第六讲 MIPS处理器设计

一．处理器设计概述

二．MIPS模型机

三．MIPS单周期处理器设计

四．MIPS多周期处理器设计简介

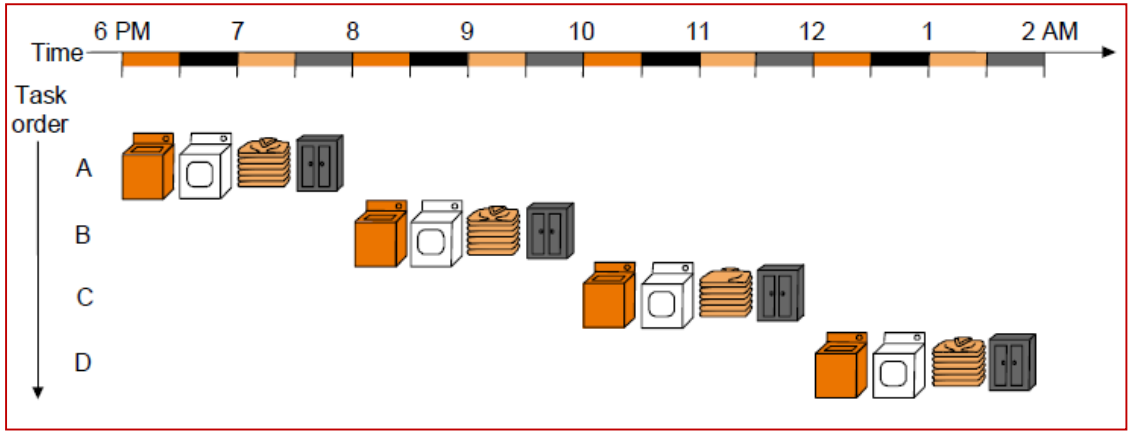
五．MIPS流水线处理器设计

1. 流水线原理
2. MIPS流水线数据通路
3. 流水线冒险

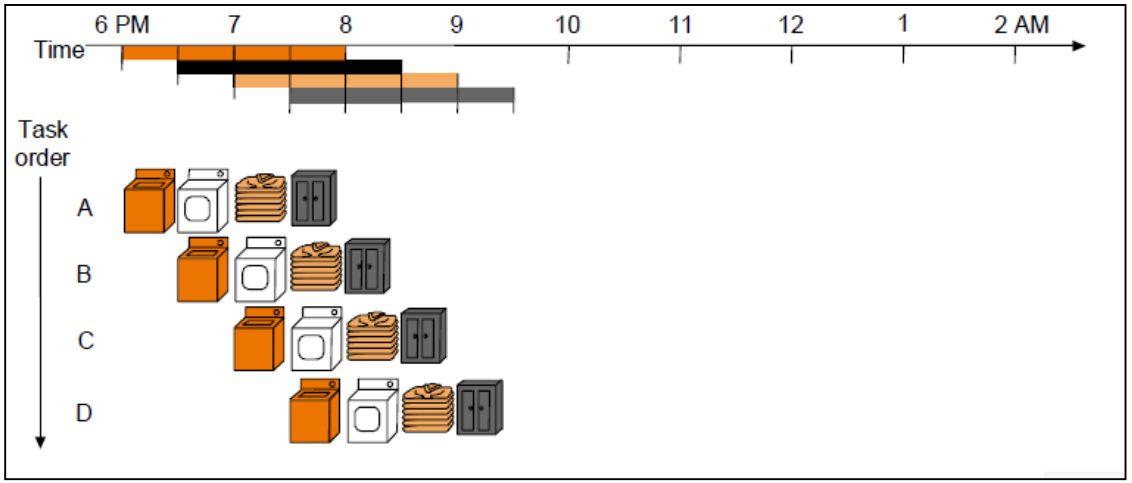
5.1 流水线原理

❖ 洗衣店流程

- 4个步骤，洗衣、烘干、折叠、储存，每步30分钟
- 4个工人（4条指令A、B、C、D）
- **非流水模式，洗4批衣服需8小时**
- **流水模式，洗4批衣服需3.5小时**



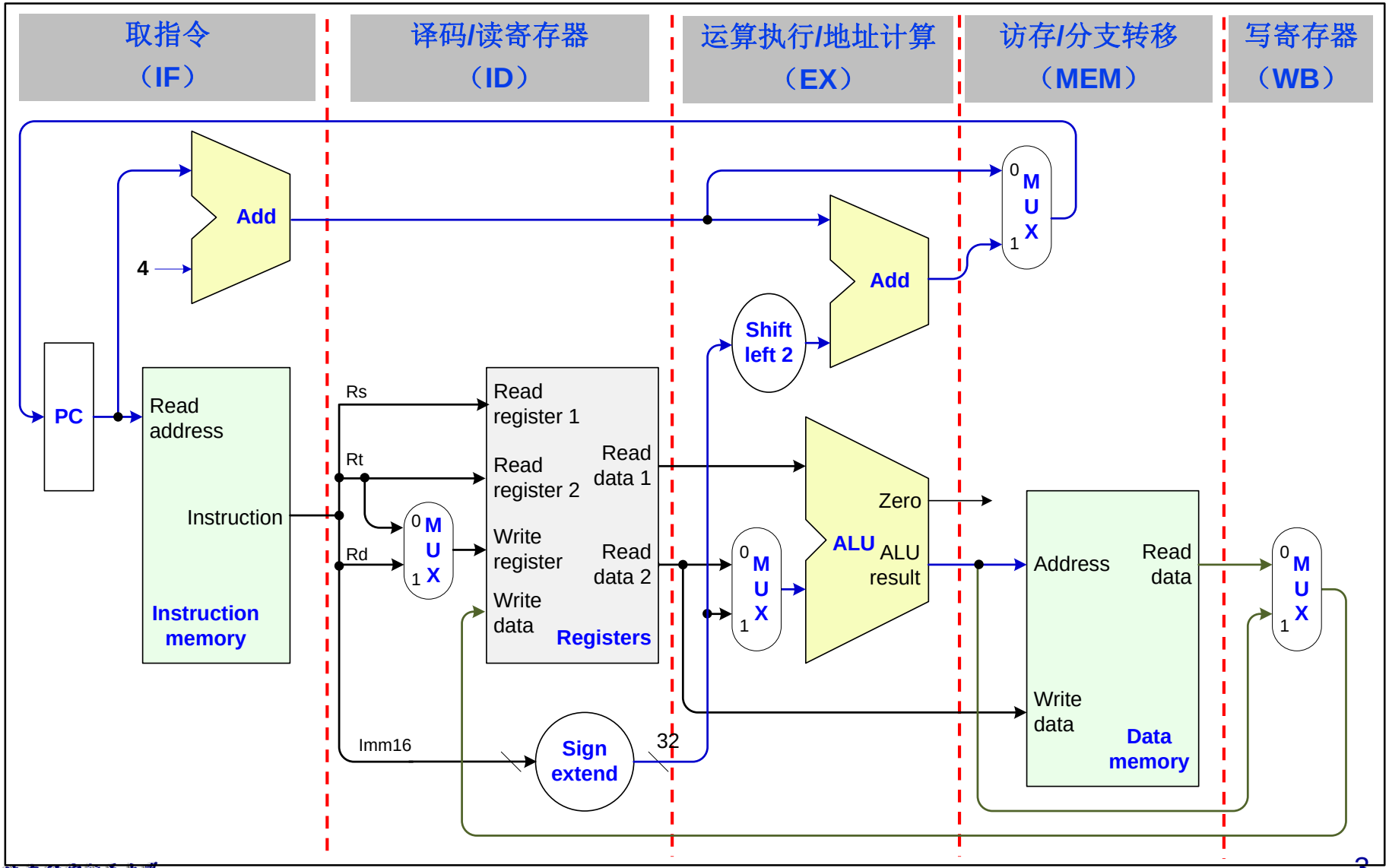
非流水线模式



流水线模式

5.1 流水线原理

❖ MIPS单周期数据通路（简化，省略了控制信号）



5.1 流水线原理

- 指令执行步骤分解
  - 步骤1：取指（*Instruction fetch cycle*--**IF**）
    - 根据PC从指令存储器（IM）中读取指令
  - 步骤2：译码/读寄存器（*Instruction decode/register fetch cycle* --**ID**）
    - 完成指令译码，并读取寄存器组（REG）
  - 步骤3：运算执行/地址计算（*Execution/ address calcution cycle* --**EX**）
    - R型指令，I型运算指令：ALU完成运算操作
    - 访存指令（lw/sw）：ALU计算存储器地址
    - beq指令：Add计算转移地址，ADD的操作可整合到ALU
  - 步骤4：访存/分支转移（*Memory access/branch completion cycle* –**MEM**）
    - 访存指令（lw/sw）：访问（读或写）数据存储器（DM）
    - beq指令：完成分支转移（修改PC）
  - 步骤5：写寄存器（*Write-back cycle* --**WB**）
    - R型指令，I型运算指令：运算结果写寄存器（REG）
    - lw指令：存储器操作数写寄存器（REG）

5.1 流水线原理

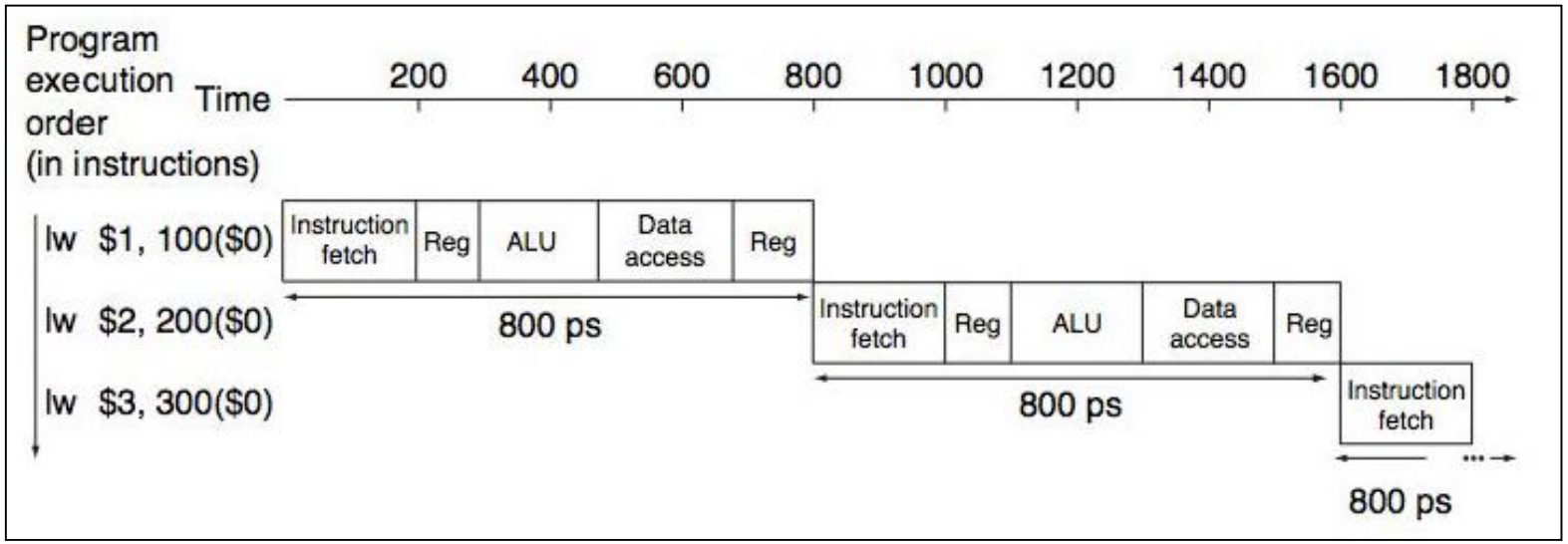
- 假定每步操作及所花时间为：
  - 步骤1：取指（IF）——200ps
  - 步骤2：译码和读寄存器（ID）——100ps
  - 步骤3：运算执行与存储器地址计算（EX）——200ps
  - 步骤4：访问内存（MEM）——200ps
  - 步骤5：写寄存器——100ps

不同类型指令的执行时间

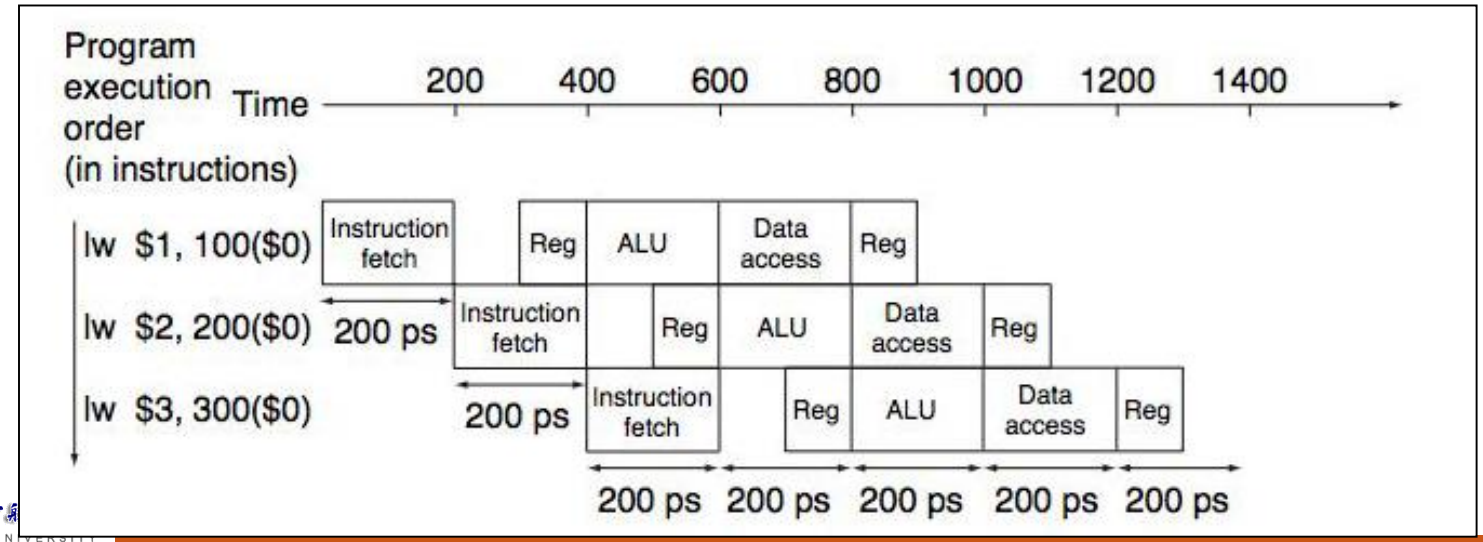
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

5.1 流水线原理

- 单周期模型



- 流水线模型（5个步骤，每个步骤200ps）



5.1 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 ≈ 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

第六讲 MIPS处理器设计

一．处理器设计概述

二．MIPS模型机

三．MIPS单周期处理器设计

四．MIPS多周期处理器设计简介

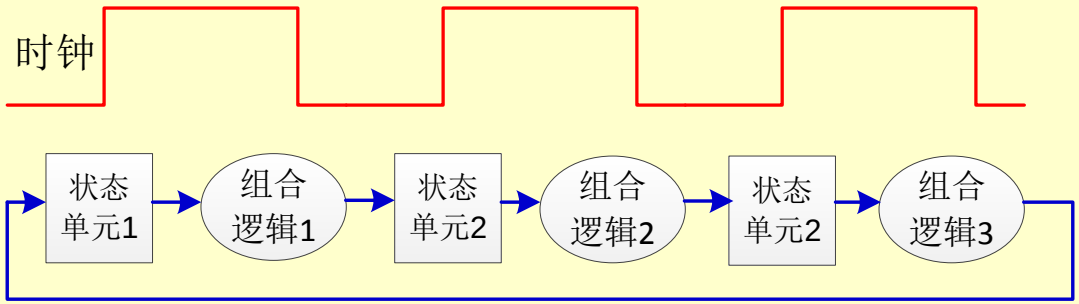
五．MIPS流水线处理器设计

1. 流水线原理
2. MIPS流水线数据通路
3. 流水线冒险

5.2 MIPS流水线数据通路

❖流水线数据通路总体考虑

- 指令执行分5个步骤，每个步骤执行周期为一个时钟周期。
- 流水线数据通路分5个流水段（5级流水线）
- 指令每一步（每个流水段）执行所需要的数据全部来自前一个状态单元的输出，经过本段功能单元处理后，在下一个时钟周期触发沿将本段处理结果（包括本段未作任何处理但是后续步骤所需数据）全部写入下一个状态单元，因此数据通路中需要增加多个寄存器以保存指令各段处理结果，以便在后续流水段周期内继续使用。这样的寄存器称为流水线寄存器。



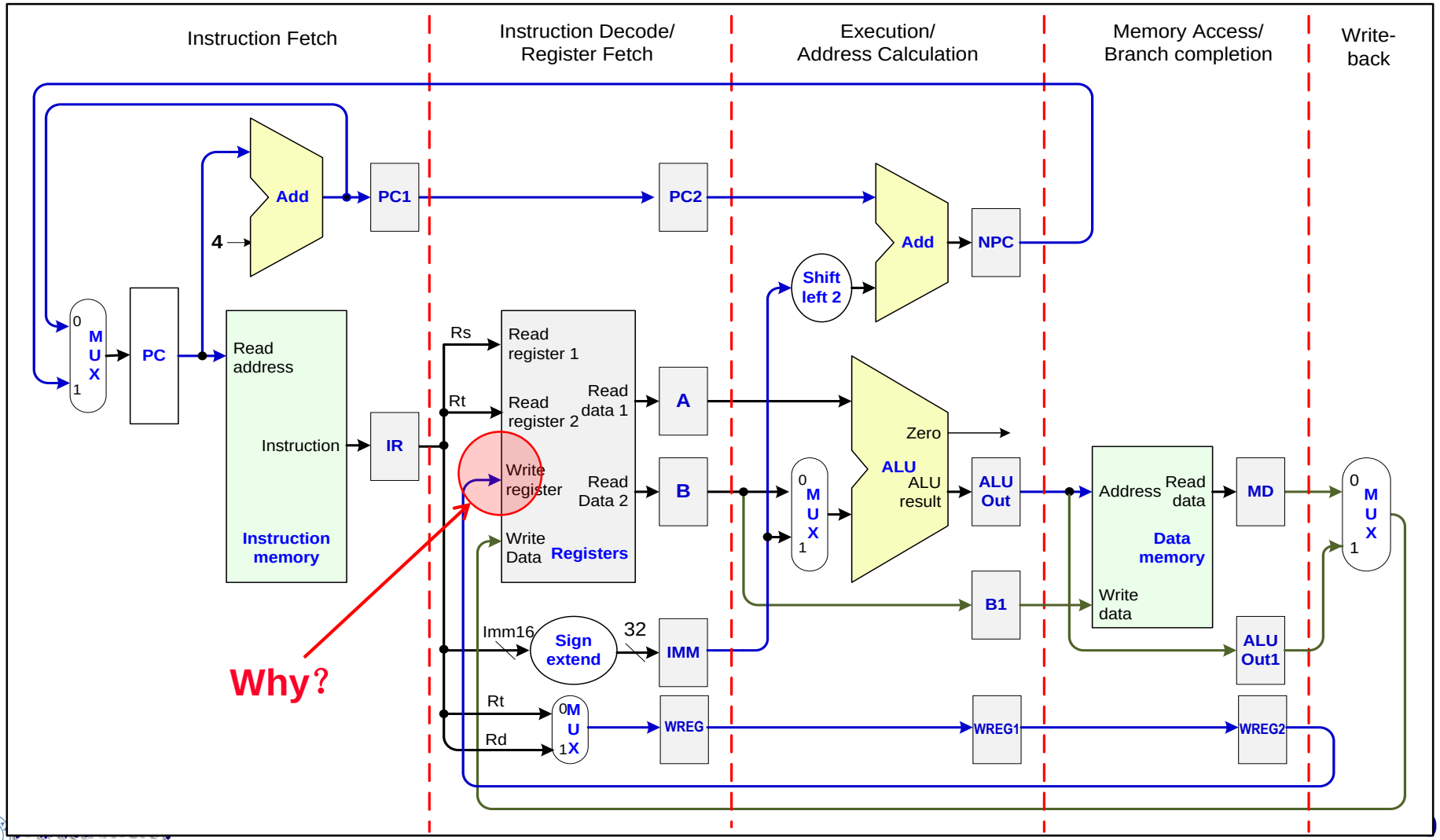
流水线通路同步

- 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

时钟同步方法

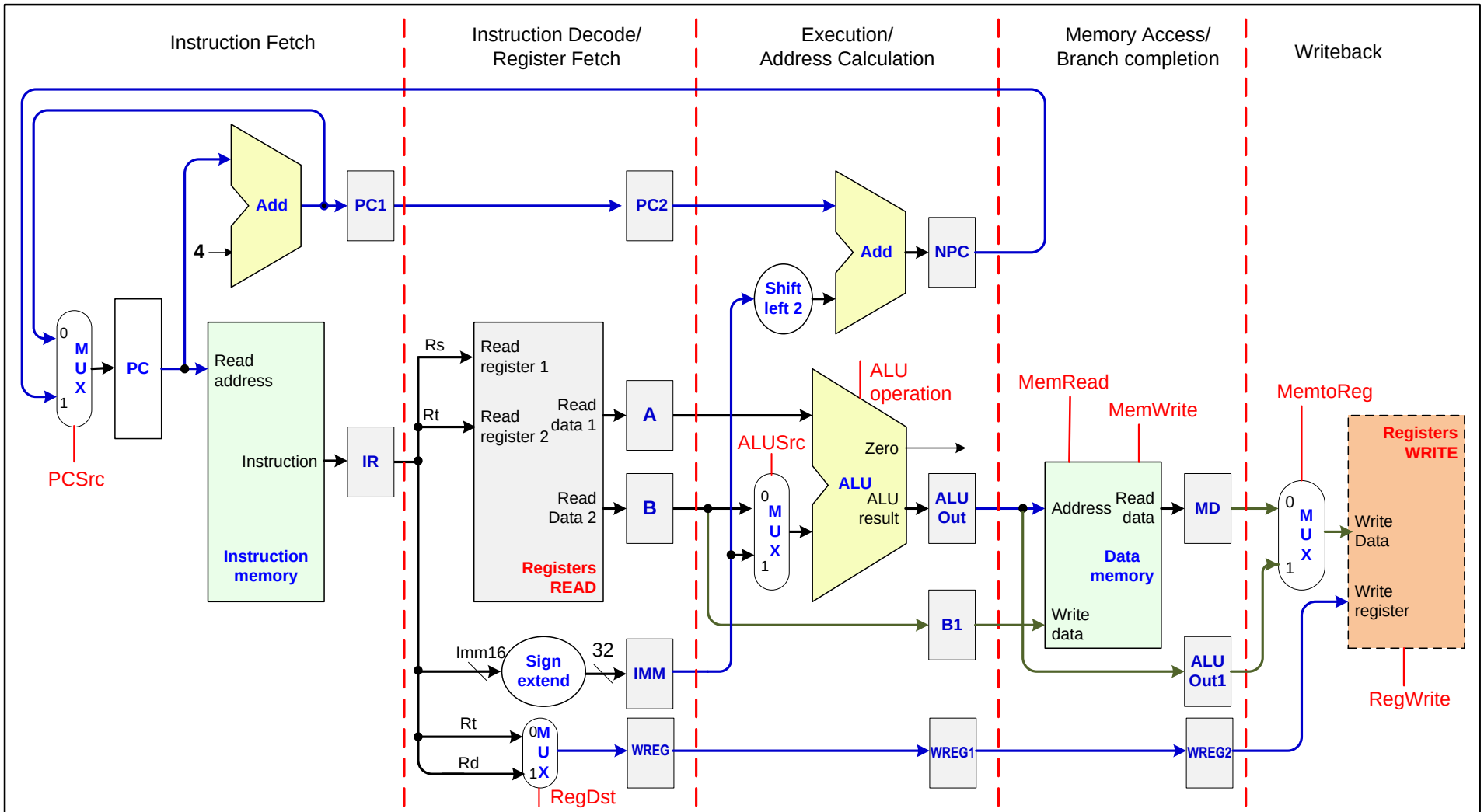
5.2 MIPS流水线数据通路

- MIPS流水线数据通路（每一步的结果均保存到寄存器）
  - 增加寄存器（PC1、IR、A、B、IMM、WREG、ALUOut、NPC、MD等）



5.2 MIPS流水线数据通路

❖ **MIPS流水线数据通路（换一种角度：将RF读端口与写端口分开理解）**



5.2 MIPS流水线数据通路

❖ MIPS流水线模型指令执行步骤分解

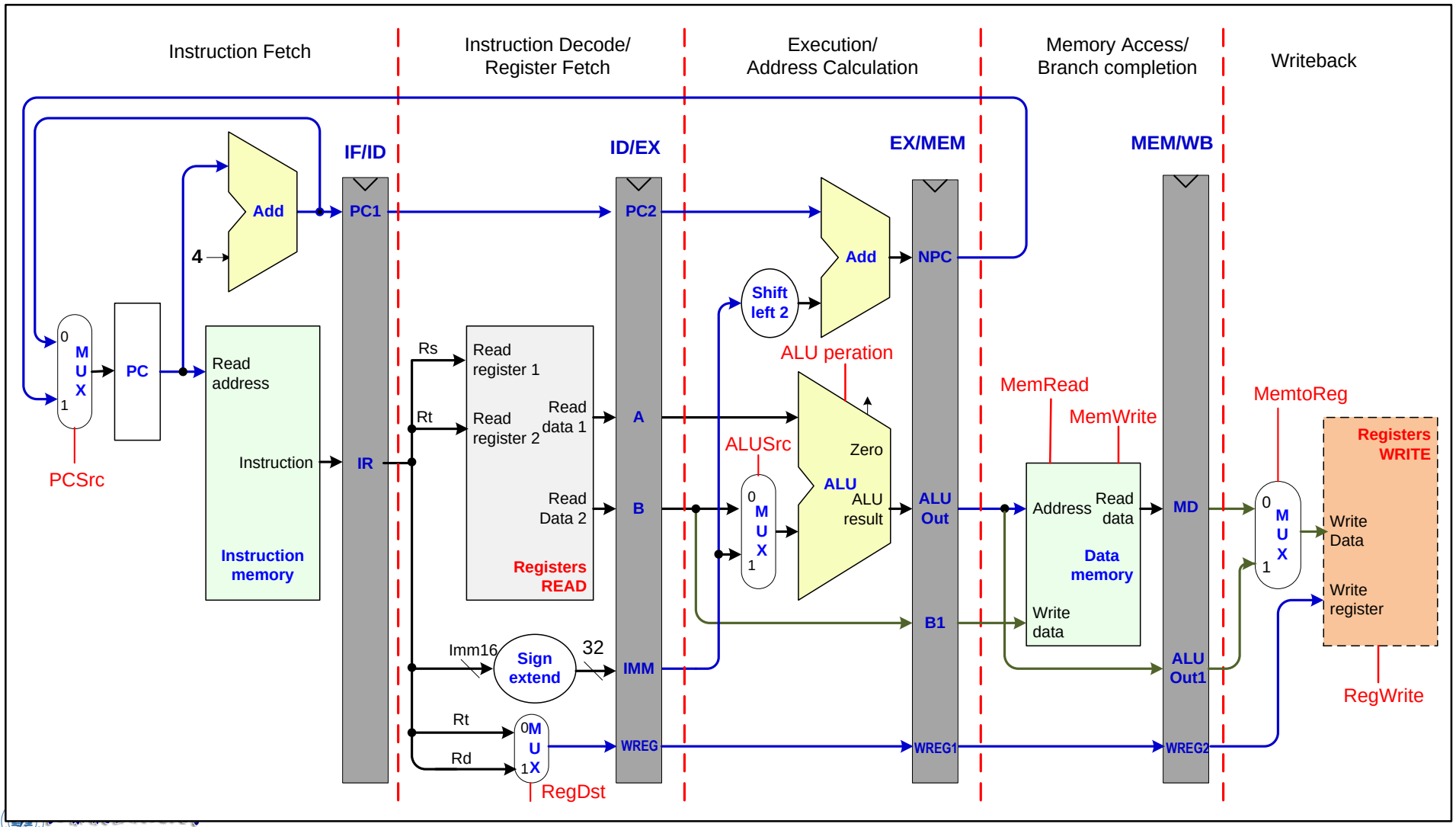
- ➢ 步骤**1**：取指（ *Instruction fetch* --**IF** ）
  - IR ← IMem[PC]
  - PC ← PC +4（PCSrc=0）
  - **PC1← PC + 4**
- ➢ 步骤**2**：译码/读寄存器（ *Instruction decode/register fetch* --**ID** ）
  - A ← Reg[IR[25:21]（Rs）
  - B ← Reg[IR[20:16]（Rt）
  - IMM ← SgnExt(IR[15:0])，**PC2 ← PC1**
  - **lw**指令：　　WREG ← Reg[IR[20:16]（Rt，RegDst=0）
  - **R**类型指令：　WREG ← Reg[IR[15:11]（Rd ，RegDst=1）
- ➢ 步骤**3**：运算执行/地址计算（ *Execution/ address calculation* -- **EX**）
  - **R**型指令：　　ALUOut ← A *func* B（ALUSrc=0）
  - **lw/sw**指令：　ALUOut ← A + IMM（ALUSrc=1）
  - **beq**指令：　　ALUOut ← A - B（ALUSrc=0）
  - NPC ← PC2 + IMM<<2，**B1 ← B，WREG1 ← WREG**

5.2 MIPS流水线数据通路

- 流水线模型指令执行步骤分解（续）
  - 步骤4：访存/分支转移（*Memory access/branch completion* -- **MEM**）
    - **lw**指令：MD ← DMem[ALUOut]（MemRead=1）
    - **sw**指令：DMem[ALUOut] ← B1（MemWrite=1）
    - **beq**指令：if (cond) PC ← NPC（PCSrc=1）
    - **ALUOut1 ← ALUOut，WREG2 ← WREG1**
  - 步骤5：写寄存器（*Write-back* --**WB**）
    - **R**型指令：Reg[WREG2] ← ALUOut1（MemtoReg=1,RegWrite=1）
    - **lw**指令：Reg[WREG2] ← MD（MemtoReg=0,RegWrite=1）

5.2 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

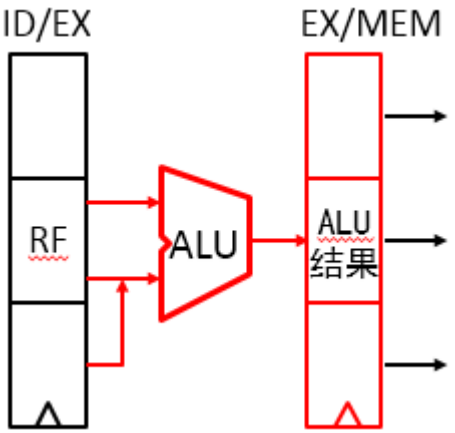


正确认识流水线寄存器

- 命名法则：前级/后级
  - 示例：IF/ID，前级为读取指令，后级为指令译码(及读操作数)
- 流水线寄存器功能：时钟上升沿到来时，保存前级结果；之后输出至下级组合逻辑
  - 也可能直接连接到下级流水线寄存器
    - 例如ID/EX保存的从RF读出的寄存器Rt的值（B），就直接传递到EX/MEM
- N级流水线：必须有N级流水线寄存器
  - 插入N-1级流水线寄存器，最后一级为Register File
- **RF的特殊性**：在流水线通路中使用2次
  - 流水线第2级：读寄存器（组合元件操作）
  - 流水线第5级：写寄存器（状态元件操作）
- **数据存储器（DM）**：读操作是组合元件操作，写操作是状态元件操作（等价于寄存器）

流水阶段的处理

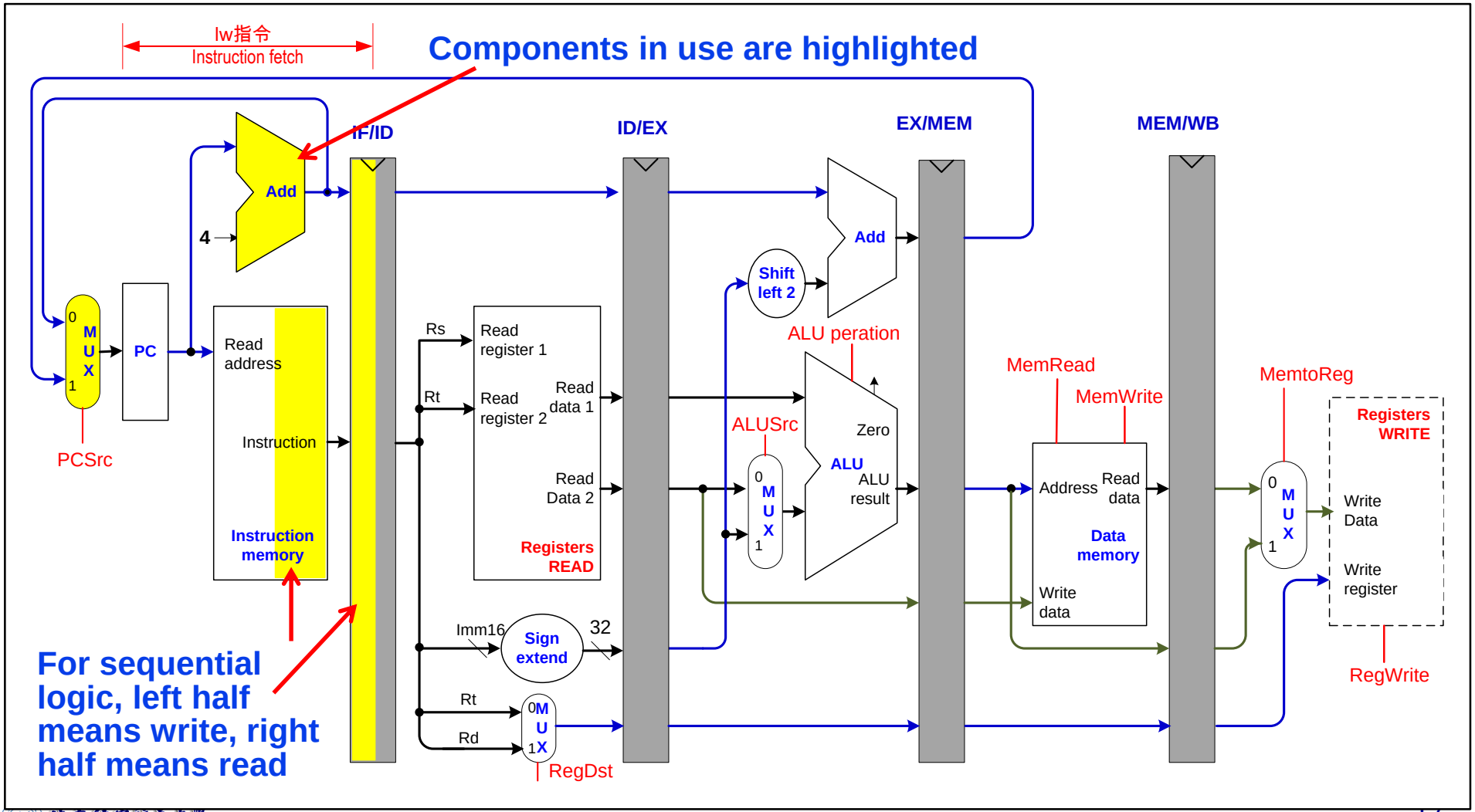
- 组合逻辑+流水寄存器
  - 起始：前级流水线寄存器的输出
  - 中间：组合逻辑（如ALU）
  - 结束：写入后级流水线寄存器
  - 当时钟上升沿到来时，组合逻辑计算结果存入后级寄存器
- 示例：EX阶段
  - 起始：ID/EX流水线寄存器中的RF寄存器/扩展单元的输出
  - 中间(组合逻辑)：ALU完成计算
  - 结束(寄存器)：在clock上升沿到来时，结果写入EX/MEM中相应寄存器



5.2 MIPS流水线数据通路（lw指令数据通路示例）

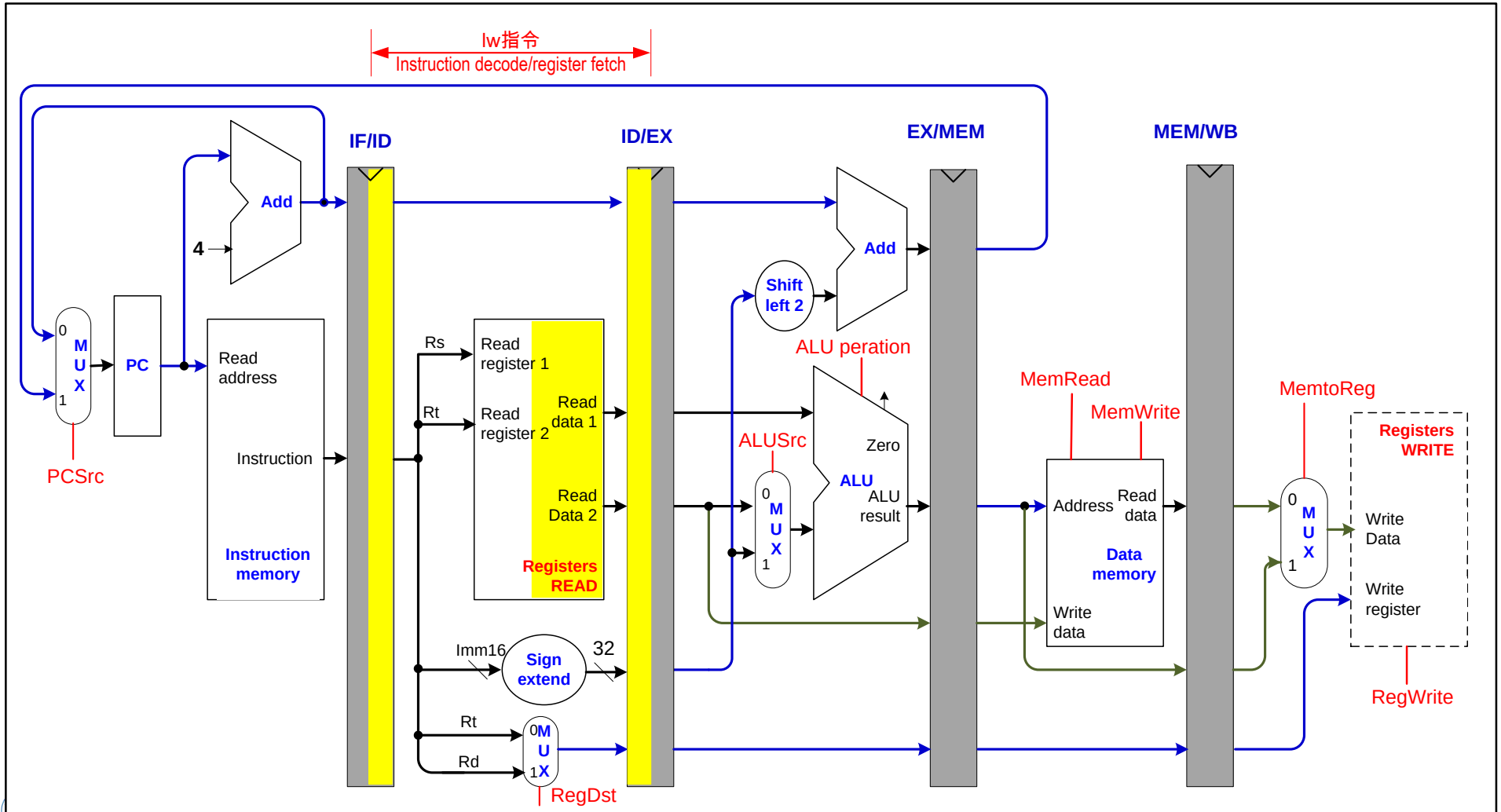
❖ 步骤1：取指（ *Instruction fetch* --**IF** ）

➢ IF/ID(IR) ← IM[PC] ， IF/ID(PC)← PC + 4， PC ← PC +4



5.2 MIPS流水线数据通路（lw指令数据通路示例）

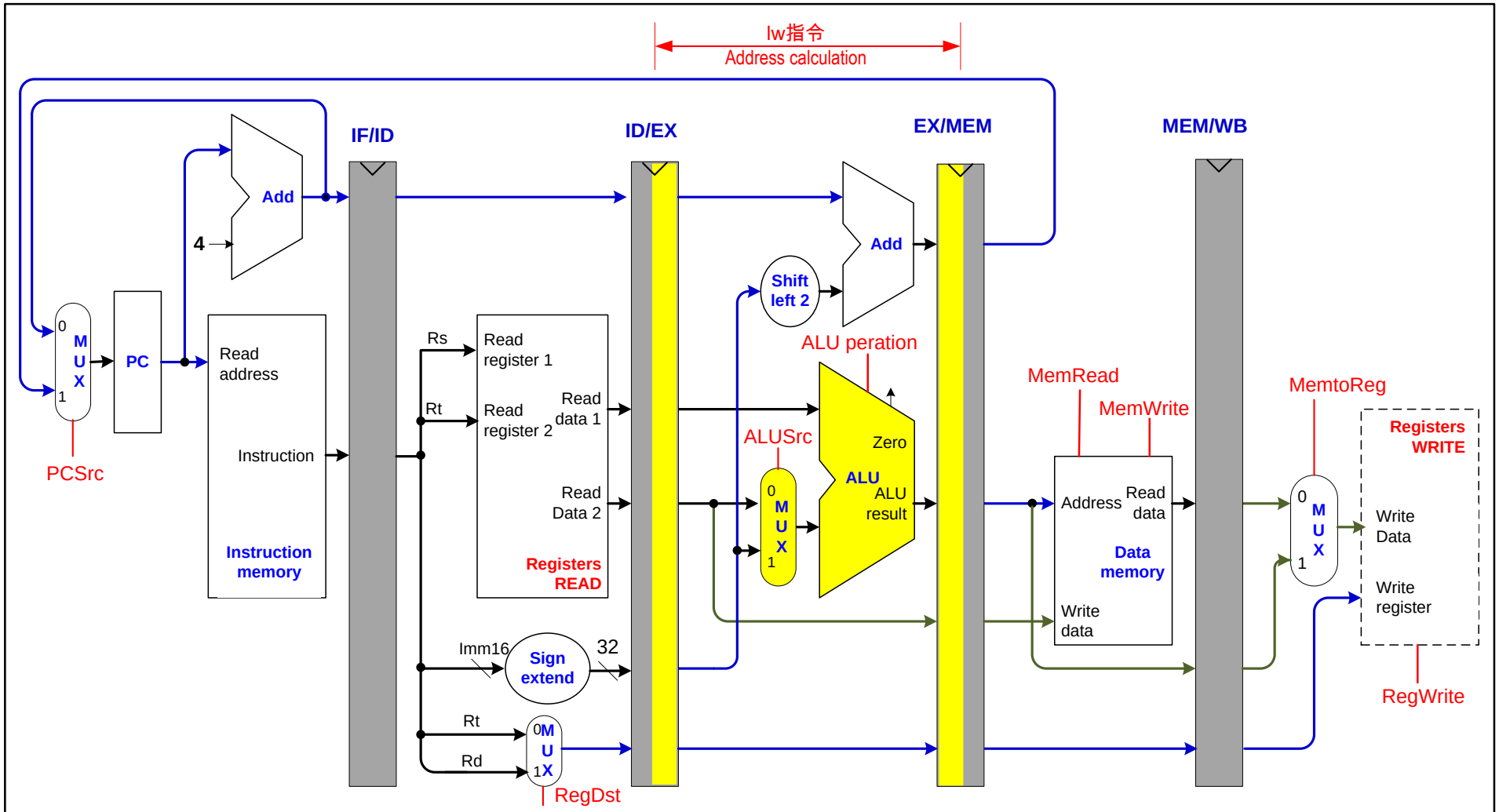
- 步骤2：译码/读寄存器（ *Instruction decode/register* --**ID** ）
  - ID/EX(A) ← Reg[Rs]，ID/EX(B) ← Reg[Rt], ID/EX(IMM) ← SgnExt(Imm16)
  - ID/EX(PC) ← IF/ID(PC)，ID/EX(WREG) ← Rt | Rd



5.2 MIPS流水线数据通路（lw指令数据通路示例）

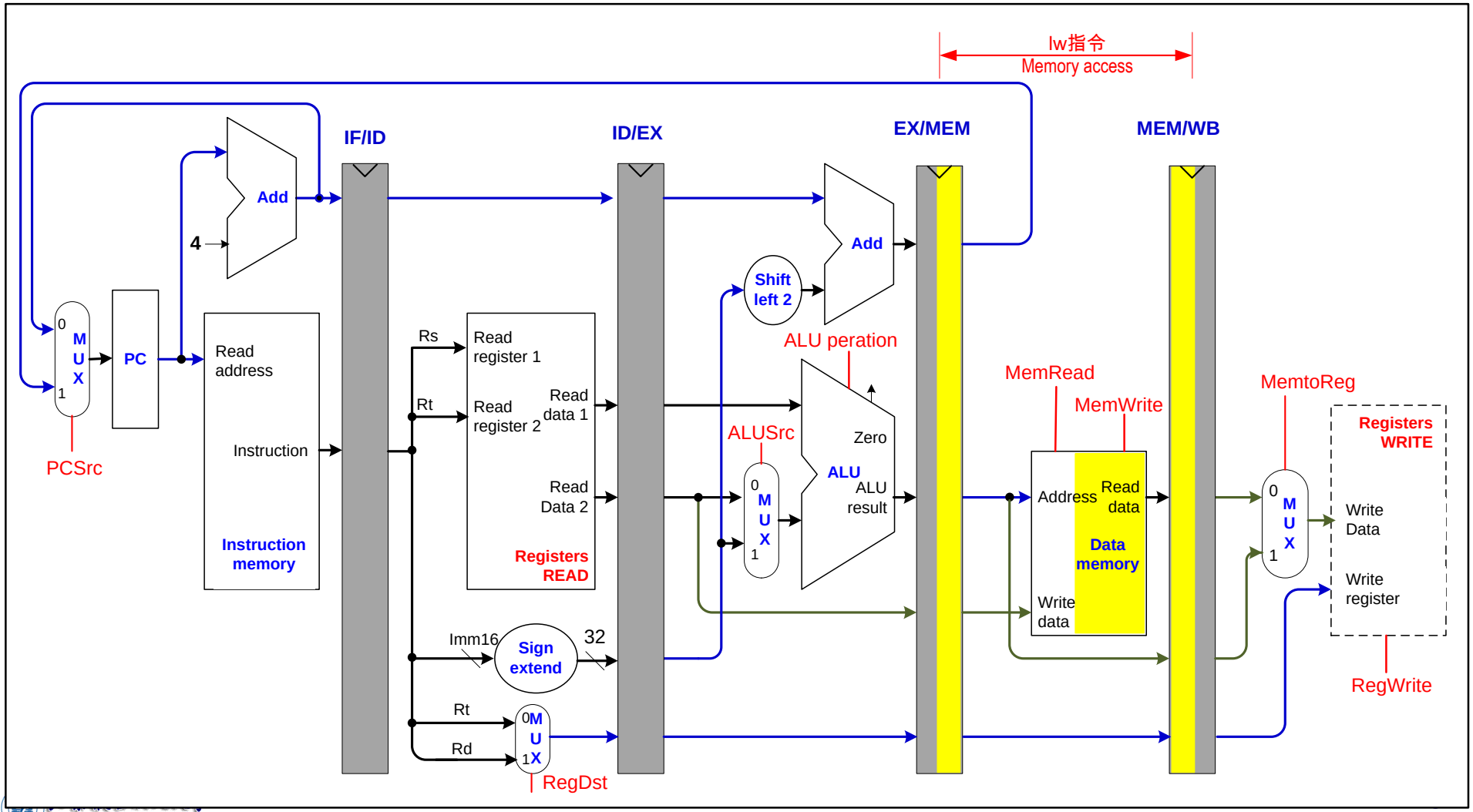
❖ 步骤3：运算执行/地址计算（ *Address calculation* -- **EX**）

➢ EX/MEM(ALU) ← ID/EX(A) + ID/EX(IMM)



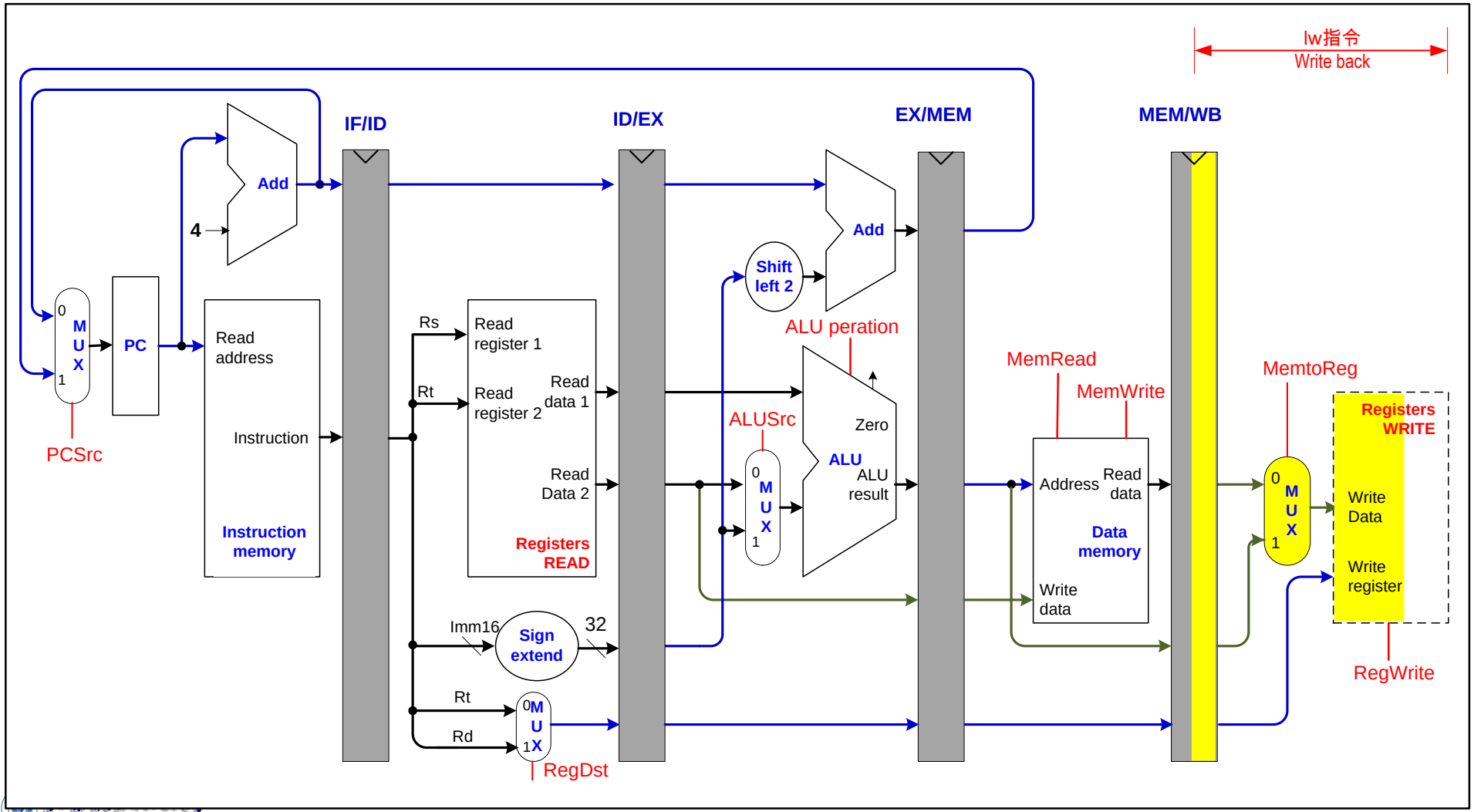
5.2 MIPS流水线数据通路（lw指令数据通路示例）

- 步骤4：访存/分支转移（*Memory access* -- **MEM**）
  - MEM/WB(MD) ← DM(EX/MEM(ALU))



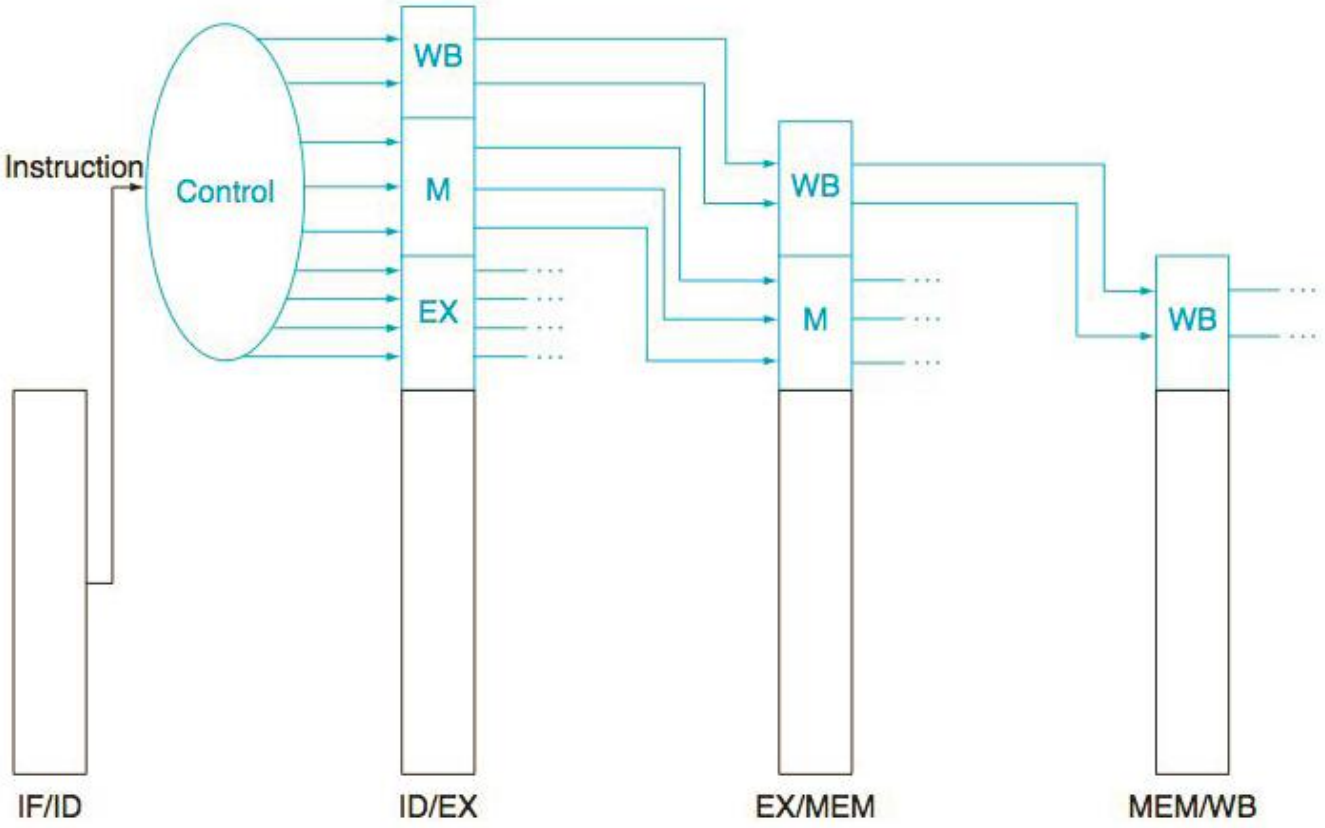
5.2 MIPS流水线数据通路（lw指令数据通路示例）

- 步骤5： 写寄存器（*Write-back* -- **WB**）
  - Reg[MEM/WB(WREG)] ← MEM/WB(MD)



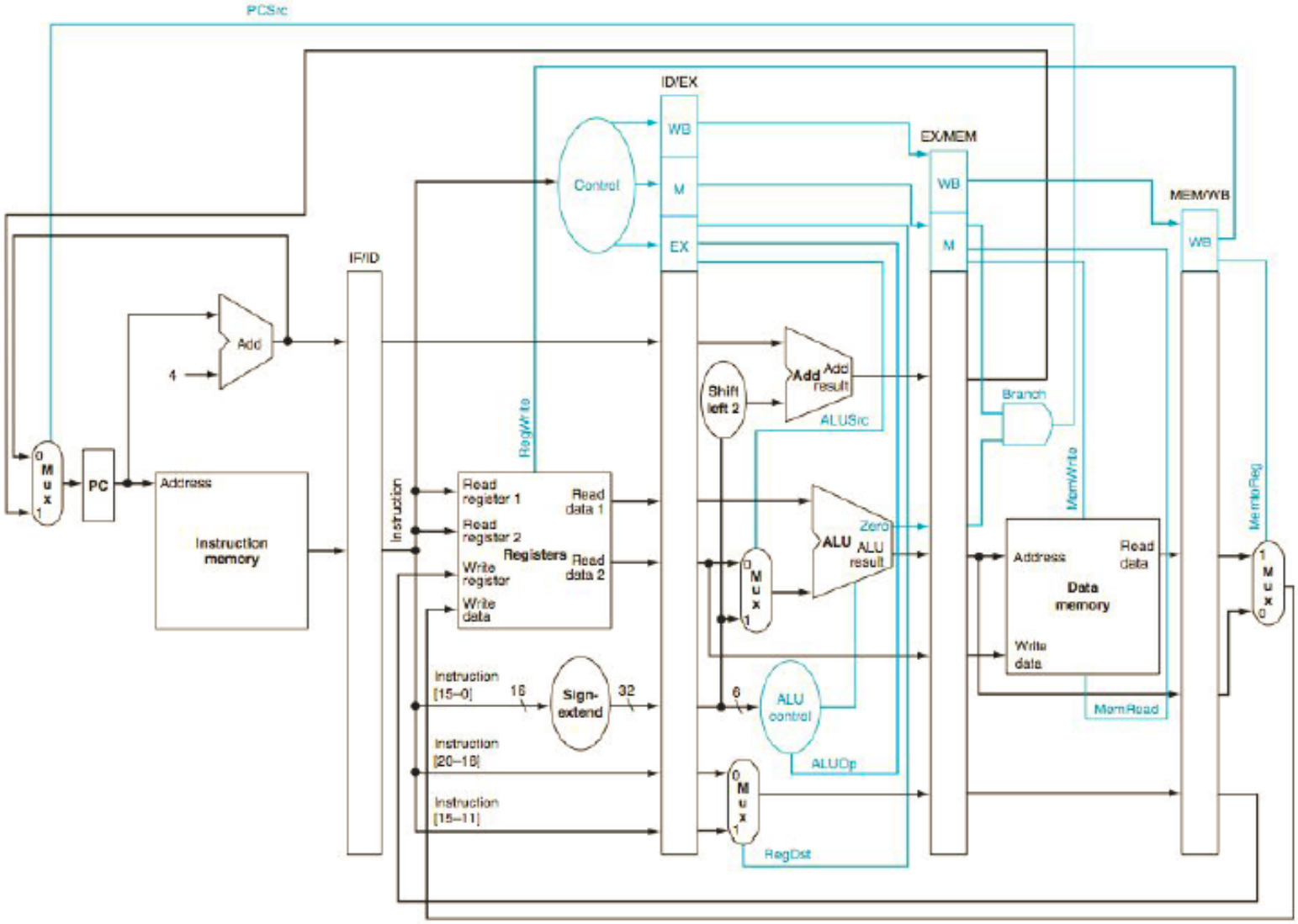
5.2 MIPS流水线数据通路——控制信号

- 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



5.2 MIPS流水线数据通路——控制信号

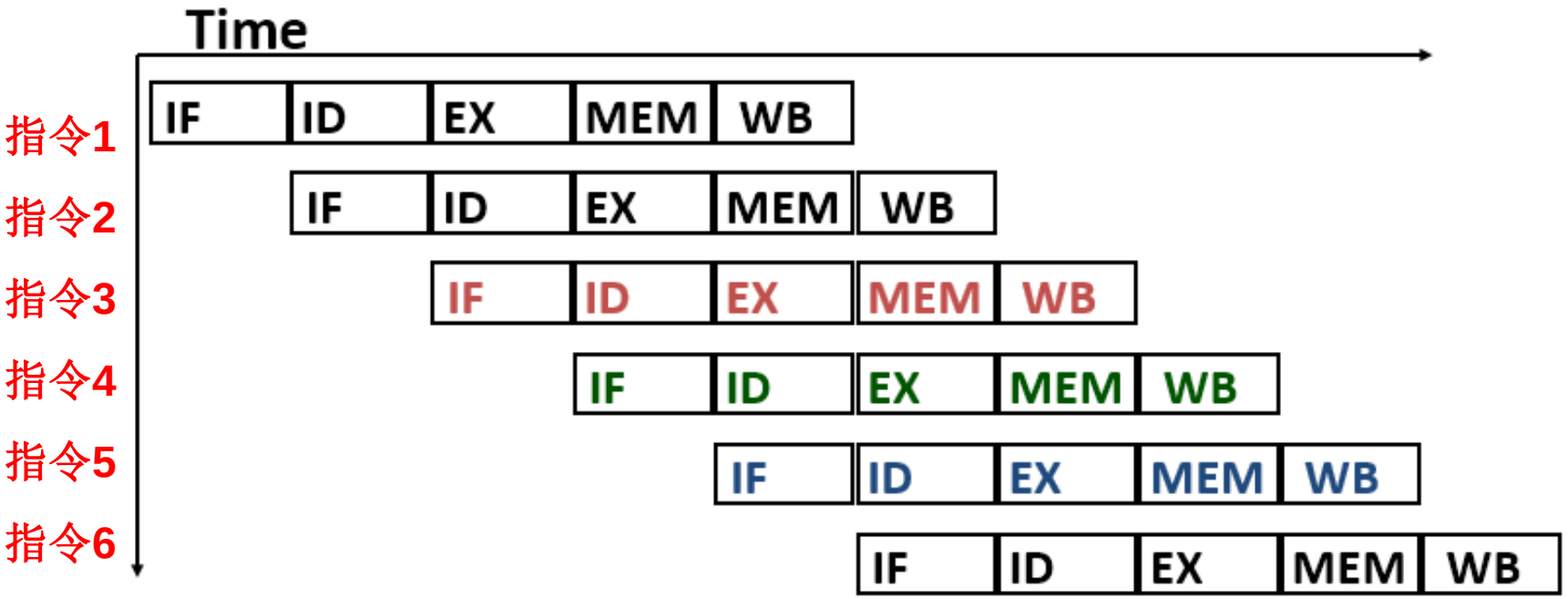
- 控制信号流水寄存器：控制信号在寄存器中传递，直至不再需要



5.2 MIPS流水线数据通路

❖指令流水图表示

- 每条指令都必须执行相同的步骤，
- 某些指令的某个步骤可能并不需要（空操作），如算术运算指令（R类型）的MEM步骤。



5.2 MIPS流水线数据通路

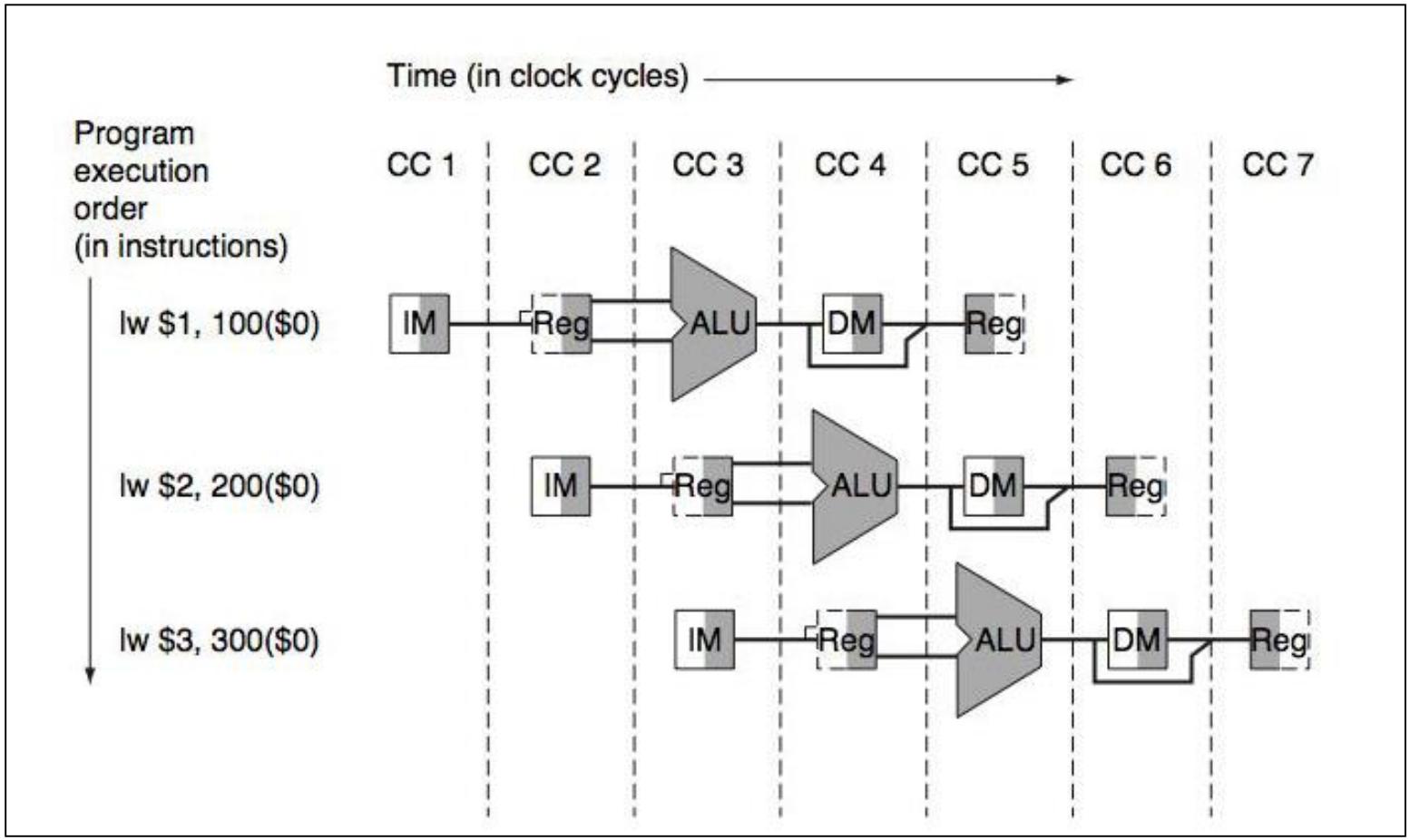
❖时钟驱动的流水线时空图：精确分析指令/时间/流水线3者关系

- 行：某个时钟，指令流分别处于哪些阶段
- 列：某个部件，在时间方向上的执行了哪些指令
- 注意区分流水阶段与流水线寄存器的关系
- 在clk5后，流水线全部充满，所有部件都在执行指令，只是不同的指令

相对PC的地址偏移	指令			IF级	ID/RF级	EX级	MEM级	WB级	
		CLK	PC	IM	IF/ID	ID/EX	**EX/MEM**	MEM/WB	RF
0	Instr 1	⮥ 1	0→4	Instr 1	Instr 1				
4	Instr 2	⮥ 2	4→8	Instr 2	Instr 2	Instr 1			
8	Instr 3	⮥ 3	8→12	Instr 3	Instr 3	Instr 2	Instr 1		
12	Instr 4	⮥ 4	12→16	Instr 4	Instr 4	Instr 3	Instr 2	Instr1	
16	Instr 5	⮥ 5	16→20	Instr 5	Instr 5	Instr 4	Instr 3	Instr 2	Instr1
20	Instr 6	⮥ 6	20→24	Instr6	Instr6	Instr5	Instr4	Instr3	Instr2

5.2 MIPS流水线数据通路

❖ 流水线模型表示



- 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。

第六讲 MIPS处理器设计

一．处理器设计概述

二．MIPS模型机

三．MIPS单周期处理器设计

四．MIPS多周期处理器设计简介

五．MIPS流水线处理器设计

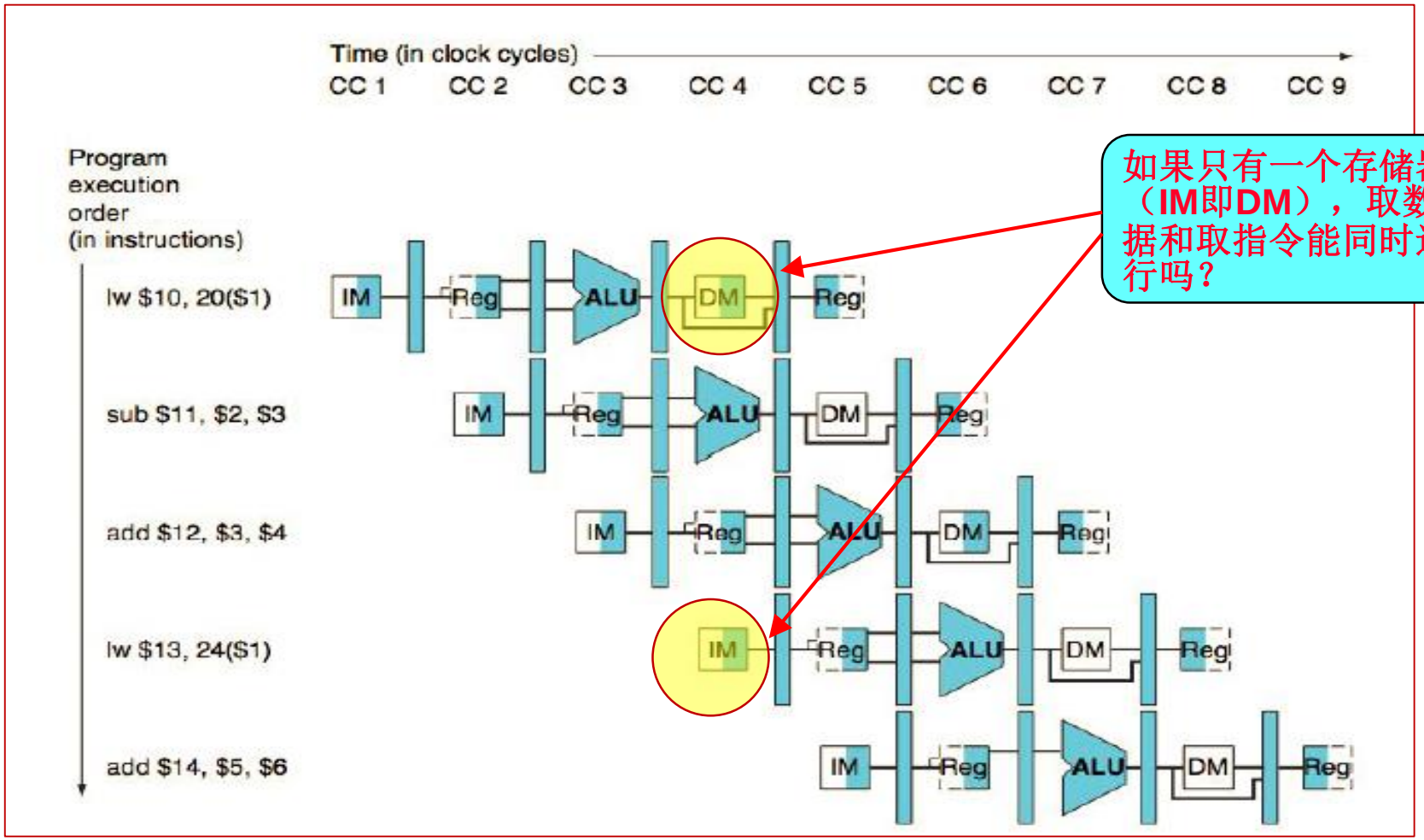
1. 流水线原理
2. MIPS流水线数据通路
3. 流水线冒险

5.3 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。**

5.3 流水线的冒险——结构冒险

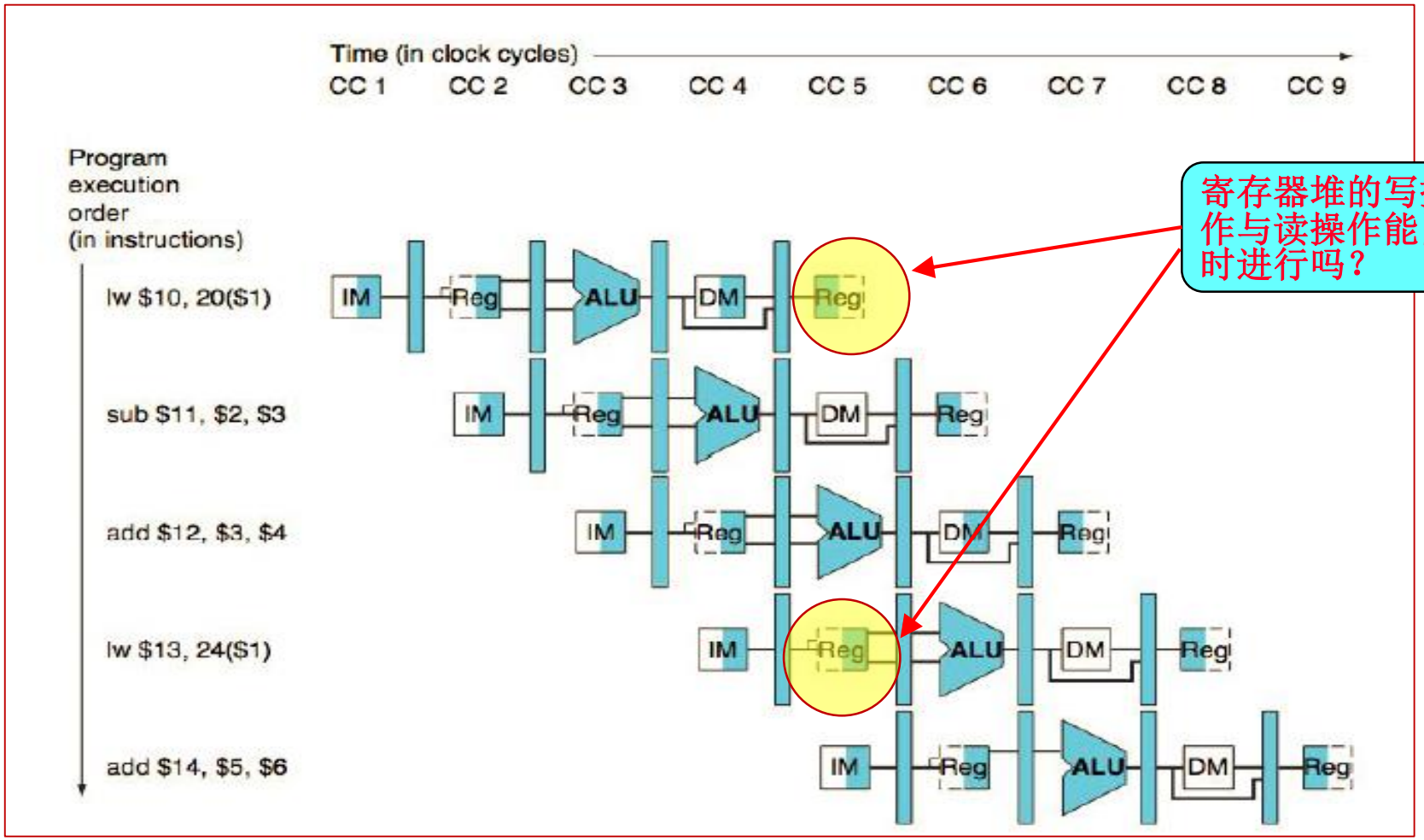
❖ 存储器同时访问问题



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！

5.3 流水线的冒险——结构冒险

❖ 寄存器堆同时访问问题



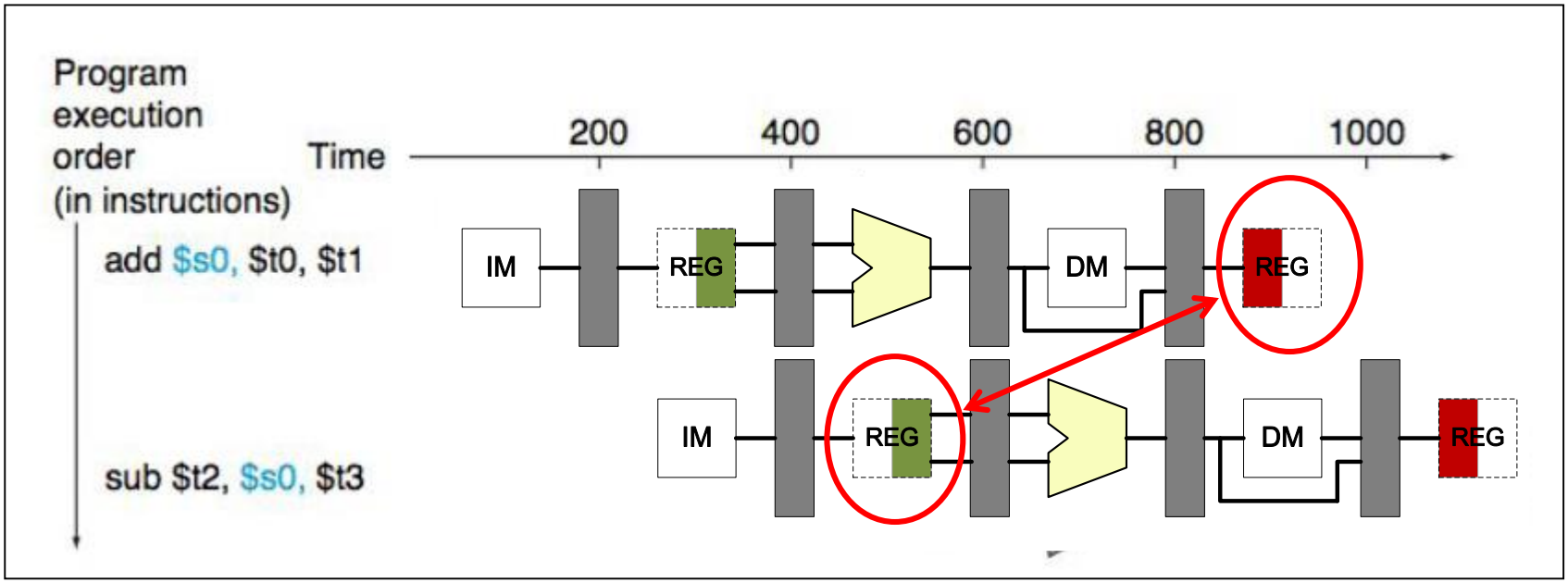
寄存器堆的读操作与写操作可以同时进行的。

5.3 流水线的冒险——数据冒险

❖ 数据冒险（Data hazard）

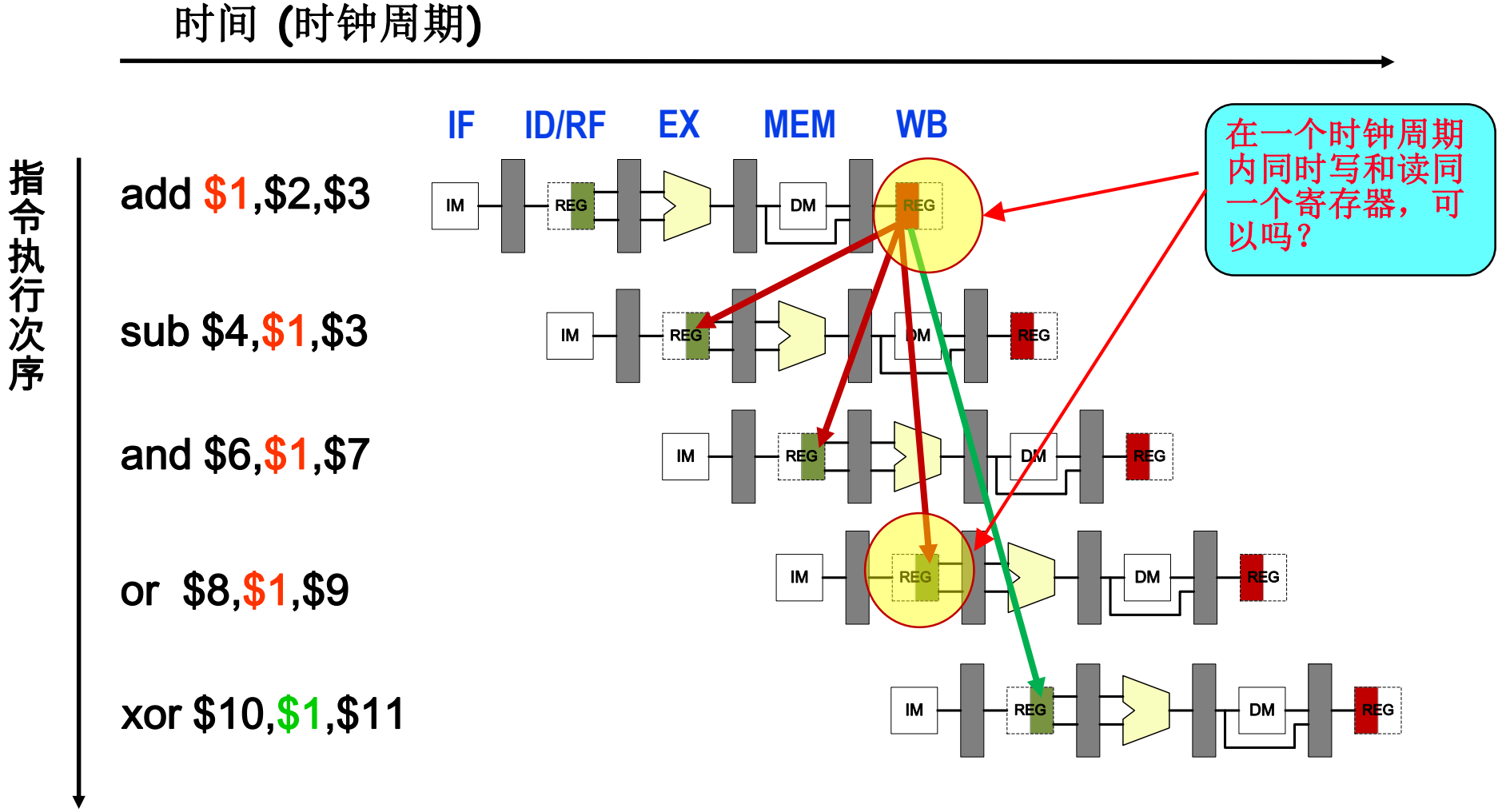
➢后一条指令执行需要使用前一条指令的结果，此时结果尚未形成，带来数据冒险

```
add  $s0, $t0, $t1      // $s0 ← $t0 + $t1
sub  $t2, $s0, $t3      //  $t2 ← $s0 + $t3
```



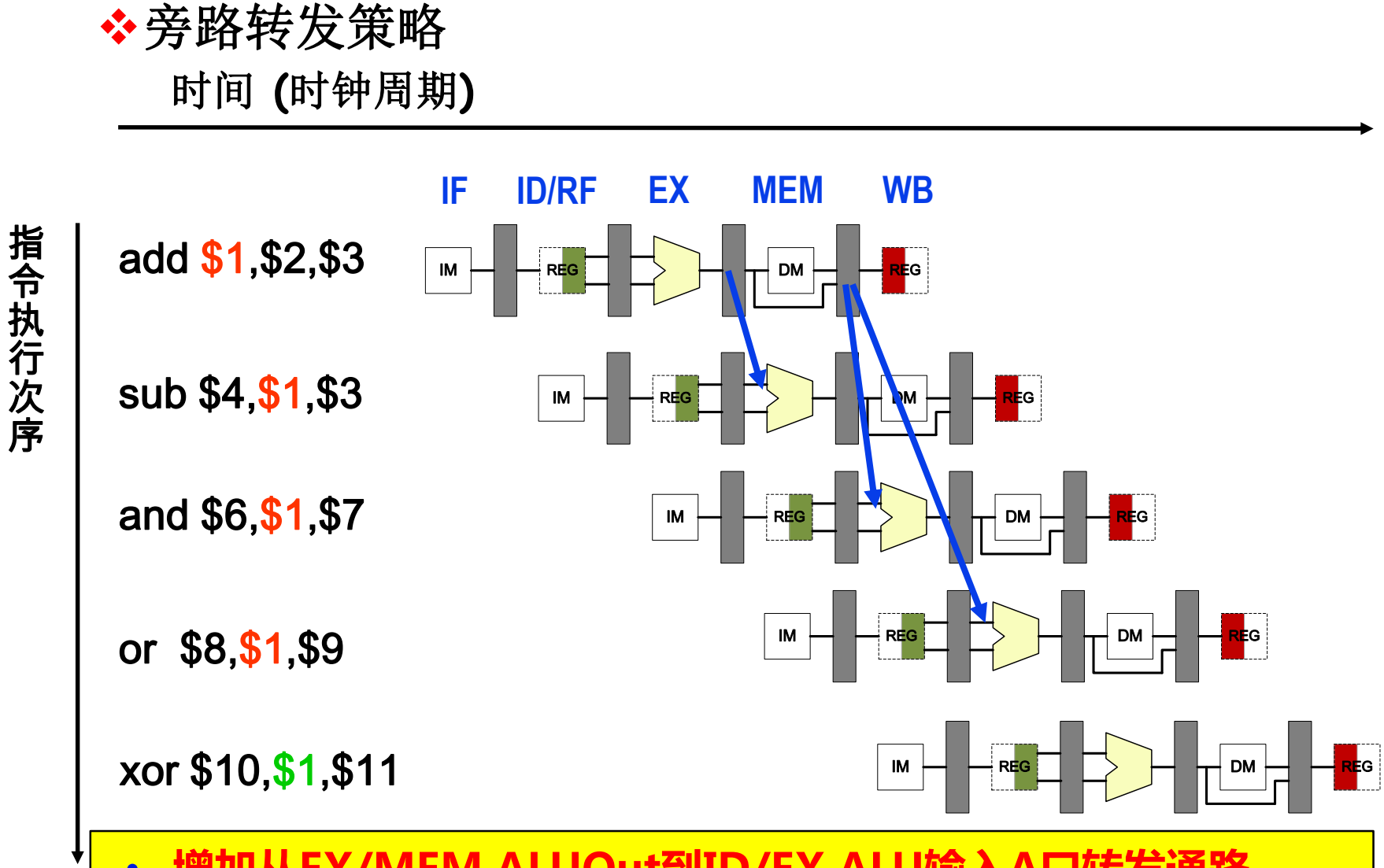
5.3 数据冒险的处理——旁路转发

❖旁路转发策略



5.3 数据冒险的处理——旁路转发

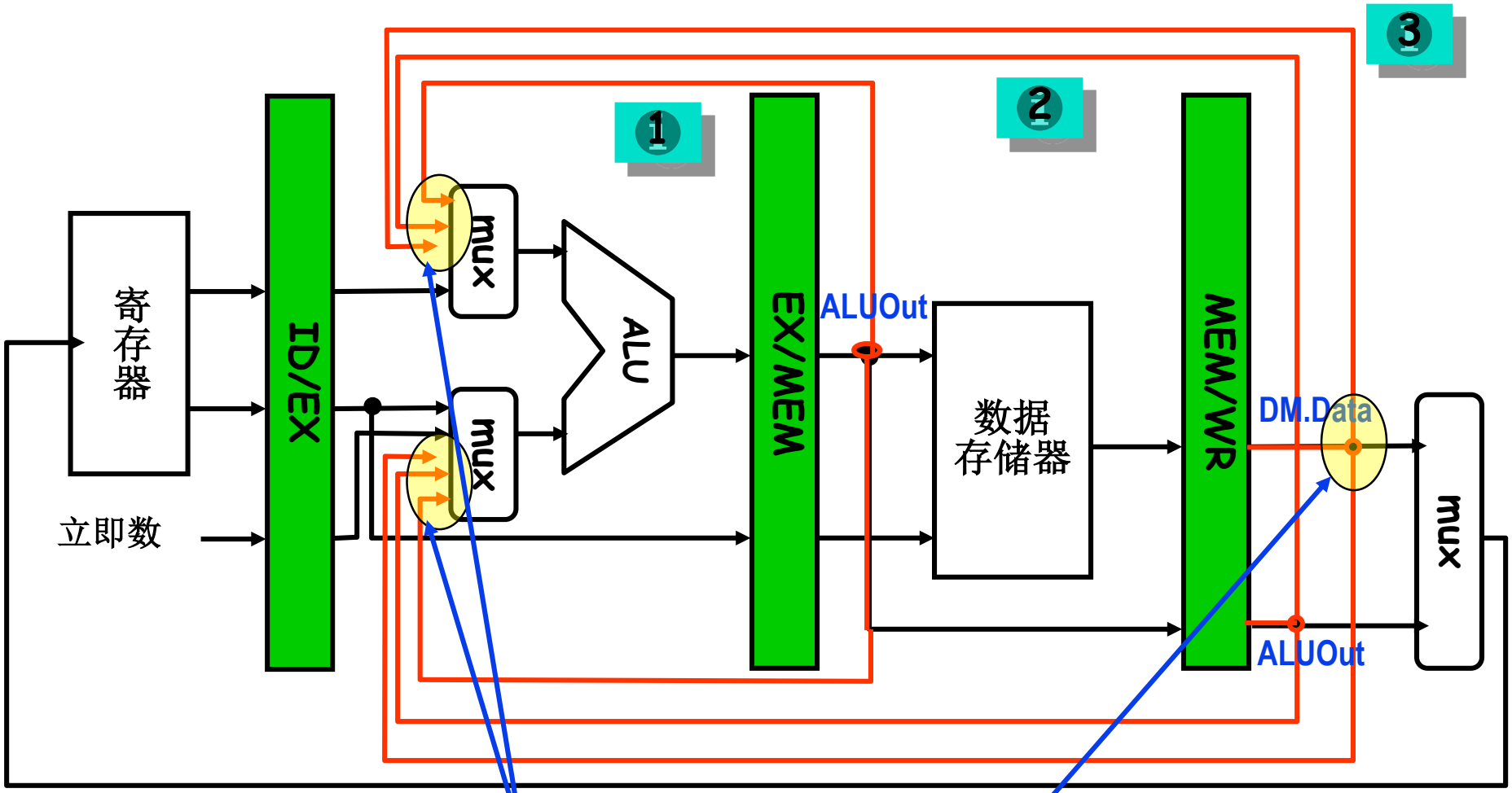
❖旁路转发策略



- 增加从EX/MEM.ALUOut到ID/EX.ALU输入A口转发通路
- 增加从MEM/WB.ALUOut到ID/EX.ALU输入A口转发通路

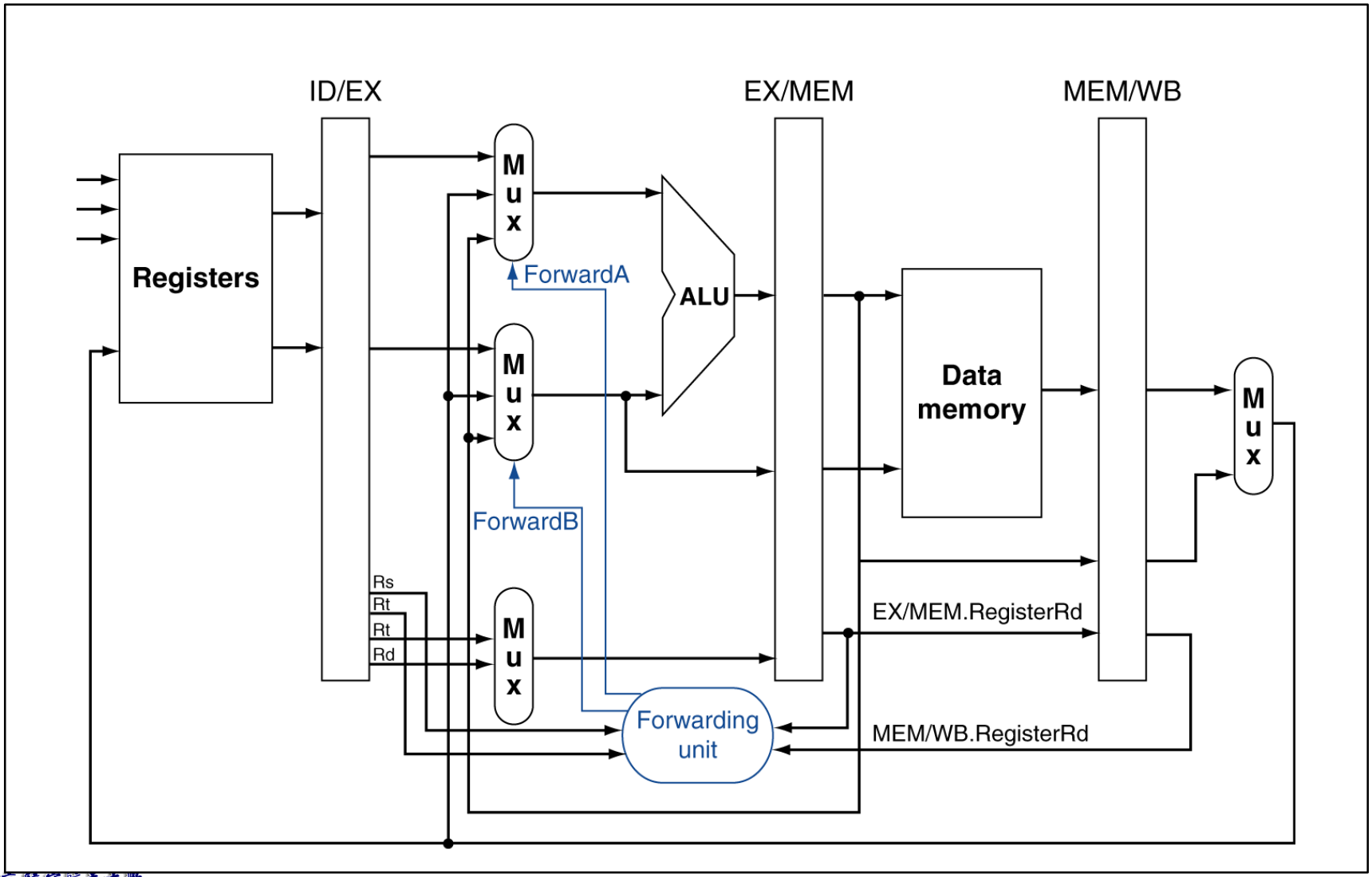
5.3 数据冒险的处理——旁路转发

❖旁路转发策略

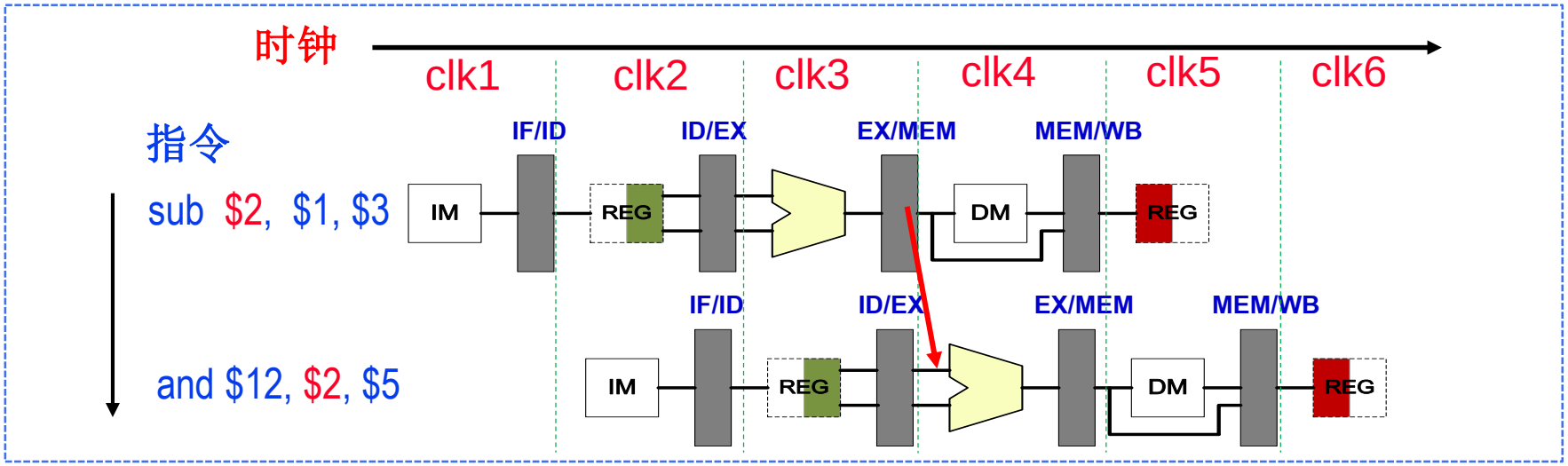


5.3 数据冒险的处理——旁路转发

❖旁路转发策略：数据通路修改，增加转发单元



5.3 数据冒险的检测

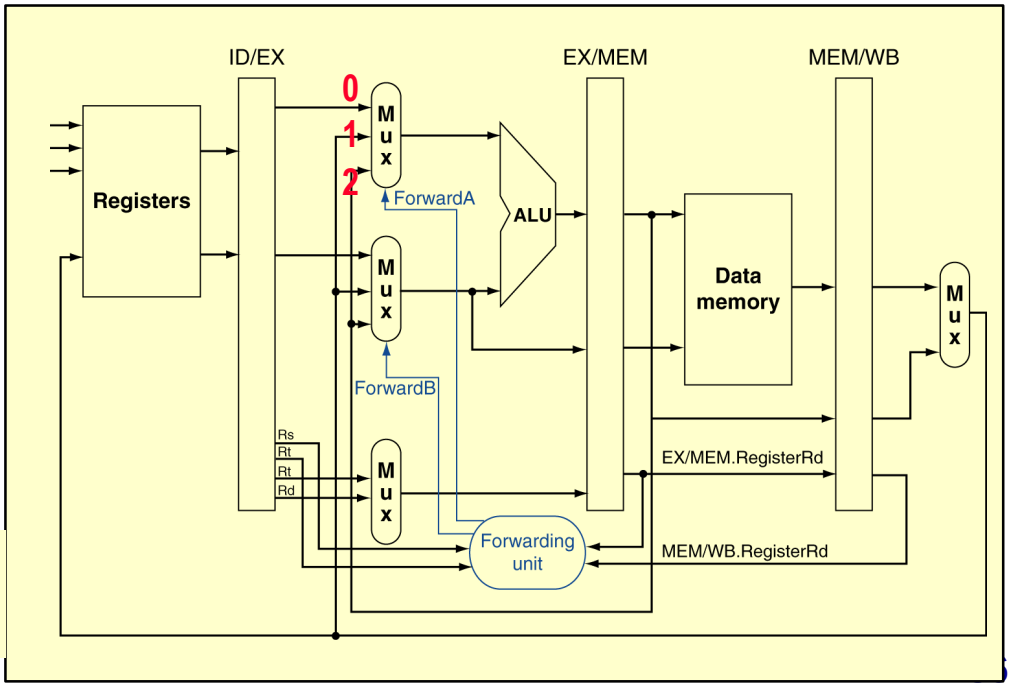


数据冒险检测：

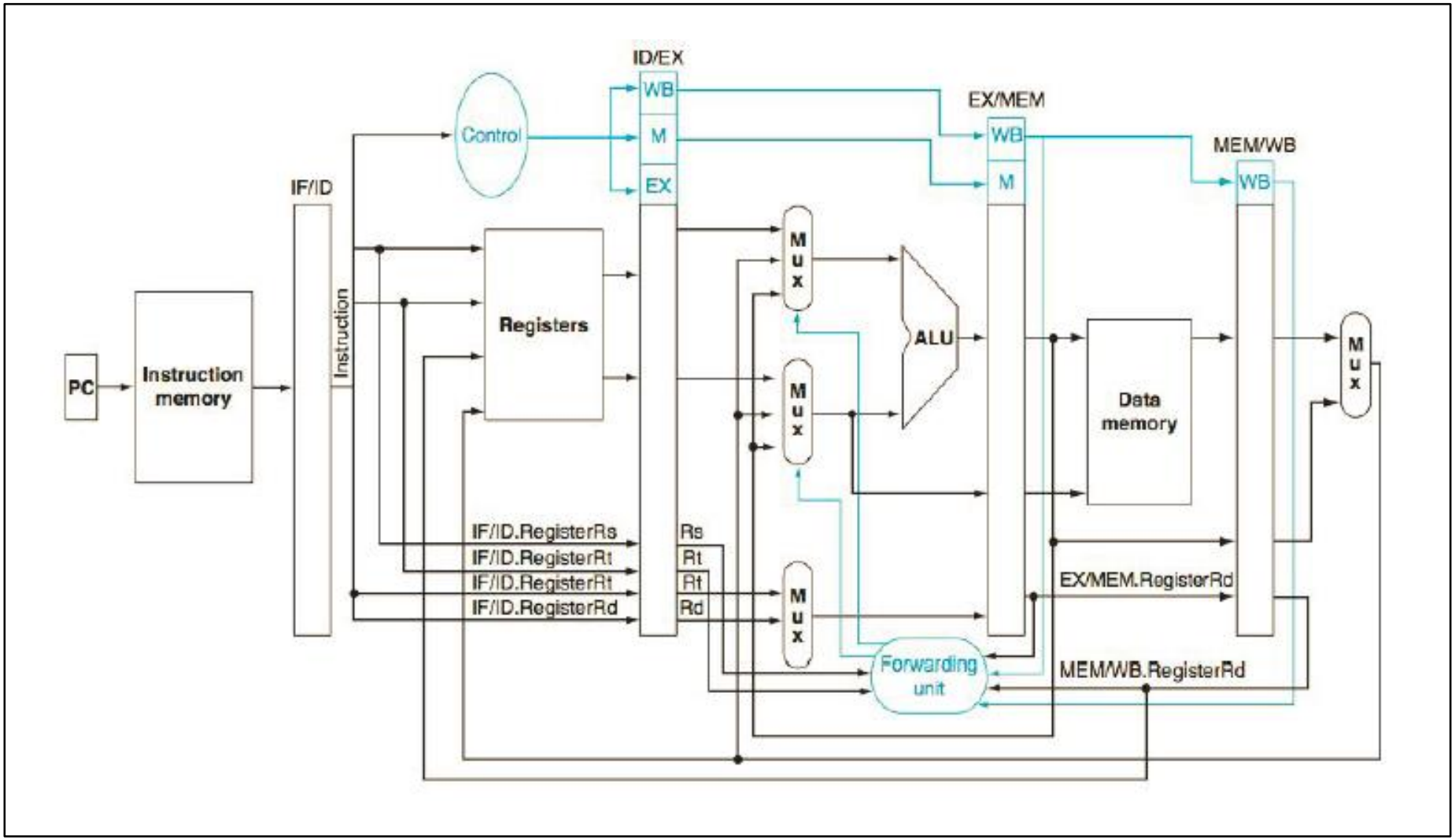
- and指令执行到EX级（sub指令执行到了MEM级）
- sub指令的rd = and指令的rs

If (EX/MEM.Regwrite and
(EX/MEM.RegisterRd=ID/EX.RegisterRs))
ForwardA=2

Q：第二条指令如果是and $12,$5,$2呢？
第三条指令如果是or $13, $2,$4呢？

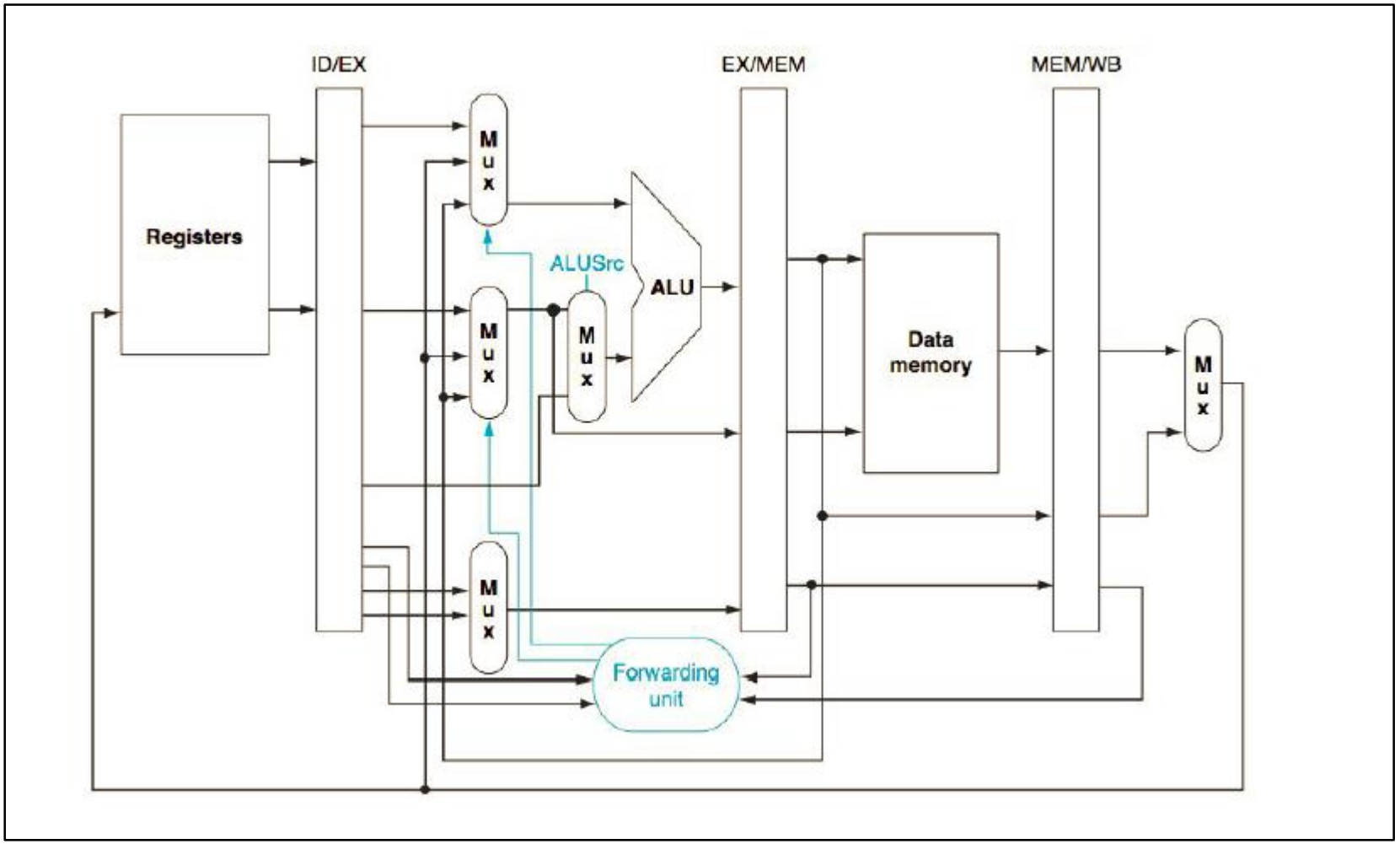


5.3 数据冒险的检测



具备旁路转发功能的数据通路

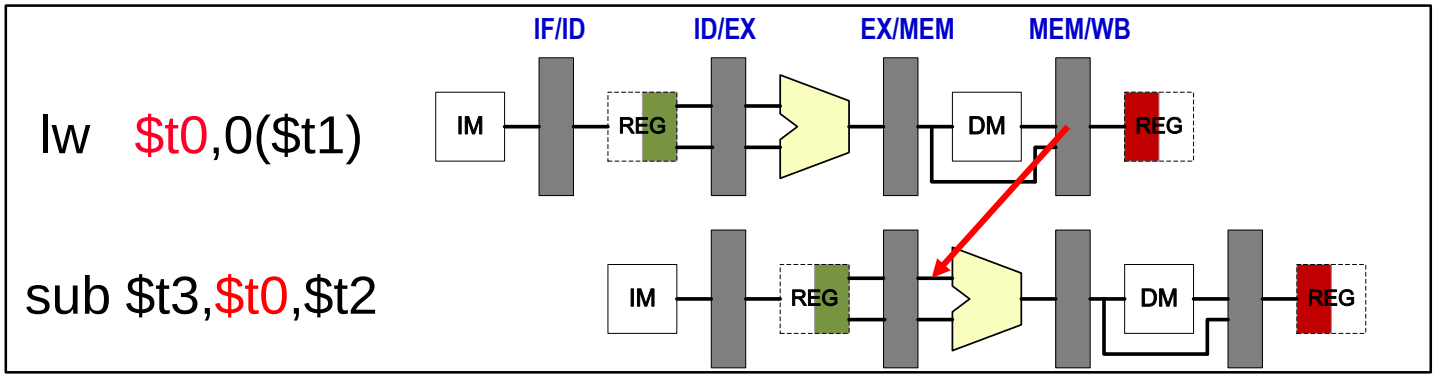
5.3 数据冒险的检测



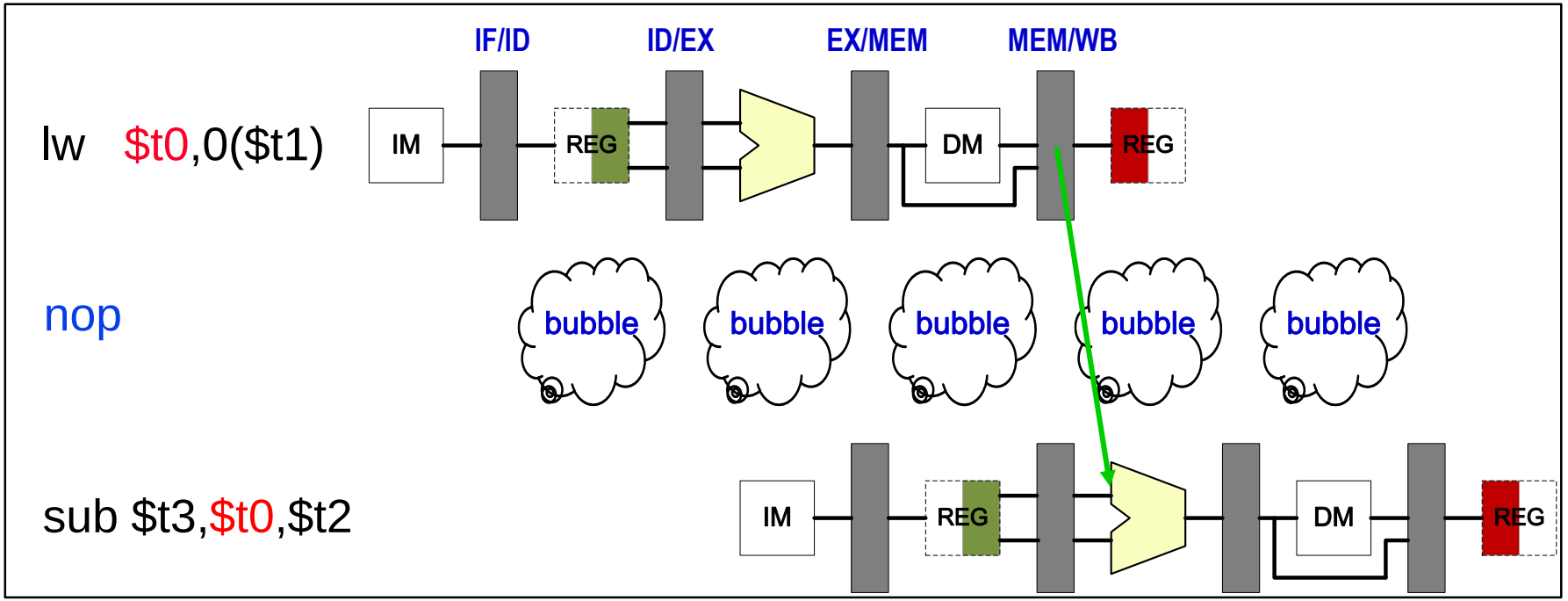
增加了ALUSrc选择，具备旁路转发功能的数据通路

5.3 数据冒险的处理——load导致的数据冒险

❖Load导致的数据冒险



旁路转发
不能解决



必须在lw指令后停顿一个周期（等价于插入一条NOP），然后转发

5.3 数据冒险的处理——load导致的数据冒险

Load导致的数据冒险分析：CLK1上升沿后

- 指令流
  - lw进入IF/ID
  - PC：指向sub指令的地址
    - PC ← PC + 4
  - IM：输出sub指令

		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	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2								
8	and $t5, $t0, $t4								
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

5.3 数据冒险的处理——load导致的数据冒险

Load导致的数据冒险分析：CLK2上升沿后

- 指令流
  - sub进入IF/ID寄存器；lw进入ID/EX寄存器
- 冲突分析：冲突出现
- 执行动作：设置控制信号，在clk3插入nop指令
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

		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	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	**lw**				
4	sub $t3, $t0, $t2	↑ 2	4→8	**sub→and**	**sub**	**lw**			
8	and $t5, $t0, $t4								
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

5.3 数据冒险的处理——load导致的数据冒险

Load导致的数据冒险分析：CLK3上升沿后

- 指令流
  - sub进入ID/EX；lw进入EX/MEM
  - ID/EX（ID/EX已被清零）向ALU提供数据
- 冲突分析：冲突解除
  - 转发机制将在clk4时可以发挥作用

		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	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	**lw**				
4	sub $t3, $t0, $t2	↑ 2	4→8	**sub→and**	**sub**	**lw**			
8	and $t5, $t0, $t4	↑ 3	8→8	and	**sub**	**nop**	**lw**		
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

5.3 数据冒险的处理——load导致的数据冒险

Load导致的数据冒险分析：CLK4上升沿后

- 指令流
  - lw：结果存入MEM/WB。
  - sub：进入ID/EX。故ALU的操作数可以从MEM/WB转发
- 执行动作
  - 控制MUX，使得MEM/WB输入到ALU

		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	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	**lw**				
4	sub $t3, $t0, $t2	↑ 2	4→8	**sub→and**	**sub**	**lw**			
8	and $t5, $t0, $t4	↑ 3	8→8	and	**sub**	**nop**	**lw**		
12	or $t7, $t0, $t6	↑ 4	8→12	and→or	**and**	**sub**	**nop**	**lw结果**	
16	add $t1, $t2, $t3								

5.3 数据冒险的处理——load导致的数据冒险

Load导致的数据冒险分析：CLK5上升沿后

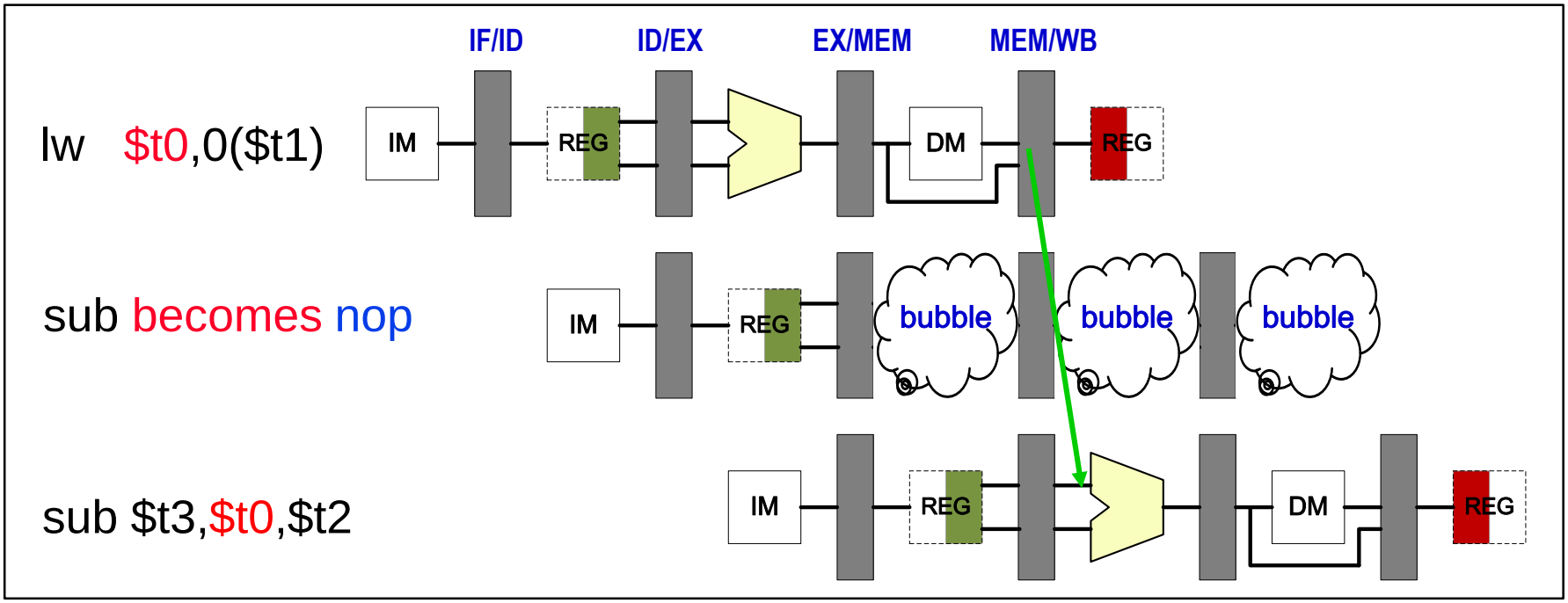
- 指令流
  - **lw**：结果回写至**RF**
  - **sub**：结果保存在**EX/MEM**

		IF级	ID级	EX级	MEM级	WB级	
地址	指令	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	lw→sub	**lw**				
4	sub $t3, $t0, $t2	**sub→and**	**sub**	**lw**			
8	and $t5, $t0, $t4	and	**sub**	**nop**	**lw**		
12	or $t7, $t0, $t6	and→or	**and**	**sub**	**nop**	**lw结果**	
16	add $t1, $t2, $t3	or→add	**or**	**and**	**sub结果**	**nop**	**lw结果**

CLK	PC
⮥ 1	0→4
⮥ 2	4→8
⮥ 3	8→8
⮥ 4	8→12
⮥ 5	12→16

5.3 数据冒险的处理——load导致的数据冒险

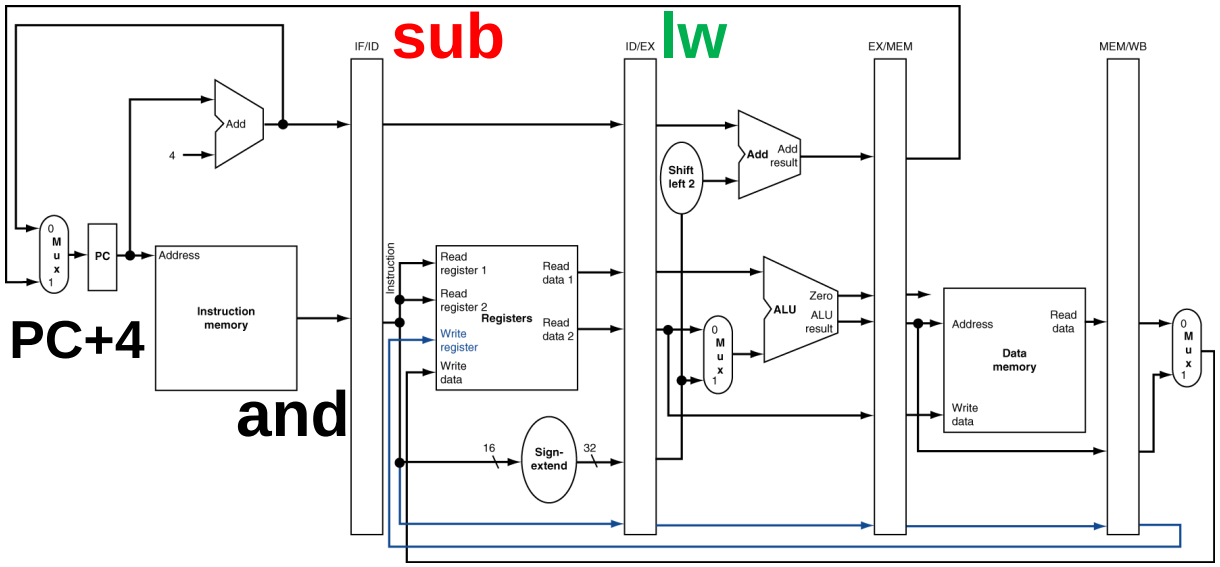
❖Load导致的数据冒险分析



5.3 数据冒险的处理——如何插入nop

- 检测条件：IF/ID的前序是lw指令，并且lw的rt寄存器与IF/ID的rs或rt相同
- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

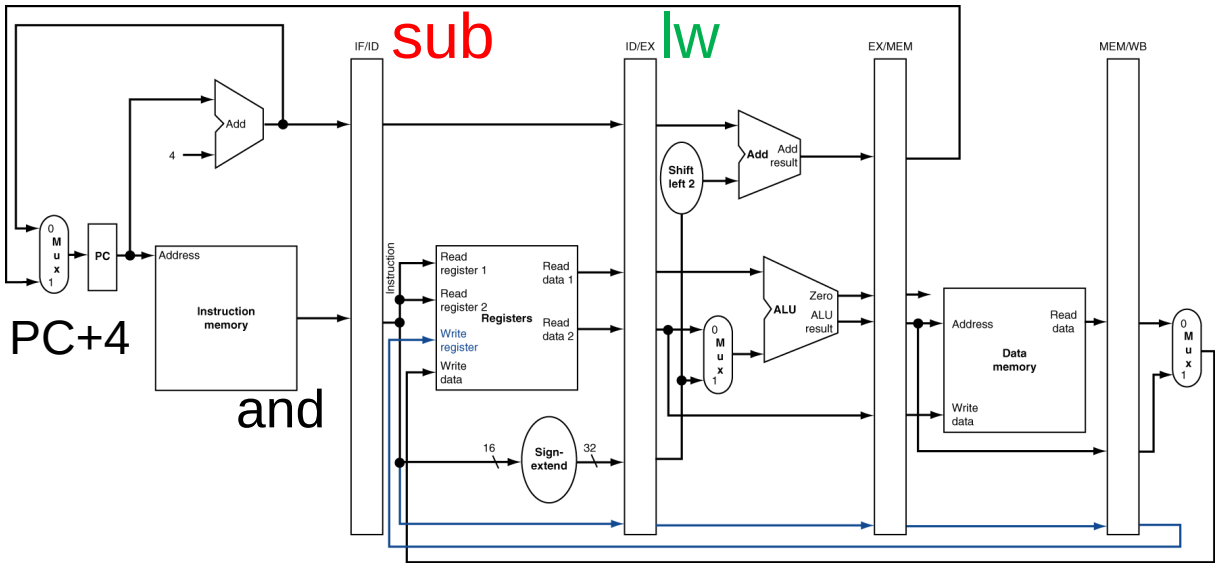
地址	指令
0	lw $t0, 0($t1)
4	sub $t3, $t0, $t2
8	and $t5, $t0, $t4
12	or $t7, $t0, $t6
16	add $t1, $t2, $t3



5.3 数据冒险的处理——如何插入nop

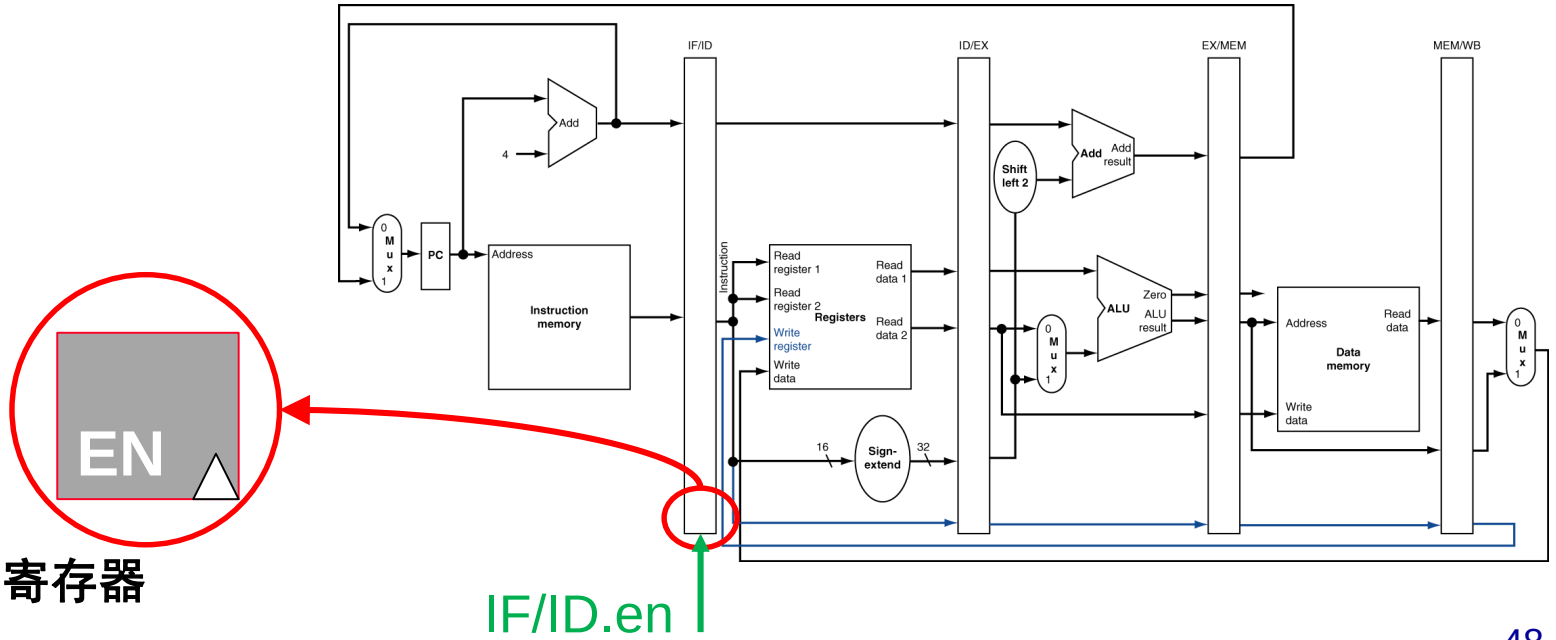
- 检测条件：IF/ID的前序是lw指令，并且lw的rt寄存器与IF/ID的rs或rt相同
- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

地址	指令
0	`lw $t0, 0($t1)`
4	`sub $t3, $t0, $t2`
8	`and $t5, $t0, $t4`
12	`or $t7, $t0, $t6`
16	`add $t1, $t2, $t3`



5.3 数据冒险的处理——如何插入nop

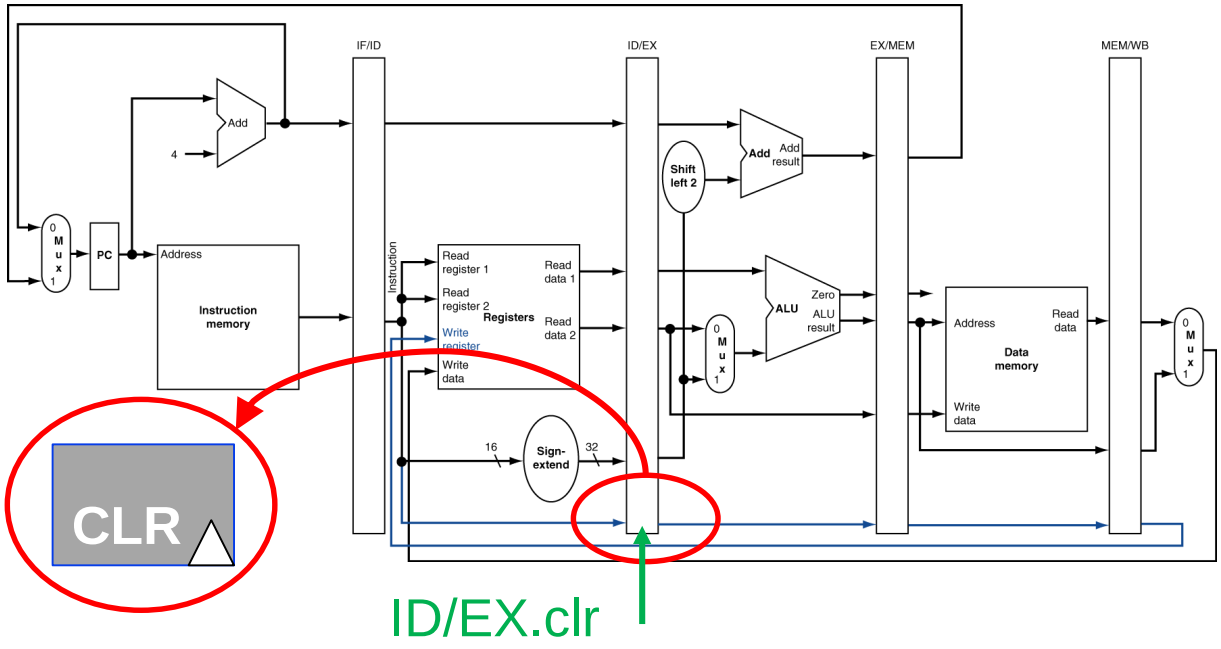
- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将IF/ID修改为使能型寄存器
- 控制系统：增加IF/ID.en控制信号
  - 当IF/ID.en为0时，IF/ID在下个clock上升沿到来时保持不变



- 使能型寄存器

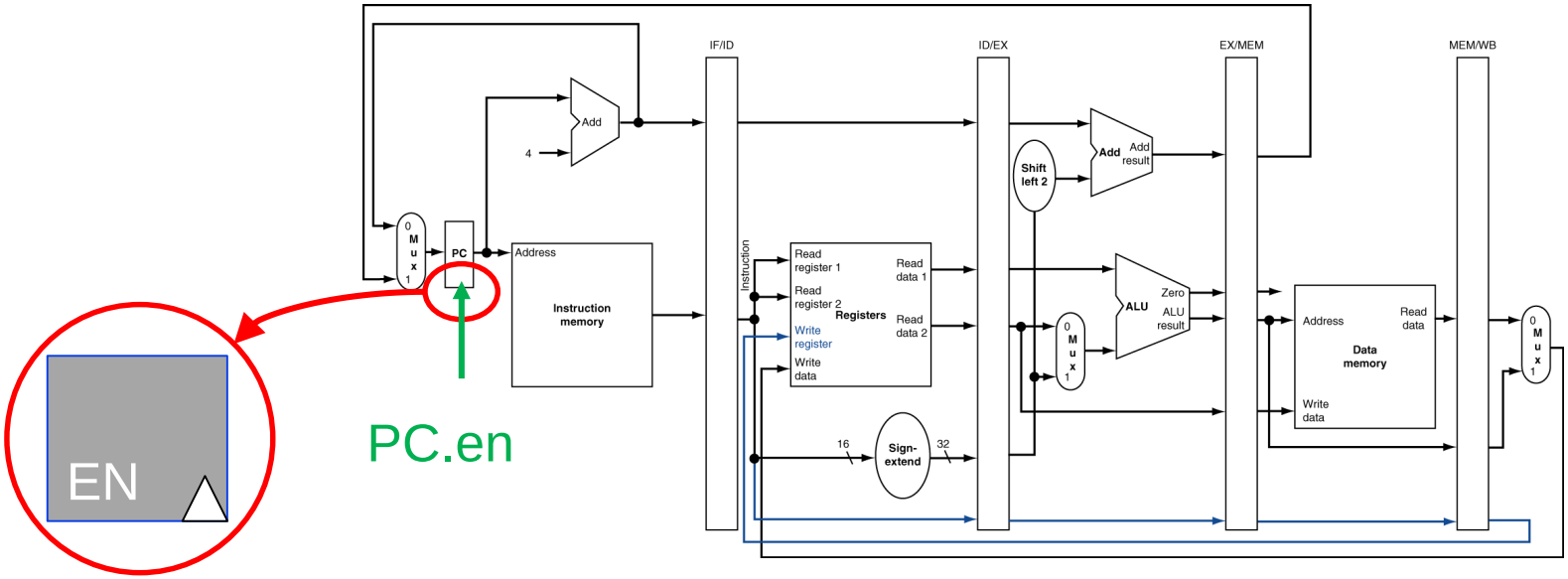
5.3 数据冒险的处理——如何插入nop

- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将ID/EX修改为复位型寄存器
- 控制系统：增加ID/EX.clr控制信号
  - 当ID/EX.clr为0时，ID/EX在下个clock上升沿到来时被清除为0



5.3 数据冒险的处理——如何插入nop

- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将PC修改为使能型寄存器
- 控制系统：增加PC.en控制信号
  - 当PC.en为0时，PC在下个clock上升沿到来时保持不变



5.3 数据冒险的处理——如何插入nop

- lw冒险处理示例伪代码
- 注意：时序关系
  - 各信号在clk2上升沿后有效
  - NOP是在clk3上升沿后发生，即寄存器值在clk3上升沿到来时发生变化(或保持不变)

```
if (ID/EX.MemRead) &
   ((ID/EX.rt == IF/ID.rs) |
    (ID/EX.rt == IF/ID.rt)
    IF/ID.en ← 禁止
    ID/EX.clr ← 清除
    PC.en ← 禁止
```

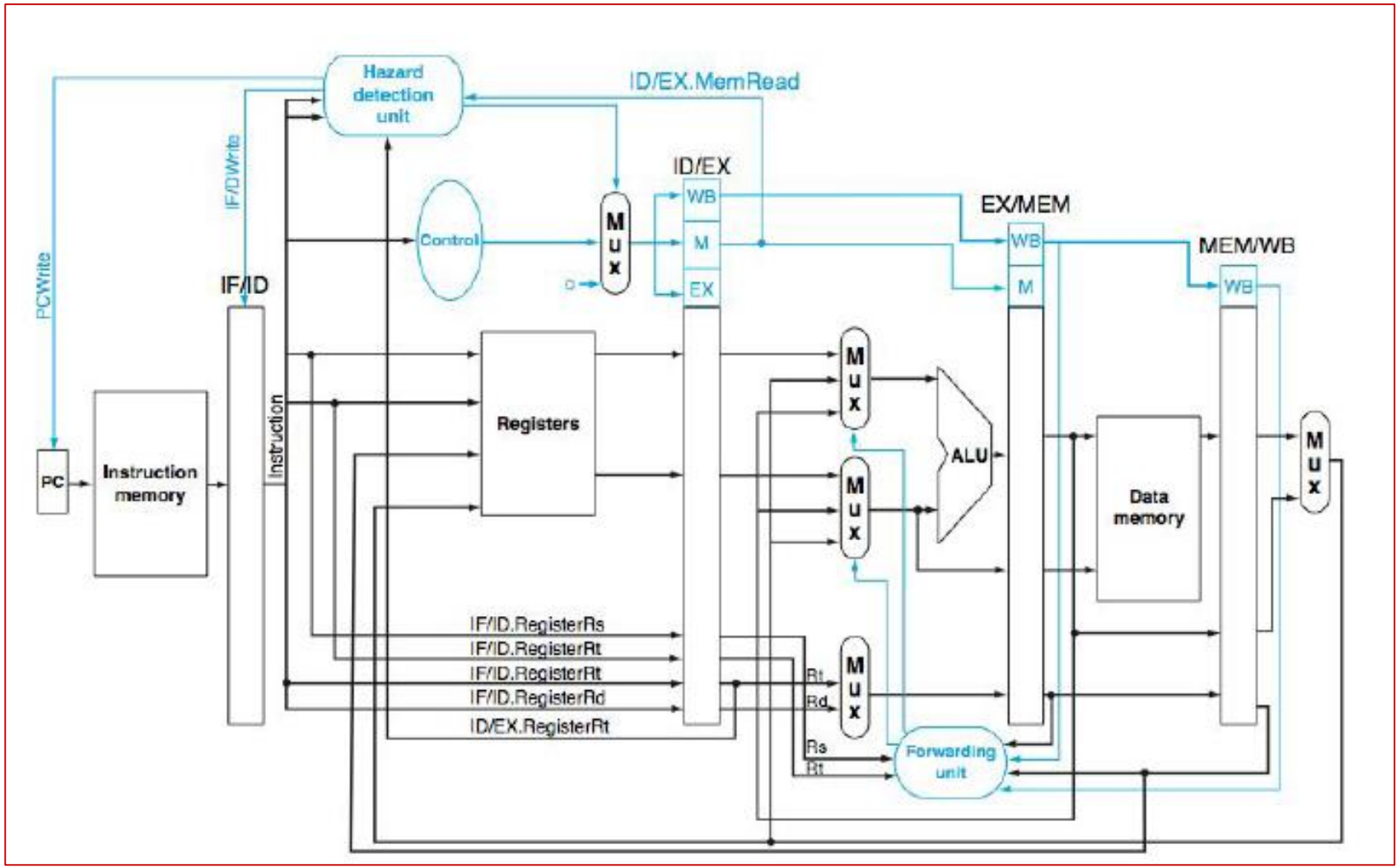
		IF级	ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4	↑ 3	8→8	and	sub	nop	lw		
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

5.3 数据冒险的处理——如何插入nop

- 由于有转发电路，因此lw指令只插入1个NOP指令
- Q：如果没有转发，需要怎么处理呢？
- A：EX/MEM，MEM/WB也均需要做冲突分析及NOP处理
  - EX/MEM，MEM/WB也需要修改，并增加相应控制信号

				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	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	**lw**				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	**sub**	**lw**			
8	and $t5, $t0, $t4	↑ 3	8	and	**sub**	**nop**	**lw**		
12	or $t7, $t0, $t6	↑ 4	8	and	**sub**	**nop**	**nop**	**lw结果**	
16	add $t1, $t2, $t3	↑ 5	8	and	**sub**	**nop**	**nop**	**nop**	**lw结果**
		↑ 6	8→12	and→or	**and**	**sub**	**nop**	**nop**	**nop**

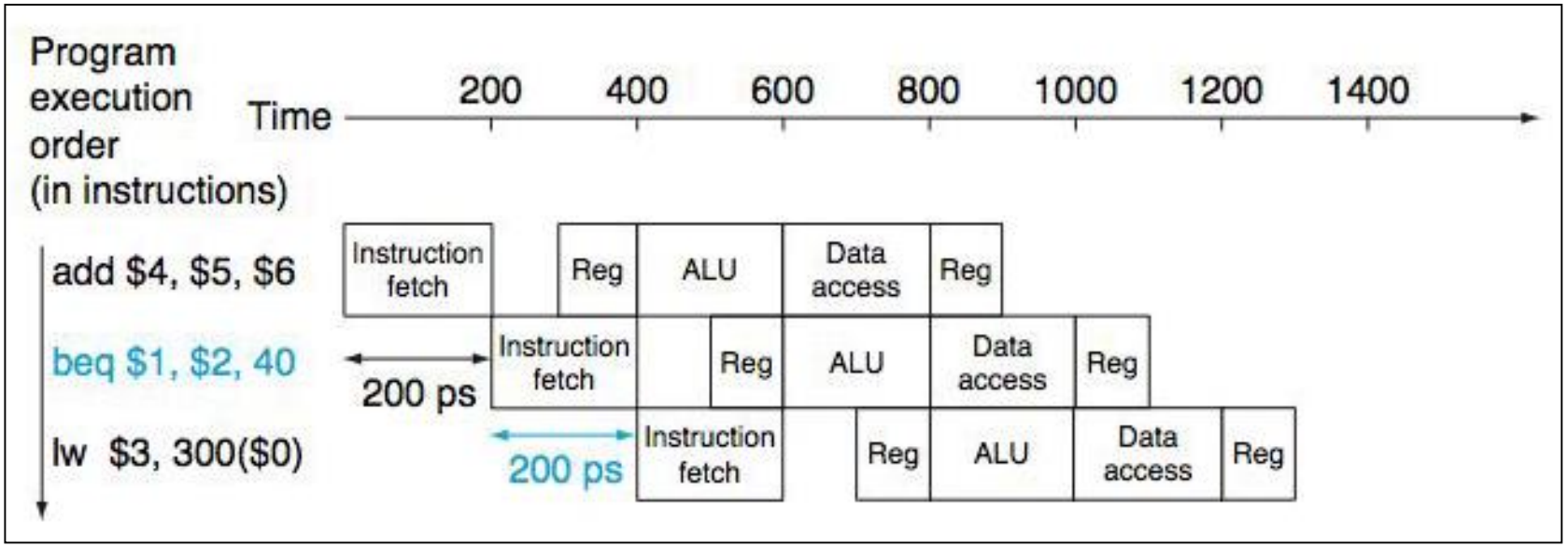
5.3 数据冒险的处理



带数据冒险检测、处理和旁路转发功能的数据通路

5.3 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。

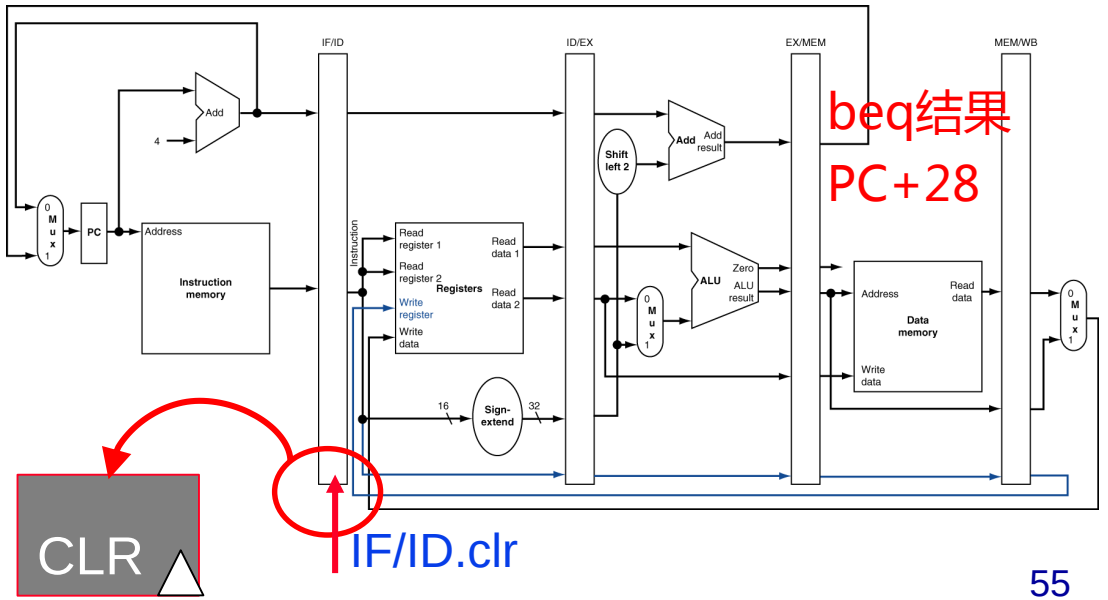


beq $1, $2, 40 执行时，可能发生条件转移，而不会执行lw $3, 300($0)

5.3 控制冒险的处理——停顿的代价

地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
					IF级	ID级	EX级	MEM级	WB级
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		
		↑ 5	28→32	lw→XX	lw	nop	nop	nop	nop
28	lw $4, 50($7)								

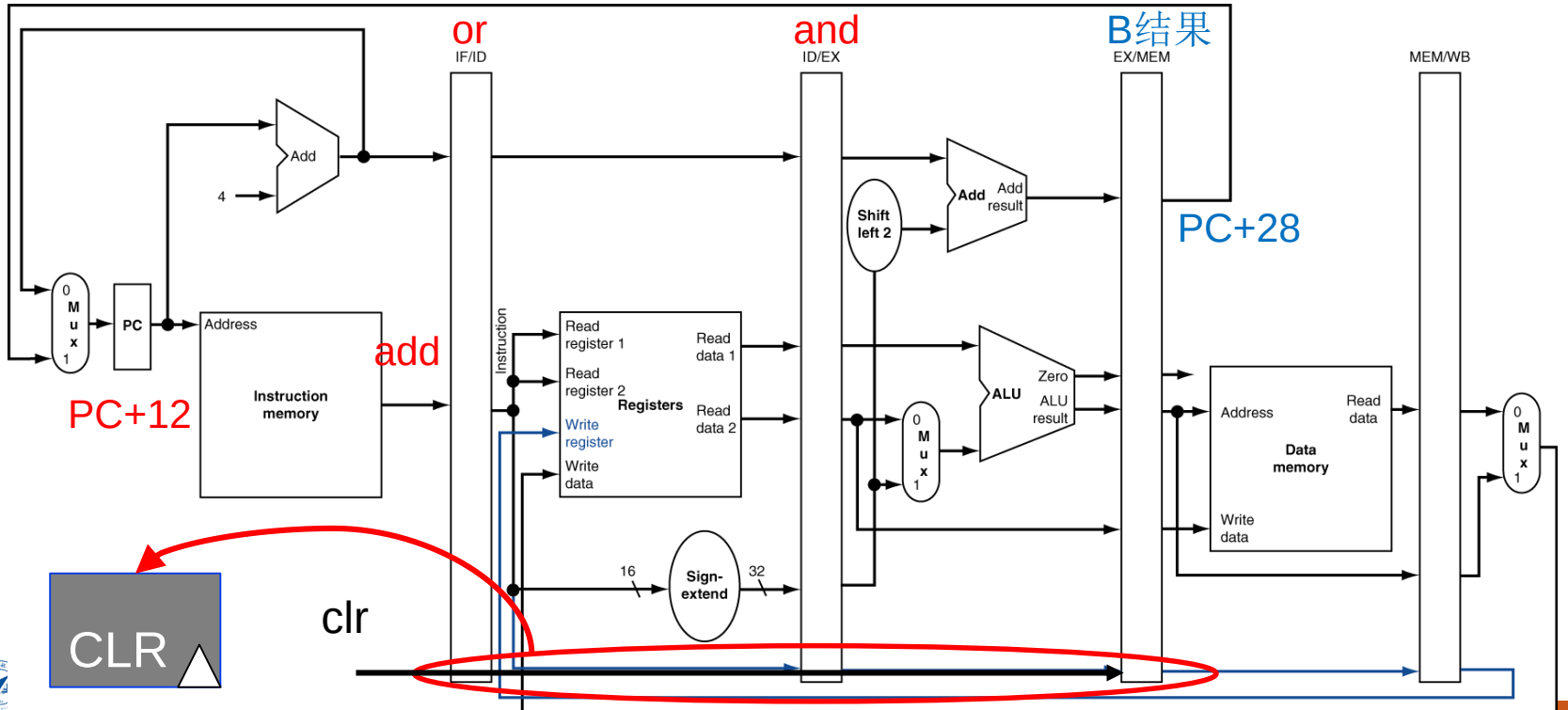
- 如不对B指令做任何处理，则必须插入3个NOP
  - b指令结果及新PC值保存在EX/MEM，因此PC在clk4才能加载正确值
  - IF/ID在clk5才能存入转移后指令(即lw指令)



5.3 控制冒险的处理——假定分支不会发生

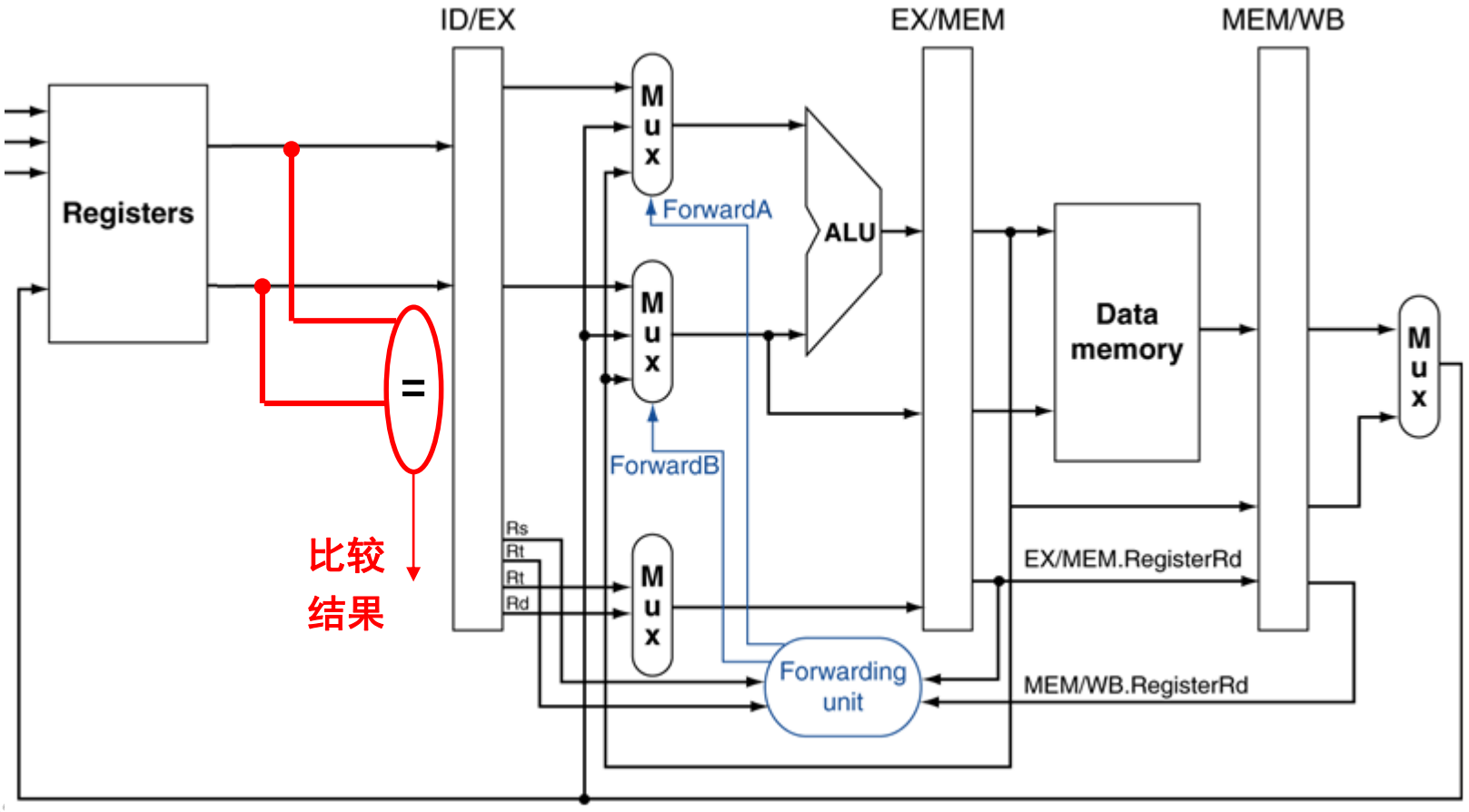
- 即使在ID级发现是B指令也不停顿
- 根据B指令结果，决定是否清除3条后继指令
  - 使得and/or/add不能前进

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)



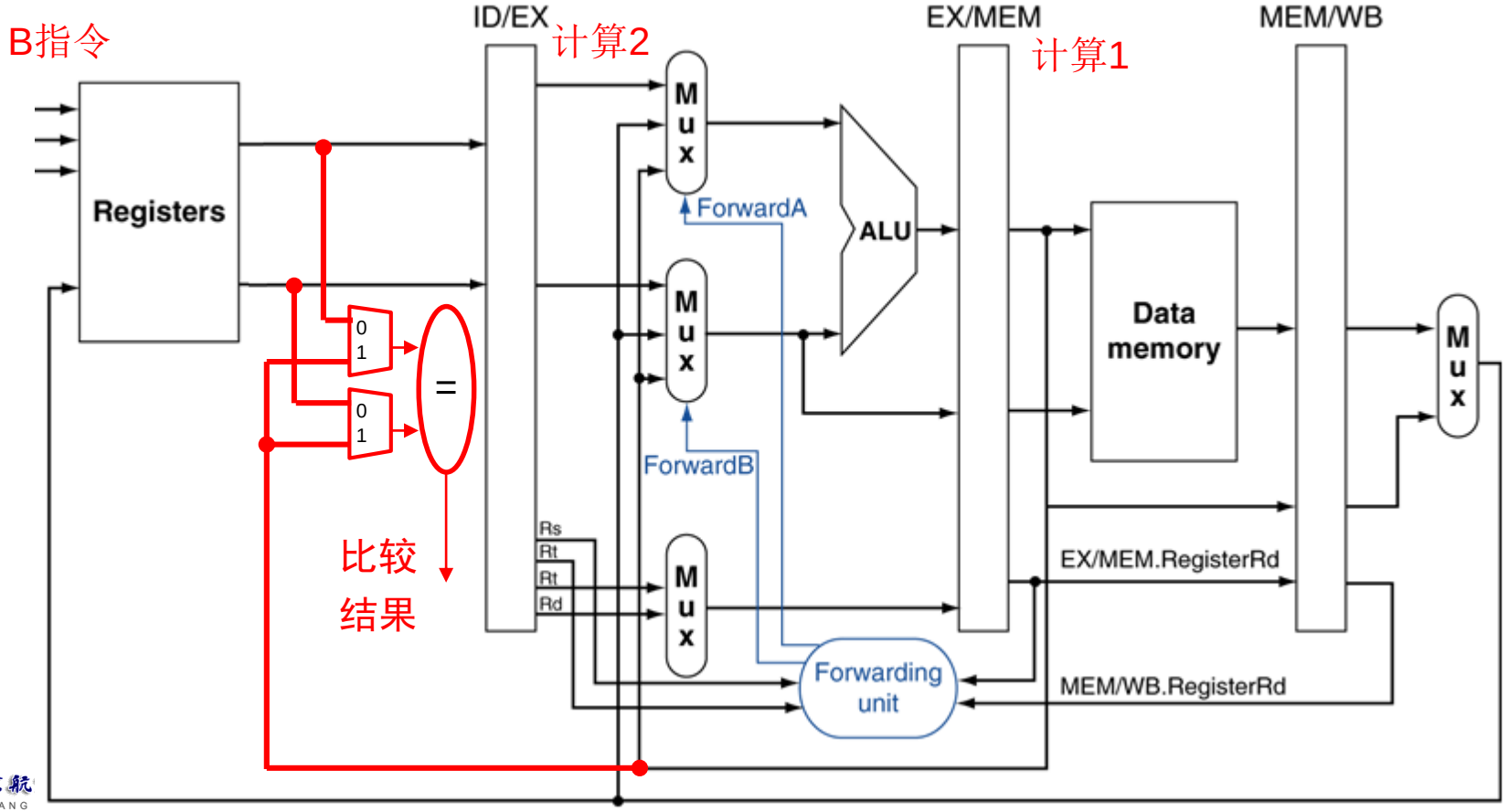
5.3 控制冒险的处理——缩短分支延迟

- 在ID阶段放置比较器，尽快得到B指令结果
  - B指令结果可以提前2个clock得到
  - B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

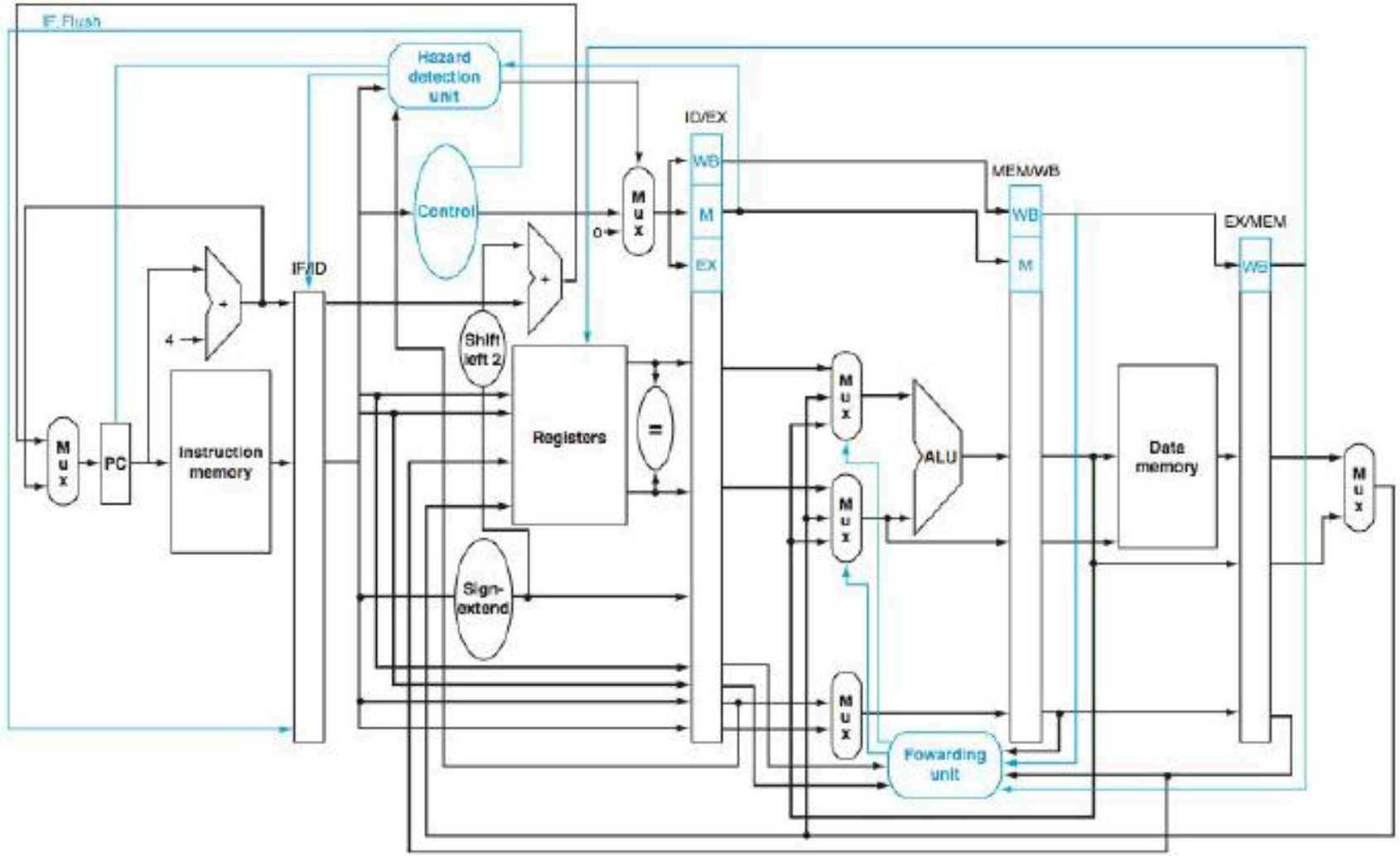


5.3 控制冒险的处理——缩短分支延迟

- 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



5.3 控制冒险的处理


IF.Flush
Hazard detection unit
Control
ID/EX
WB
M
EX
MEM/WB
EX/MEM
IF/ID
Shift left 2
Registers
ALU
Data memory
Instruction memory
PC
Sign-extend
Fowarding unit
M u x
=
+
4