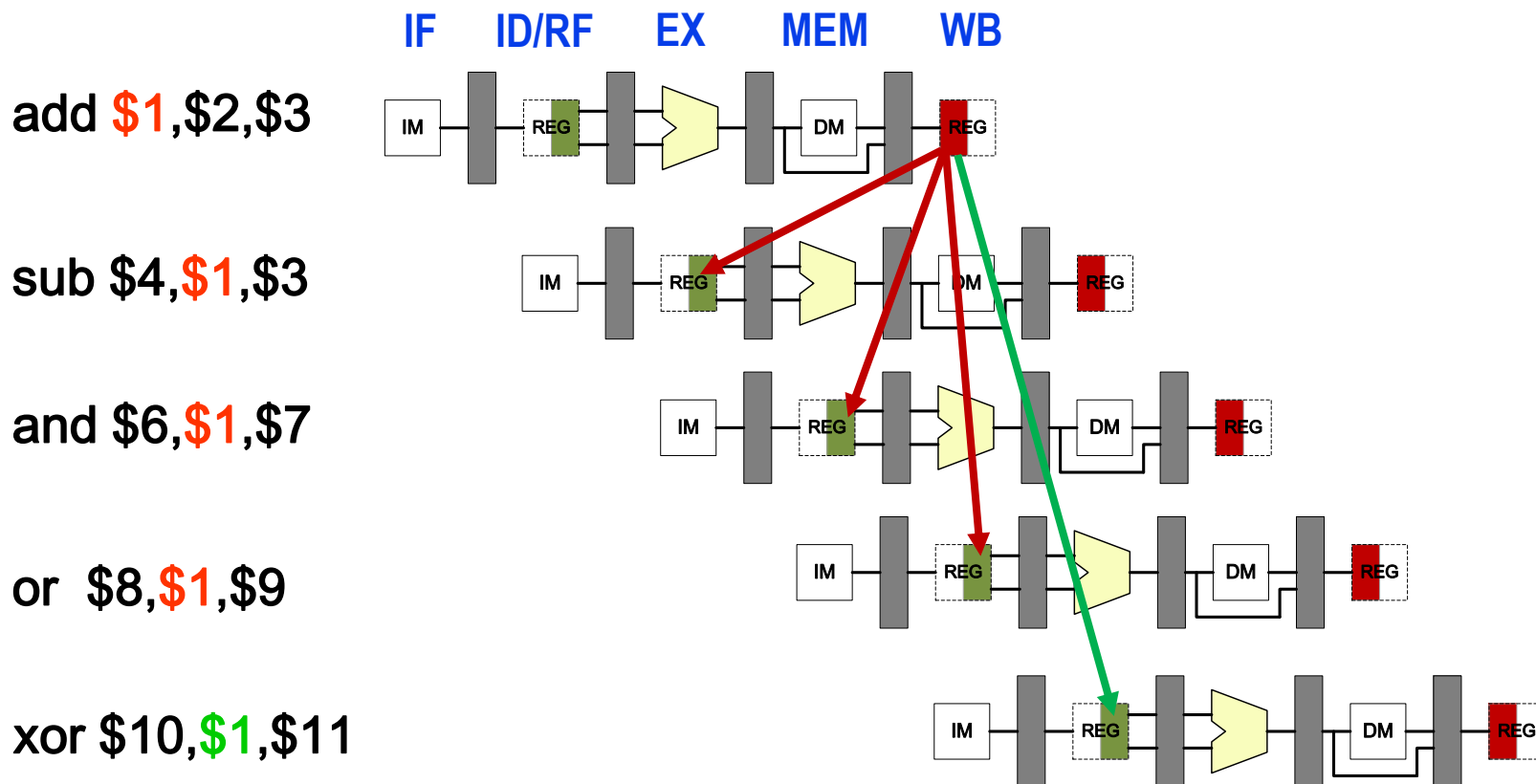
假定数据通路只有一个存储器，此时存储器访问将发生冲突！

流水线的冒险----数据冒险

❖ 数据冒险问题

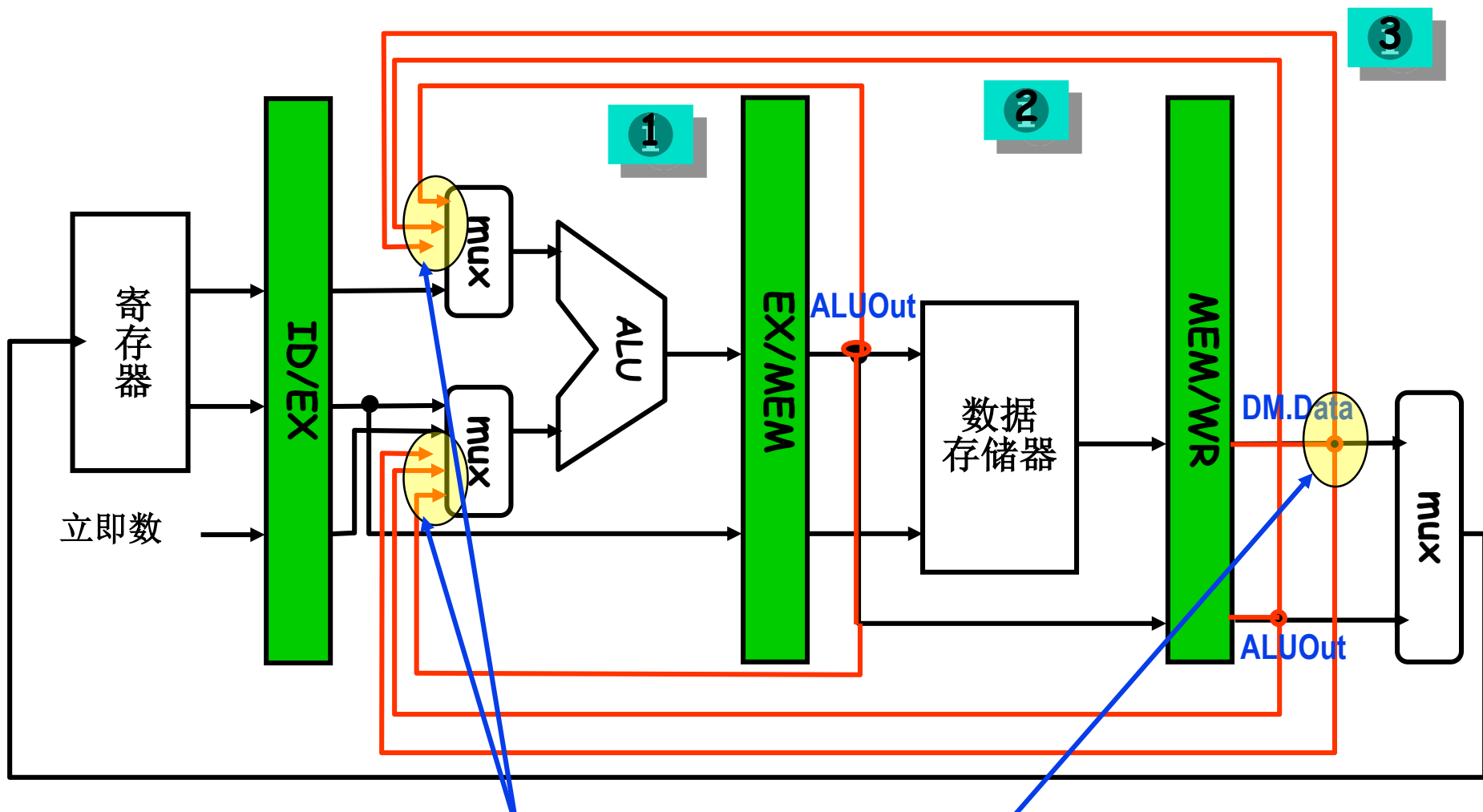
时间 (时钟周期)

指令执行次序



数据冒险的处理——旁路转发

❖ 旁路转发策略

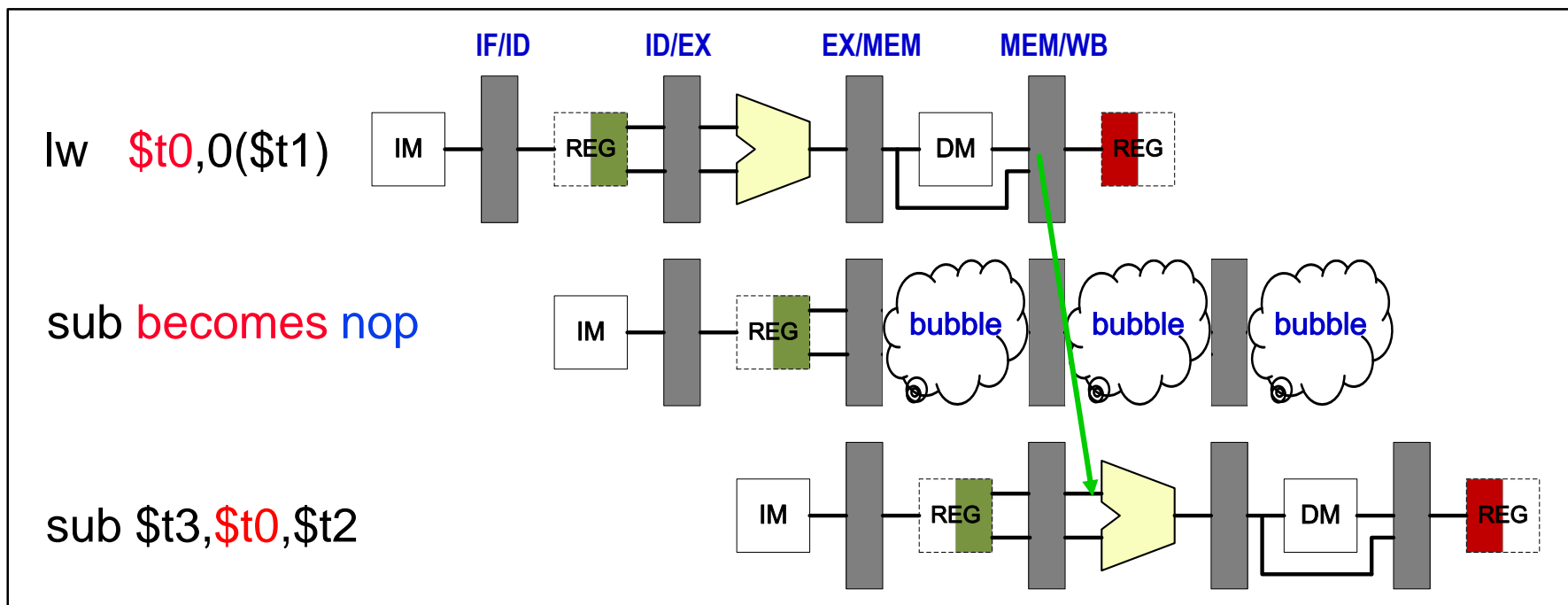
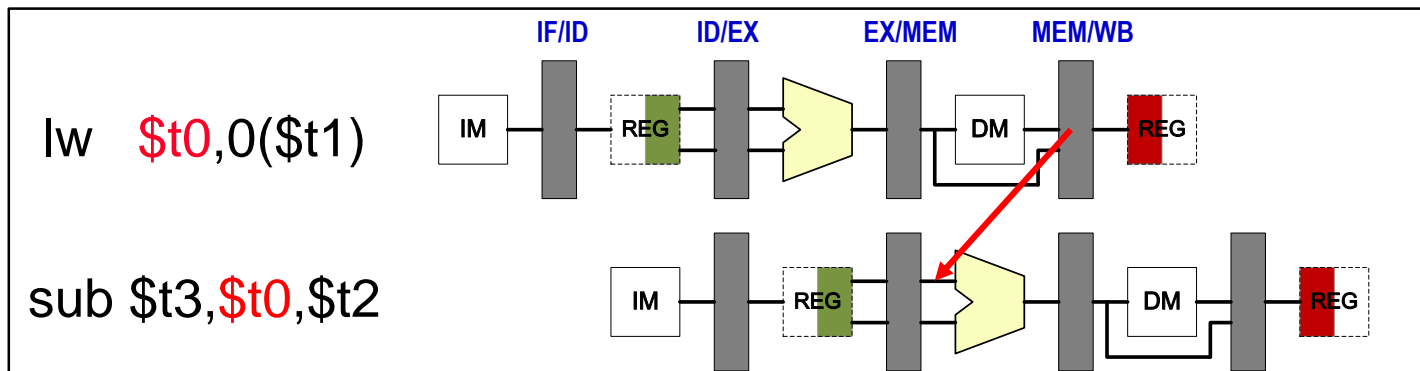


为什么要转发到ALU
两个输入端口？

这条转发通路为什么
情况而设计？

load数据冒险处理——停顿（插入nop）

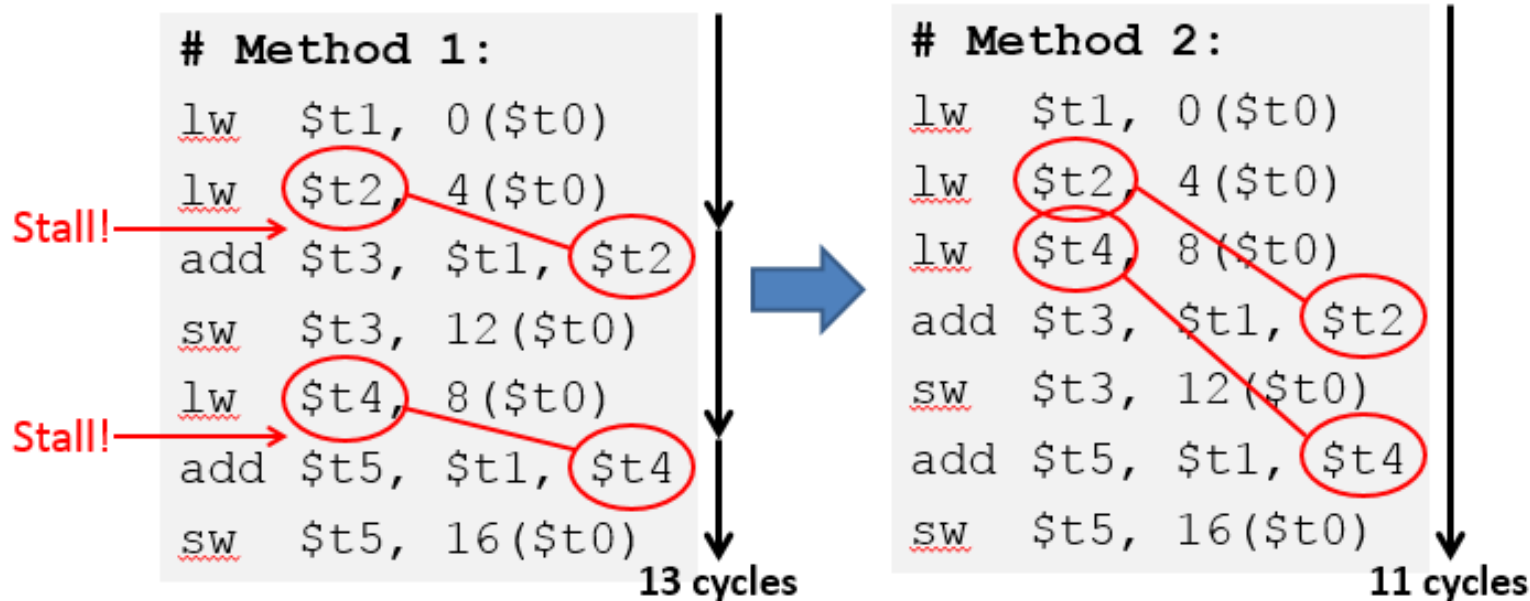
❖ Load导致的数据冒险



load数据冒险处理——编译优化指令调度

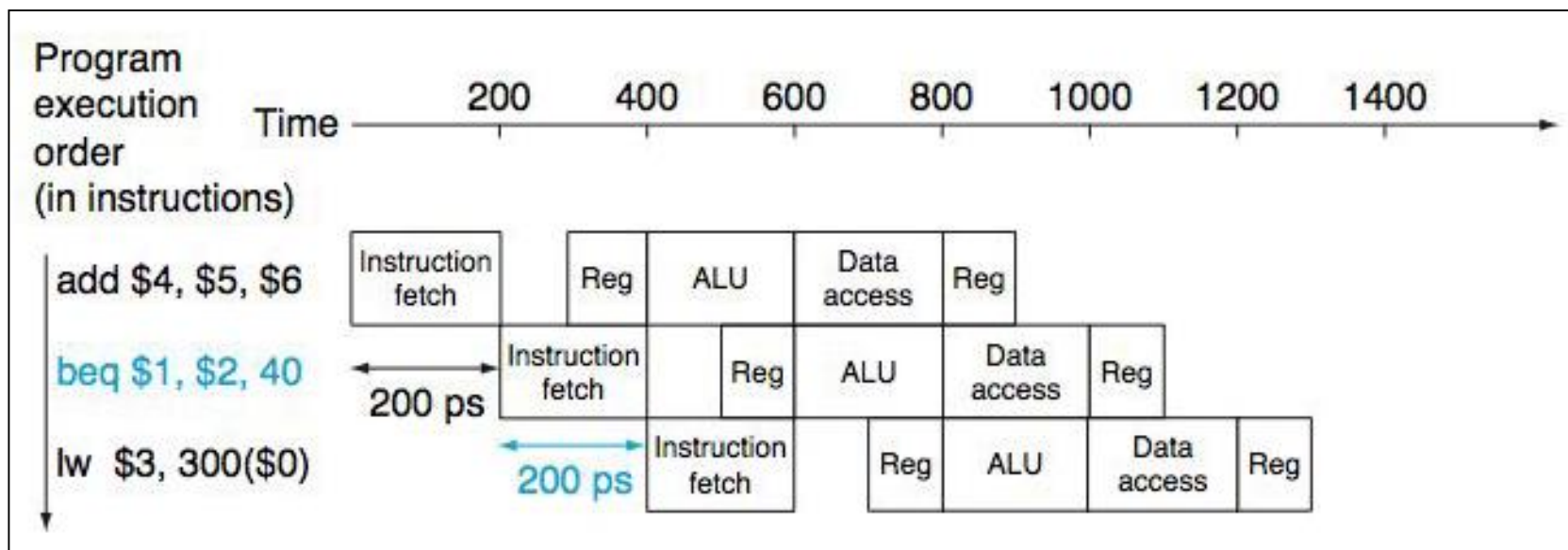
- ❖ Slot after a load instruction is called a *load delay slot*
- ❖ 没有编译优化: putting a **nop** in the **slot**, 需要延迟一个时钟周期
- ❖ 编译指令调度优化: put an **unrelated instruction** in the **slot**, 不需要延迟

- MIPS code for $A=B+E$; $C=B+F$;



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。

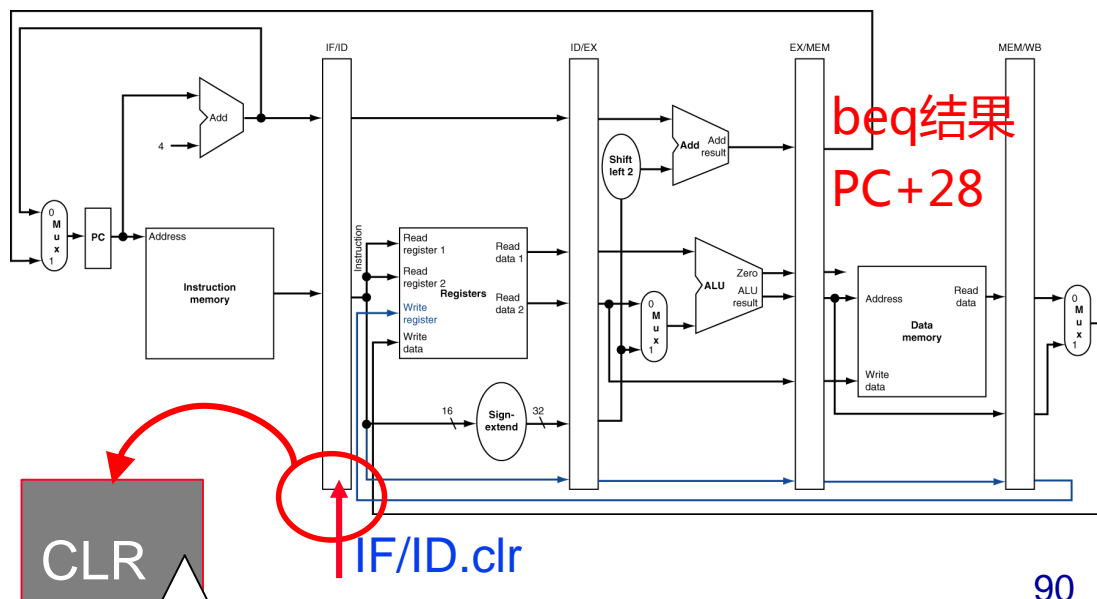


beq \$1, \$2, 40 执行时，可能发生条件转移，而不会执行lw \$3, 300(\$0)

控制冒险——停顿的代价

地址	指令								
		CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	beq \$1, \$3, 24	↑ 1	0 → 4	beq → 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	and → lw	nop	nop	nop		
28	lw \$4, 50(\$7)	↑ 5	28 → 32	lw → XX	lw	nop	nop	nop	nop

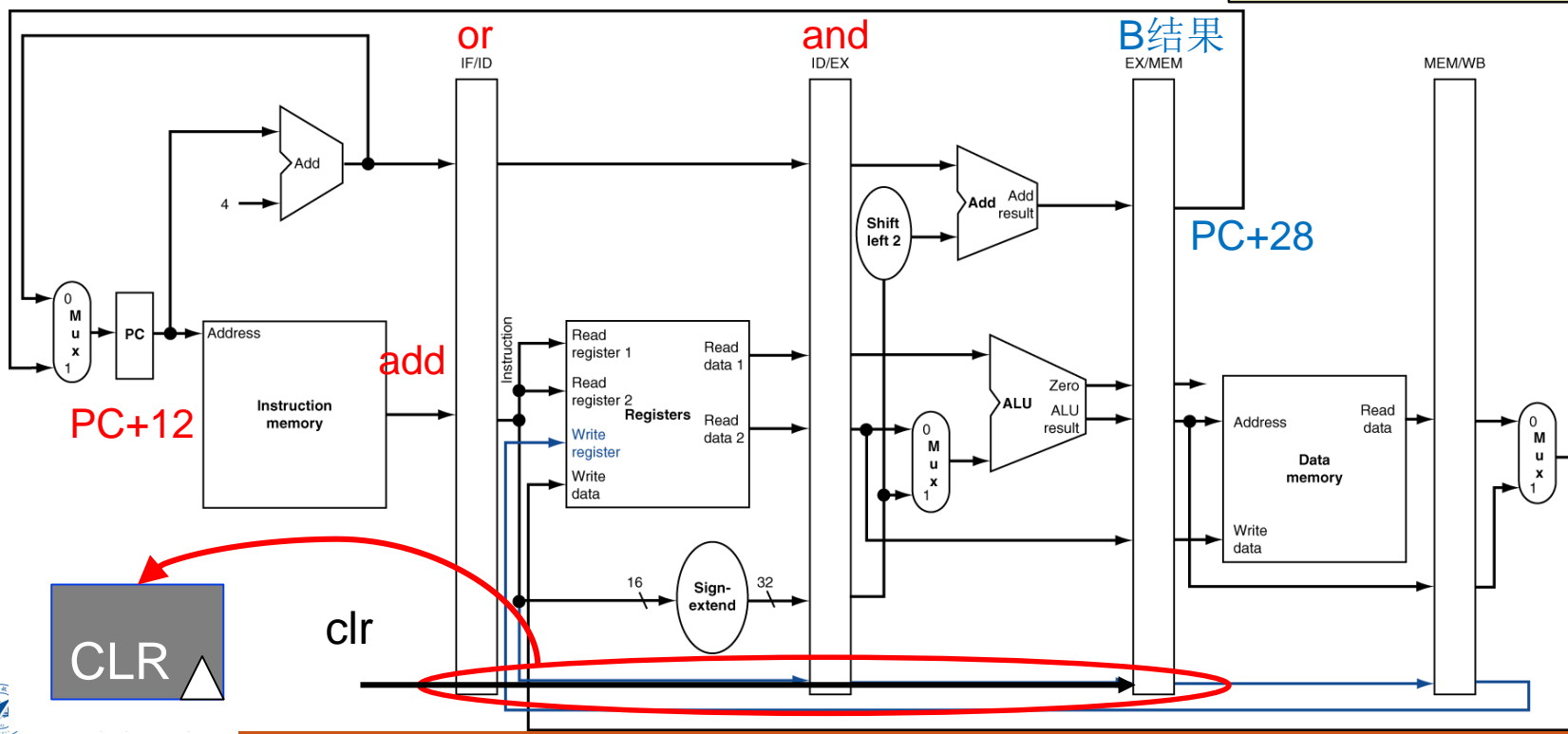
- ◆ 如不对beq指令做任何处理，则必须插入3个NOP
 - beq指令结果及新PC值保存在EX/MEM，因此PC在clk4才能加载正确值
 - IF/ID在clk5才能存入转移后指令(即lw指令)



控制冒险的处理——假定分支不会发生

- ❖ 即使在ID级发现是B指令也不停顿
- ❖ 根据B指令结果，决定是否清除3条后继指令
 - 使得and/or/add不能前进

PC相对偏移	指令
0	beq \$1, \$3, 24
4	and \$12, \$2, \$5
8	or \$13, \$6, \$2
12	add \$14, \$2, \$2
28	lw \$4, 50(\$7)



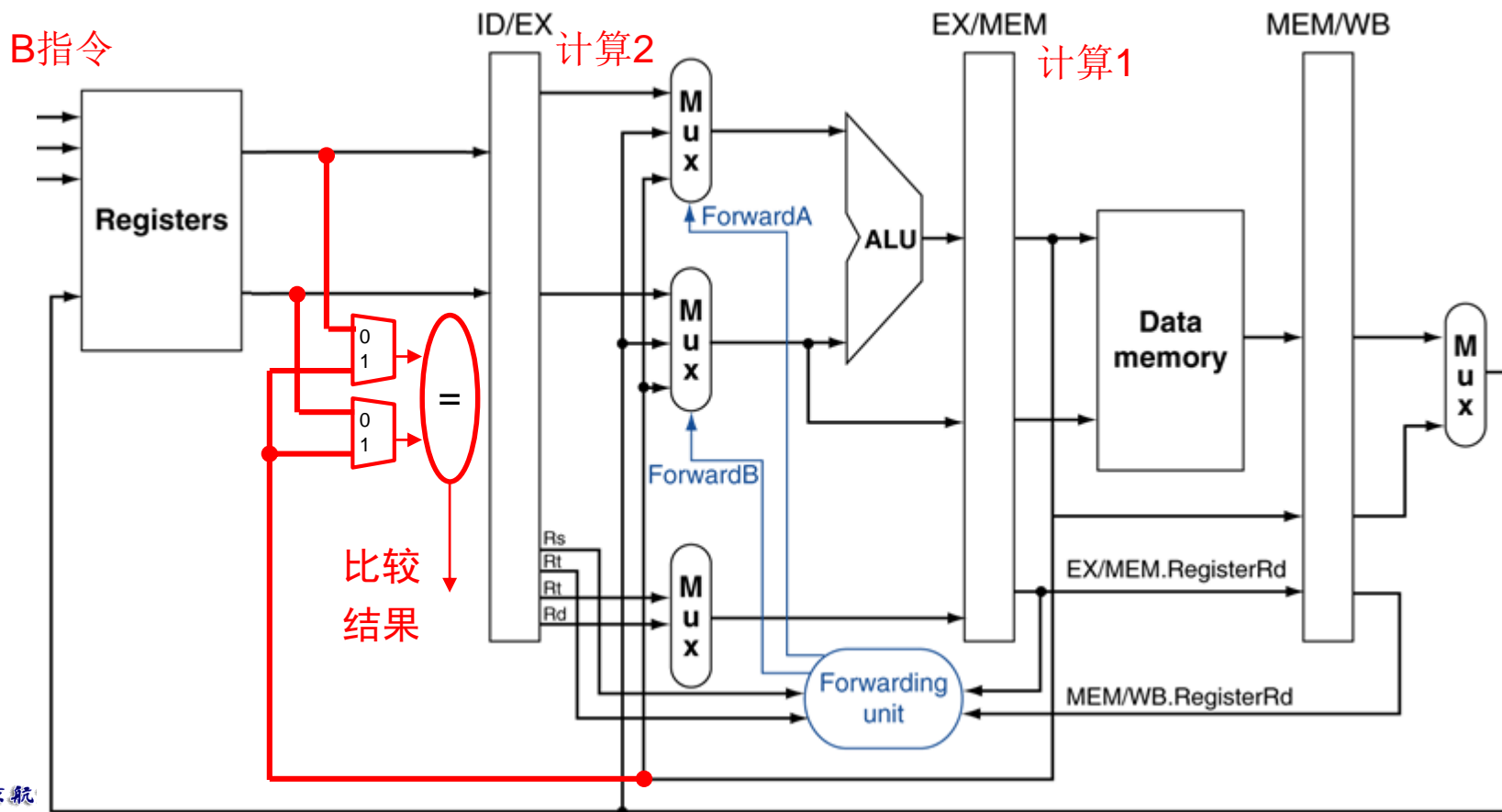
控制冒险的处理——缩短分支延迟

❖ 比较器前置后，会产生数据相关（数据冒险问题）

➤ B指令可能依赖于前条指令的结果

❑ 依赖计算1：从ALU转发数据

❑ 依赖计算2：只能暂停



控制冒险的处理——Branch Delay Slot（分支延迟槽）

Delayed Branch Example

Nondelayed Branch

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

Exit:

Delayed Branch

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

Why not any of the
other instructions?

Exit:

控制冒险的处理——Branch Delay Slot（分支延迟槽）

❖ MIPS uses this *delayed branch* concept

- Re-ordering instructions is a common way to speed up programs
- **Compiler** finds an instruction to put in the branch delay slot

❖ MIPS Green Sheet for **jal**:

$R[31] = PC + 8$; $PC = \text{JumpAddr}$

- $PC + 8$ because of *jump delay slot*!
- Instruction at **$PC + 4$** always gets executed before **jal** jumps to label, so return to $PC + 8$

计算机性能评价

❖ 示例二

对于一台 400MHz 计算机执行标准测试程序，程序中指令类型，执行数量和平均时钟周期数如下：

指令类型	指令执行数量	平均时钟周期数
整数	45000	1
数据传送	75000	2
浮点	8000	4
分支	1500	2

求该计算机的有效 CPI、MIPS 和程序执行时间。

解： $CPI = \sum (IC_i \times CPI_i) / IC$

$$CPI = \frac{45000 \times 1 + 75000 \times 2 + 8000 \times 4 + 1500 \times 2}{45000 + 75000 + 8000 + 1500} = 1.776$$

$$MIPS \text{ 速率} = \frac{f}{CPI} = \frac{400 \times 10^6}{1.776} = 225.225 MIPS$$

程序执行时间=

$$(45000 \times 1 + 75000 \times 2 + 8000 \times 4 + 1500 \times 2) / (400 \times 10^6) = 5.75 \times 10^{-4} s$$

第四部分：关于考试

❖课程成绩：（ $A*85\%+B*15\%$ ）*40%+C*60%

➤卷面成绩（卷面100分）：A

➤平时成绩（作业成绩）：B

➤实验成绩（总分100分）：C

❖ 总体原则：

- 难度相比往年相当
- 有8个实验作为基础，MIPS设计试题弱化

❖ 试题类型

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

❖ 分数分布

- 基本概念：10
- 组合逻辑与时序逻辑：25
- 指令系统与汇编语言：20
- 存储系统（主存+CACHE+虚存）：25
- I/O：5分
- MIPS处理器：15分

