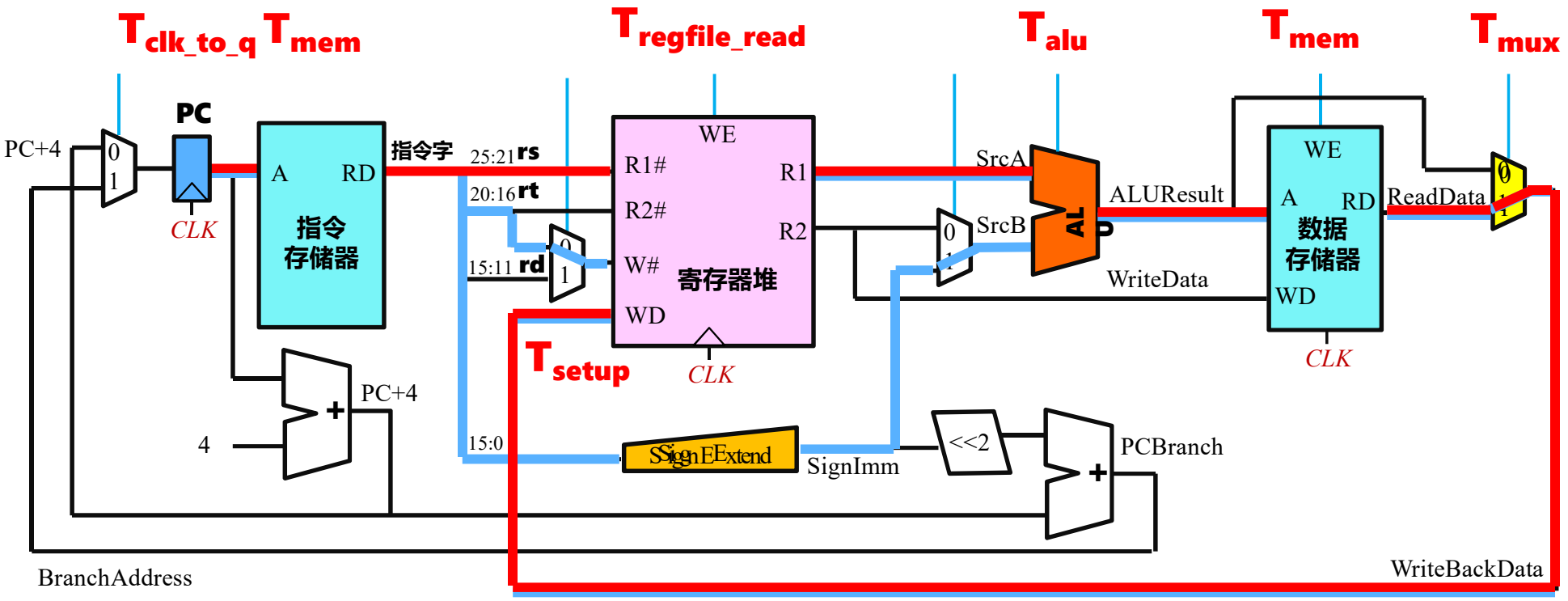
多周期 MIPS CPU

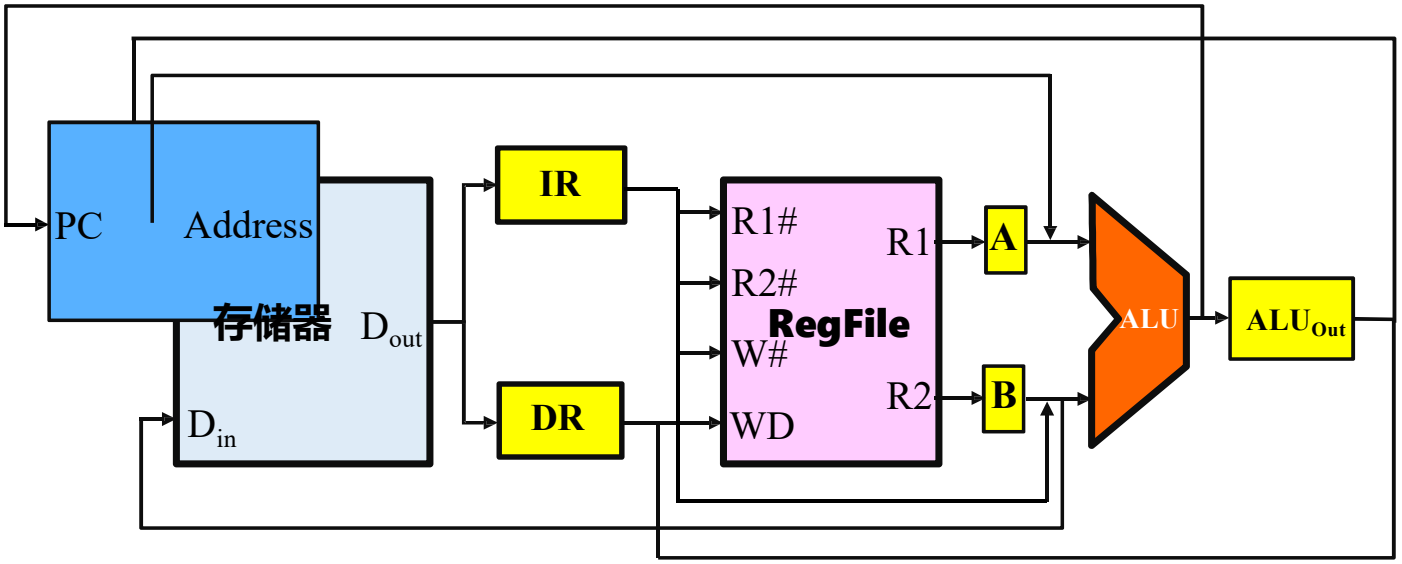
单周期MIPS关键路径---LW指令



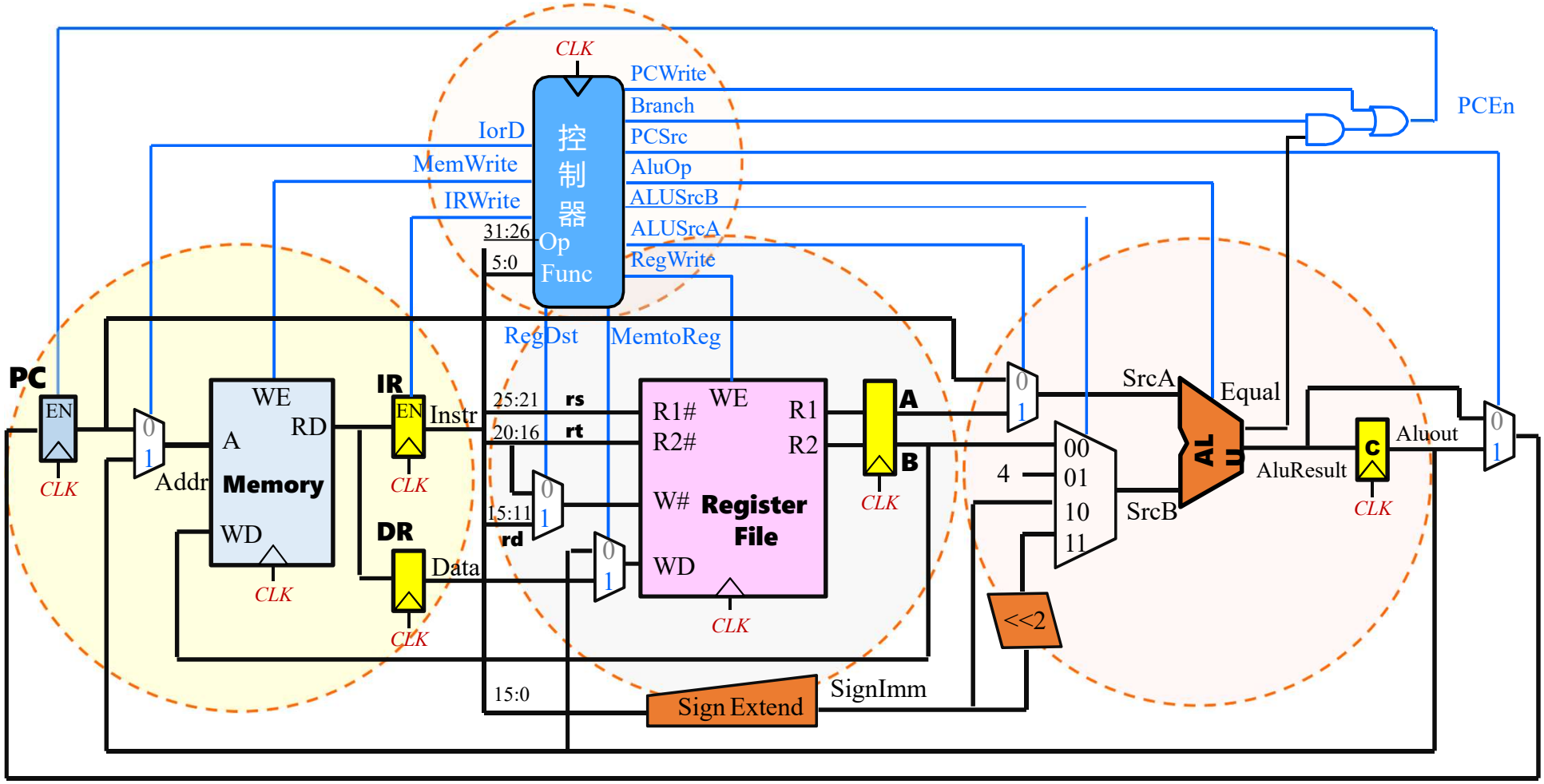
性能取决于最慢的指令，时钟周期过长

多周期MIPS数据通路特点

- 不再区分指令/数据存储器，分时使用功能部件
- 时钟周期变小、传输通路变短
- 功能部件输出端增加寄存器锁存数据

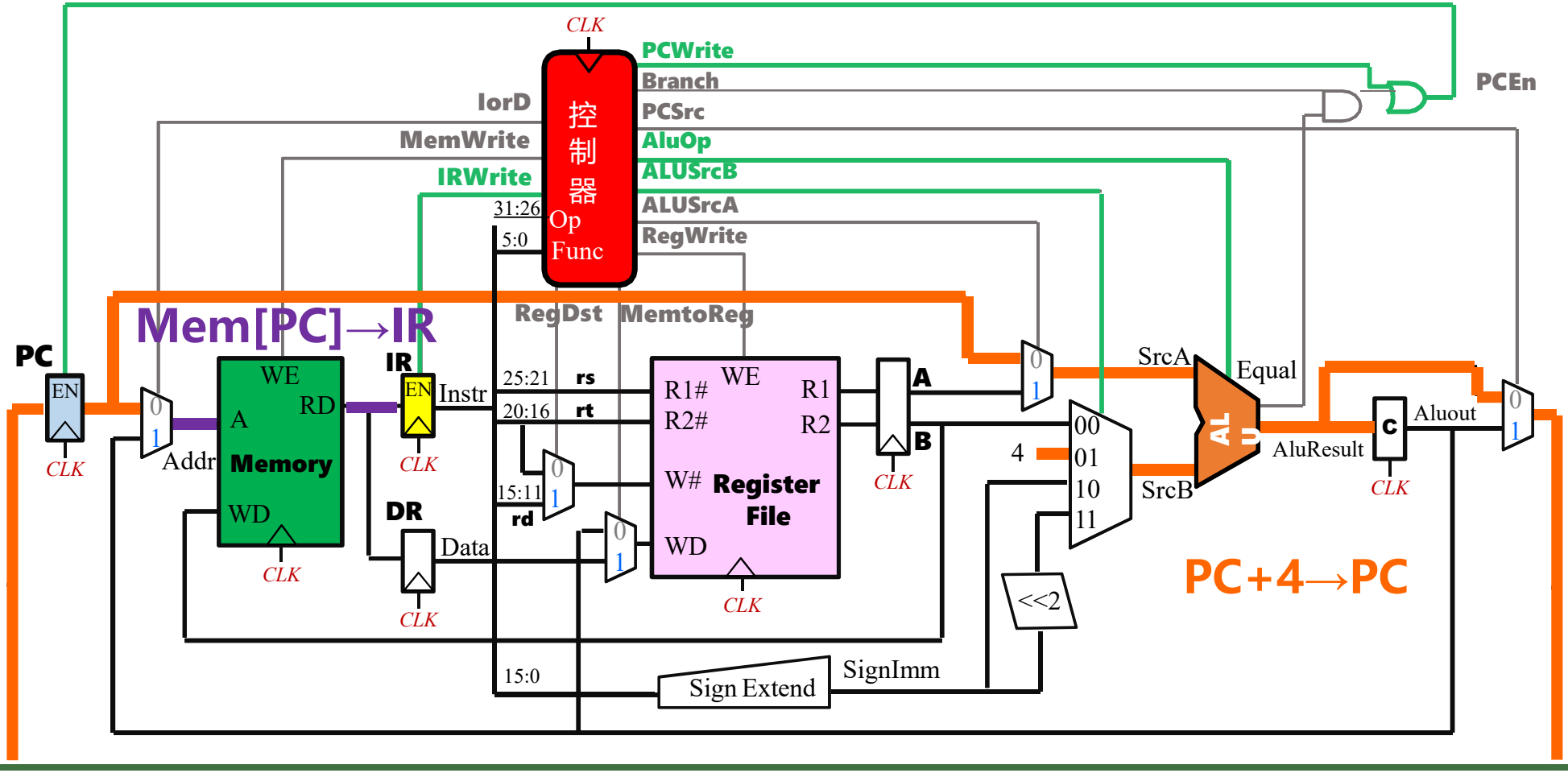


多周期MIPS CPU数据通路



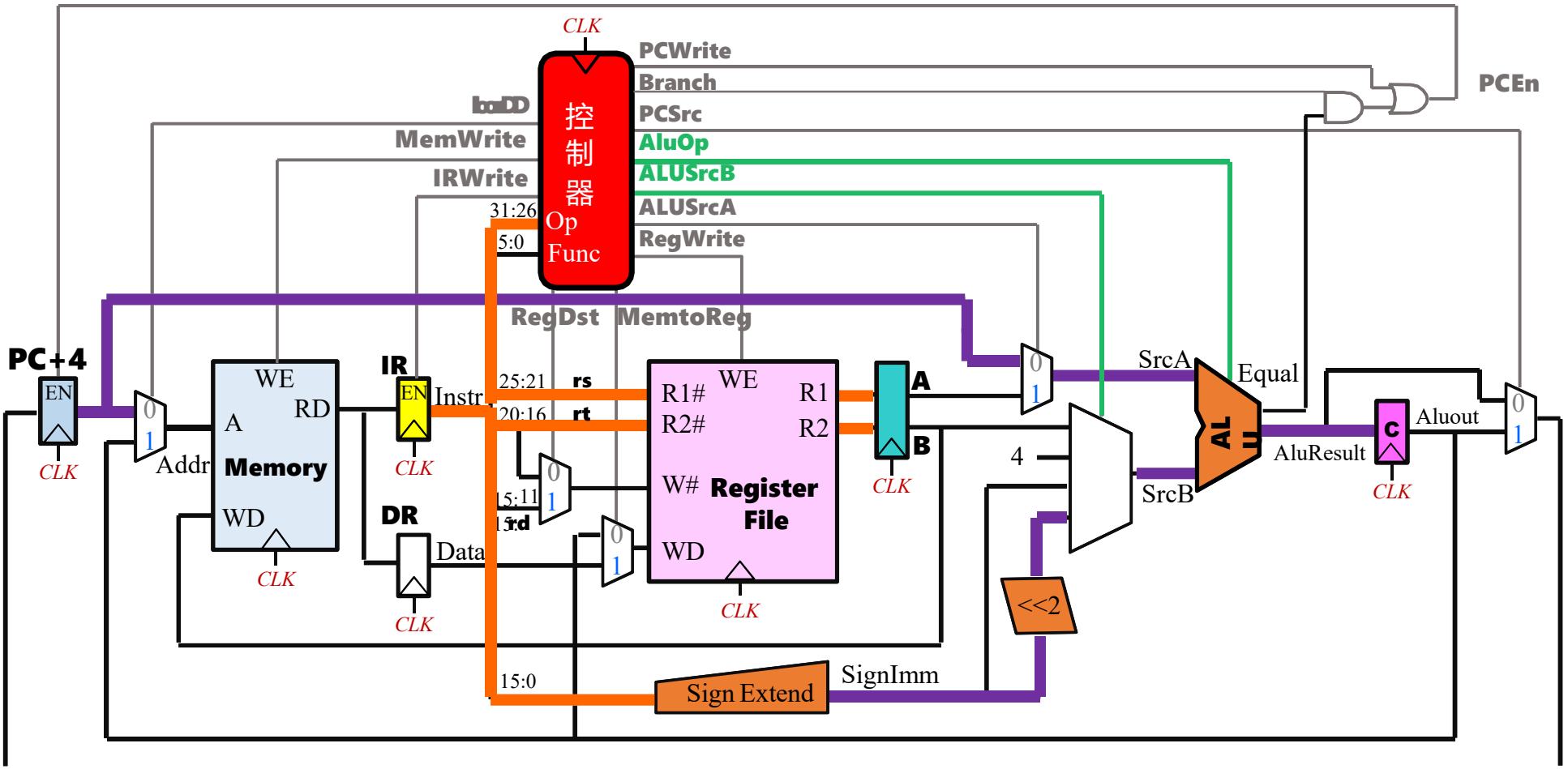
多周期MIPS取指令阶段T1

Mem[PC]→IR　PC+4→PC

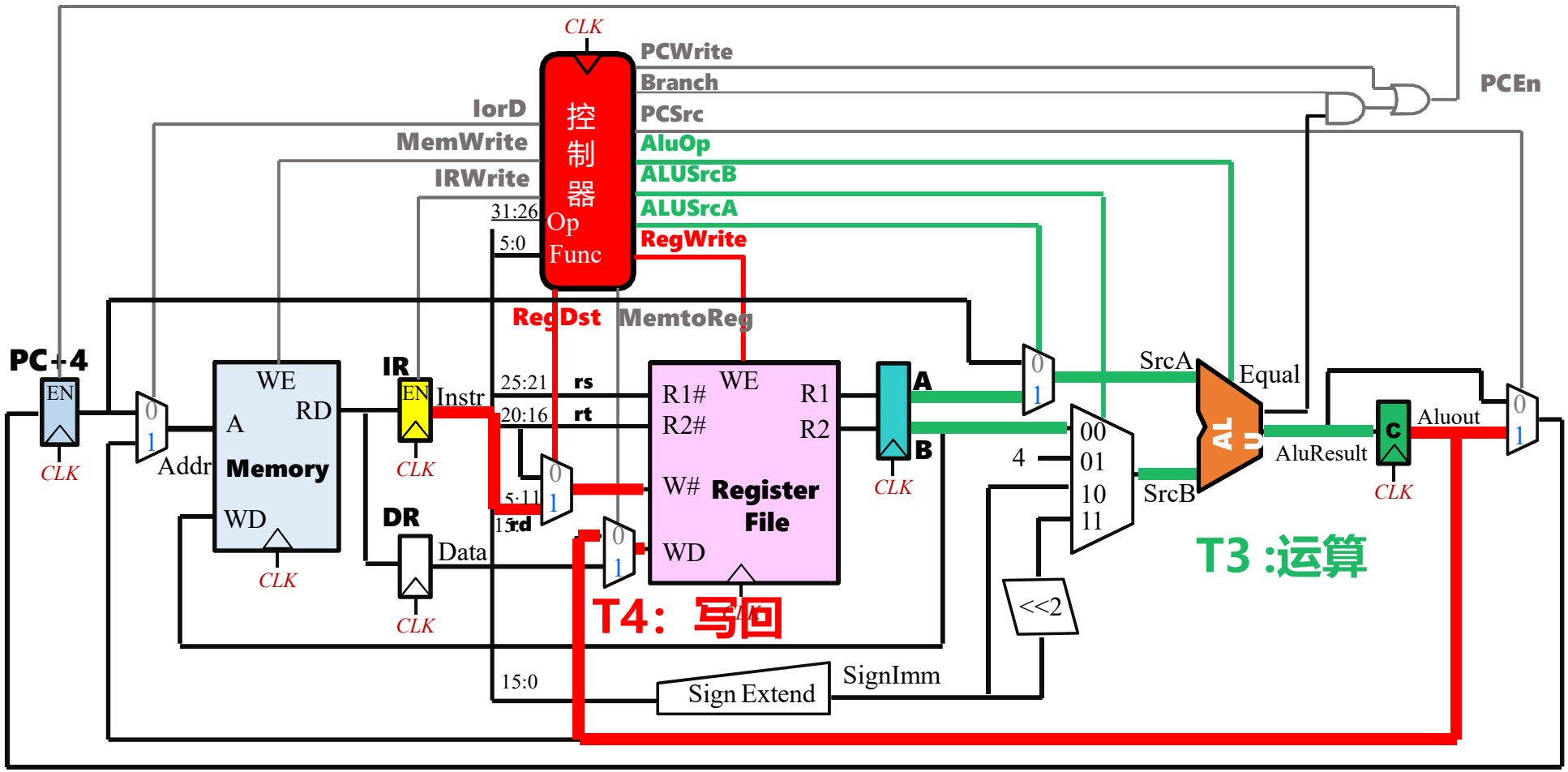


多周期MIPS取指令阶段T2

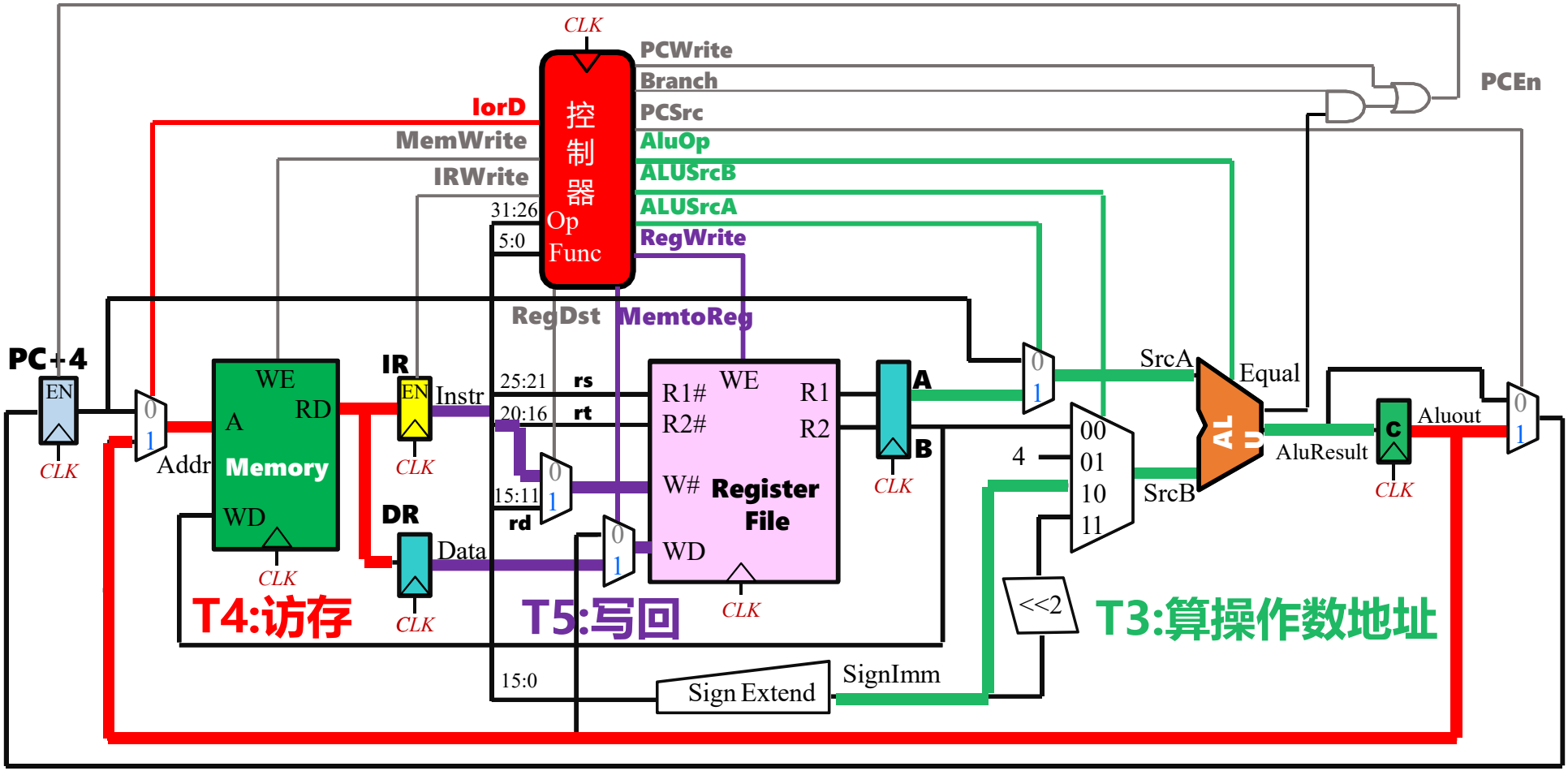
译码、Reg→A、B、PC+4+Imm16<<2→C



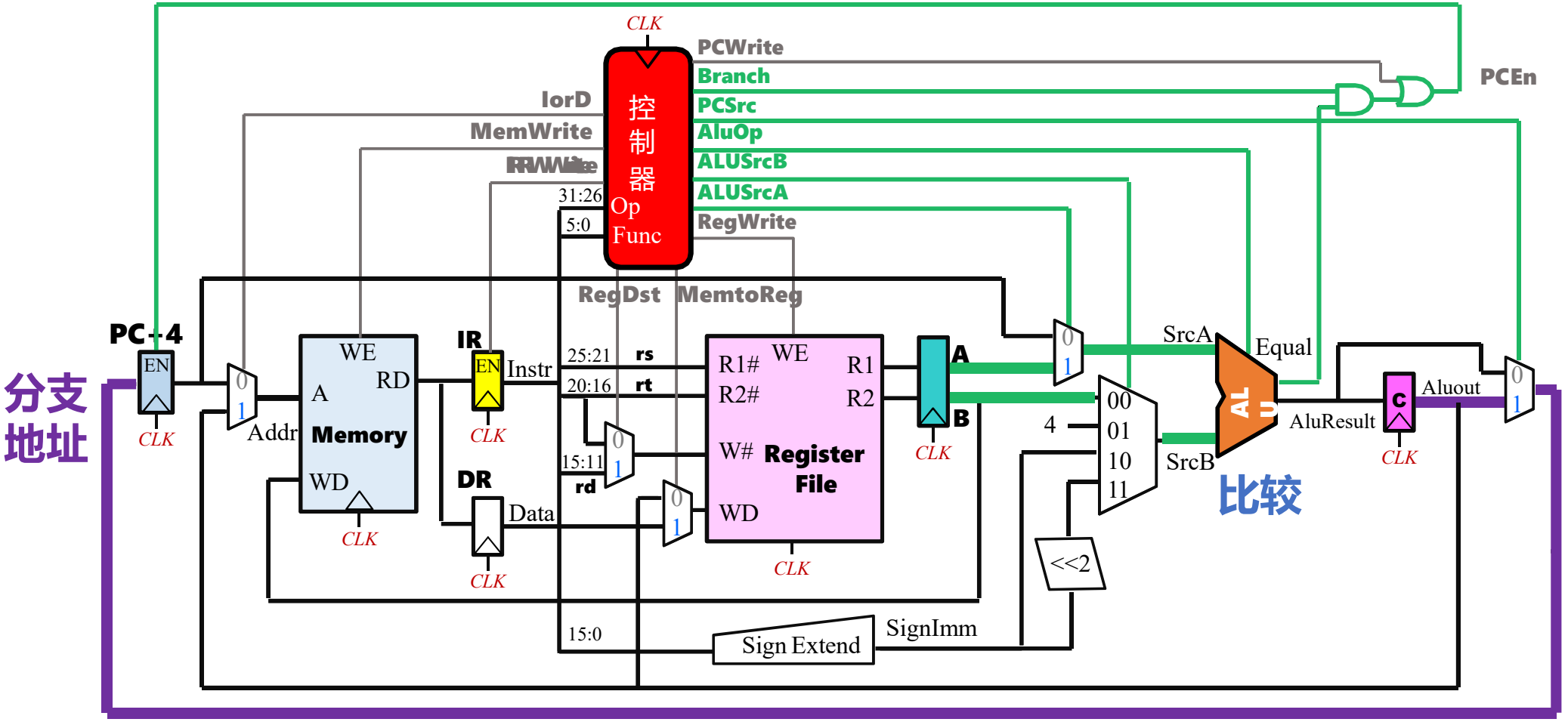
R型指令执行状态周期T3~T4



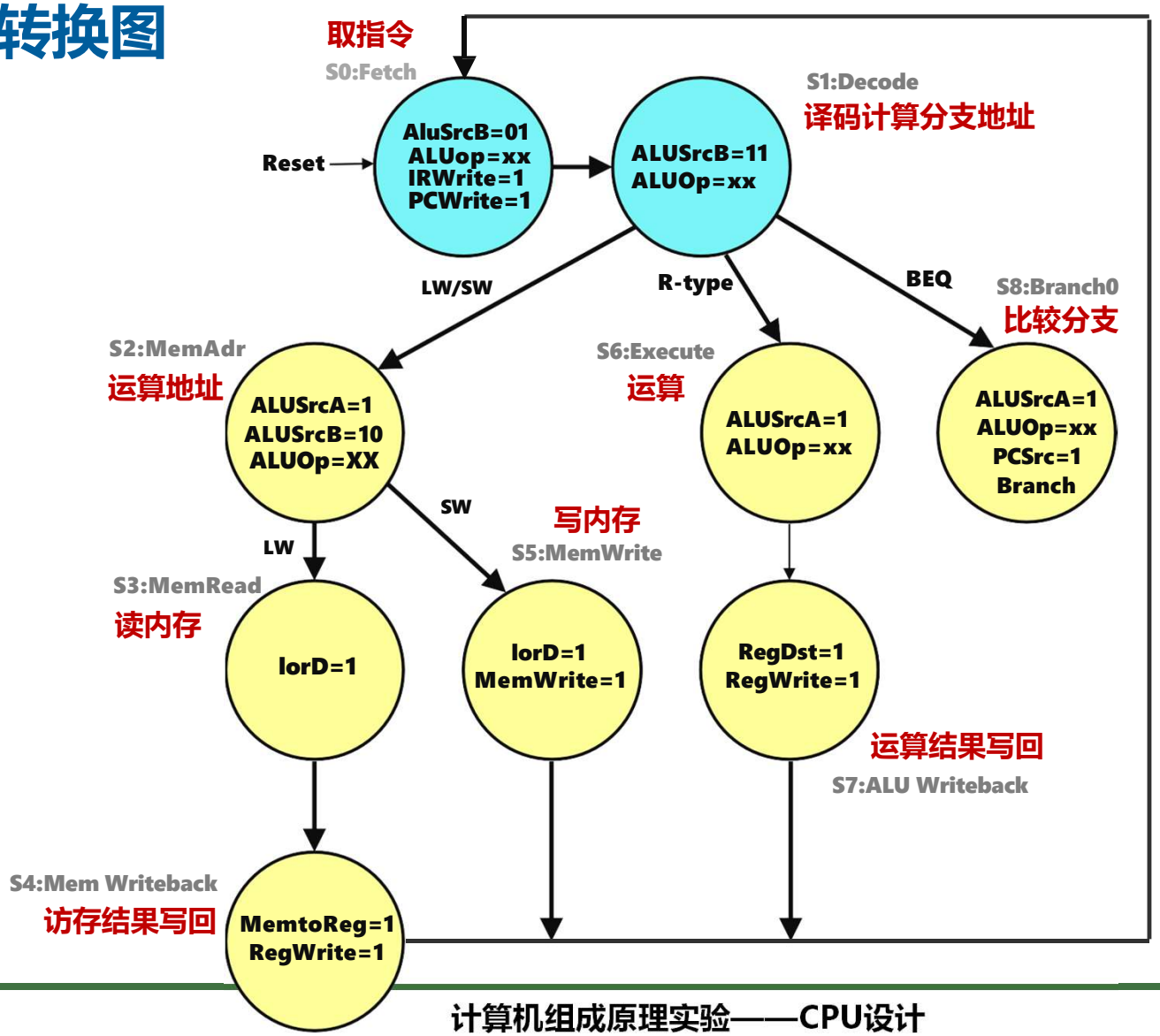
LW指令执行状态周期T3~T5



Beq指令执行状态周期T3



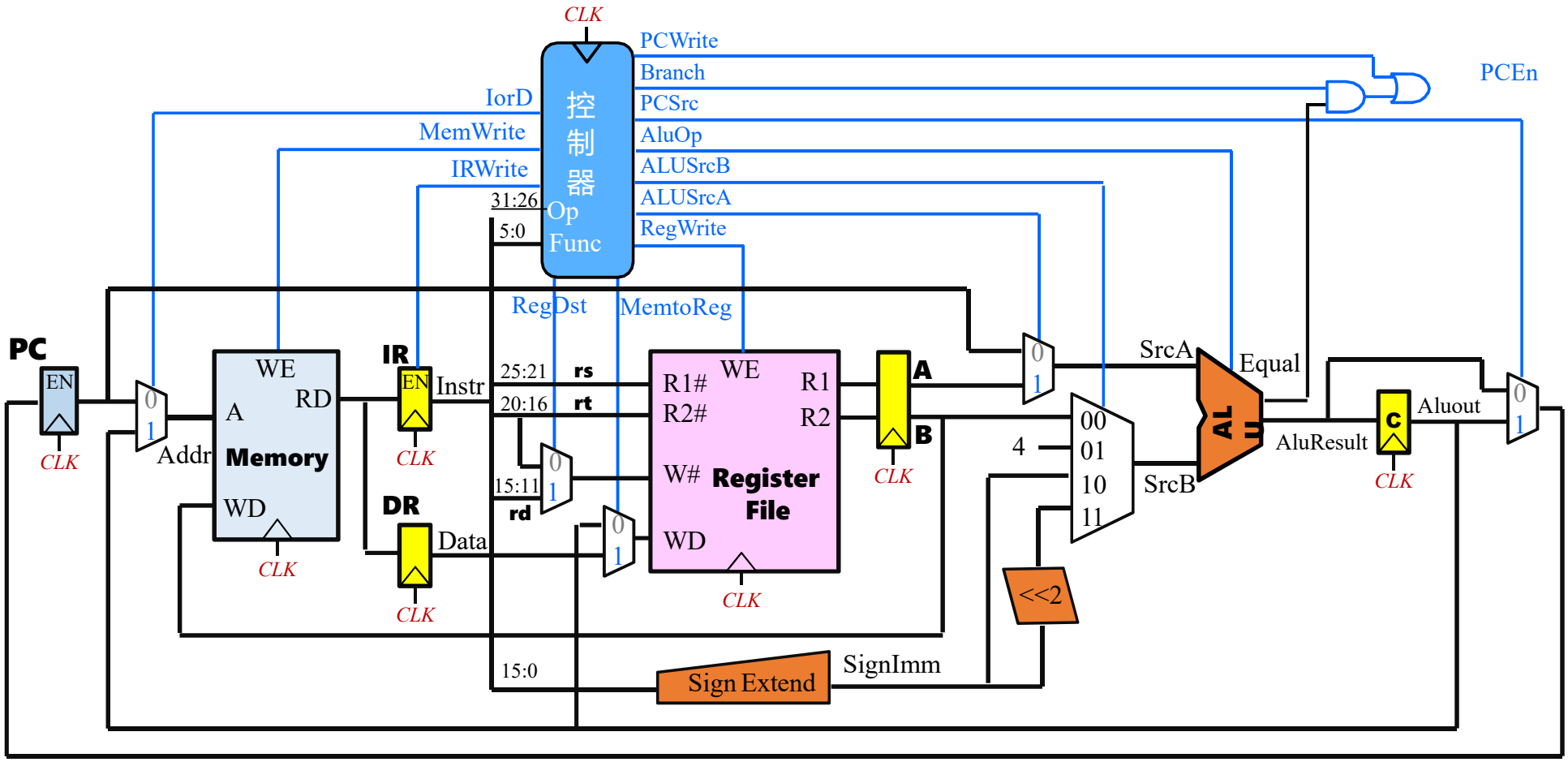
多周期状态转换图


取指令
S0:Fetch
AluSrcB=01
ALUop=xx
IRWrite=1
PCWrite=1
Reset
S1:Decode
译码计算分支地址
ALUSrcB=11
ALUOp=xx
LW/SW
R-type
BEQ
S8:Branch0
比较分支
ALUSrcA=1
ALUOp=xx
PCSrc=1
Branch
S2:MemAdr
运算地址
ALUSrcA=1
ALUSrcB=10
ALUOp=XX
S6:Execute
运算
ALUSrcA=1
ALUOp=xx
SW
写内存
S5:MemWrite
LW
S3:MemRead
读内存
IorD=1
IorD=1
MemWrite=1
RegDst=1
RegWrite=1
运算结果写回
S7:ALU Writeback
S4:Mem Writeback
访存结果写回
MemtoReg=1
RegWrite=1

核心指令集8条 （可实现内存区域冒泡排序）

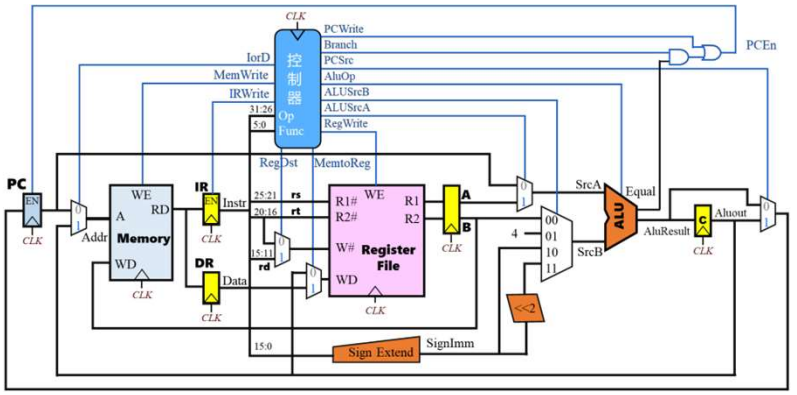
#	MIPS指令	RTL功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt] 溢出时产生异常，且不修改R[$rd]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt] 小于置1，有符号比较
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm) 溢出产生异常
4	lw $rt,imm($rs)	R[$rt]←$Mem_{4B}$(R[$rs]+$SignExt_{16b}$(imm))
5	sw $rt,imm($rs)	Mem_{4B}(R[$rs]+$SignExt_{16b}$(imm))←R[$rt]
6	beq $rs,$rt,imm	if(R[$rs] = R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
8	syscall	系统调用，这里用于停机

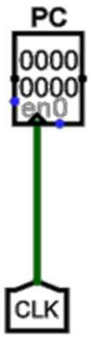
多周期MIPS CPU数据通路参考

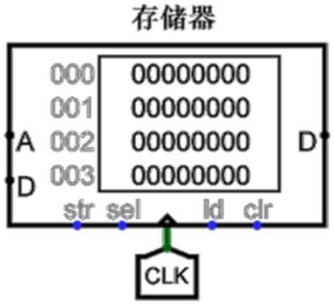


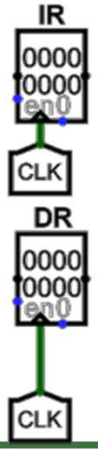
步骤1：构建多周期MIPS CPU数据通路

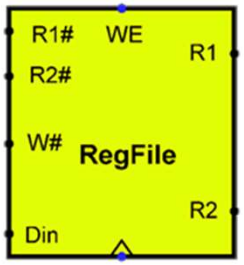
- 在MIPS多周期CPU子电路中，利用如下组件构建CPU数据通路
  - PC、MEM、IR、DR、RegFile、ALU、Controller

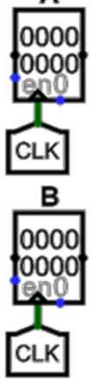


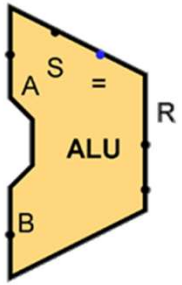


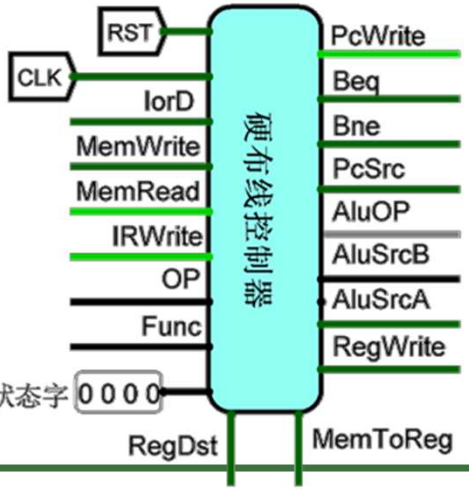






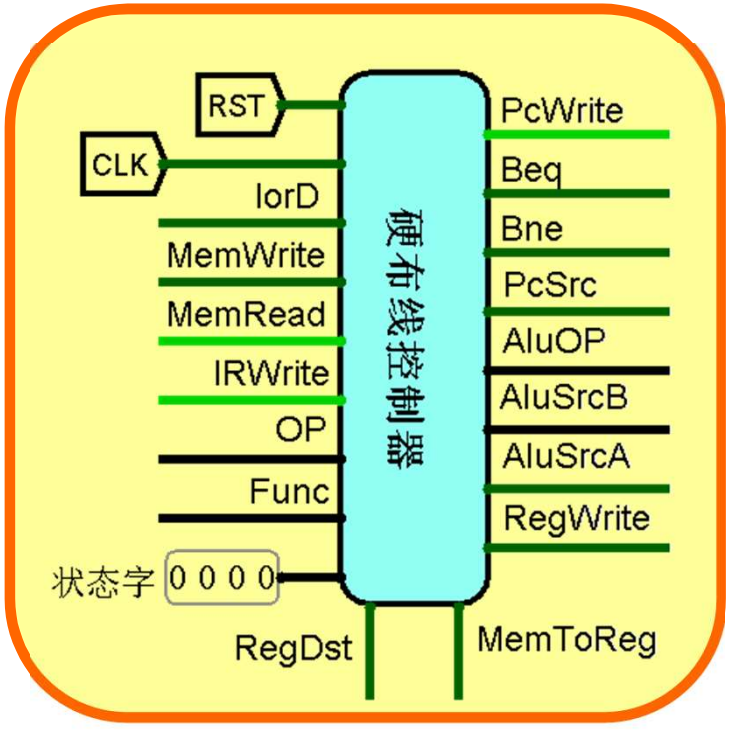






步骤2：设计控制器

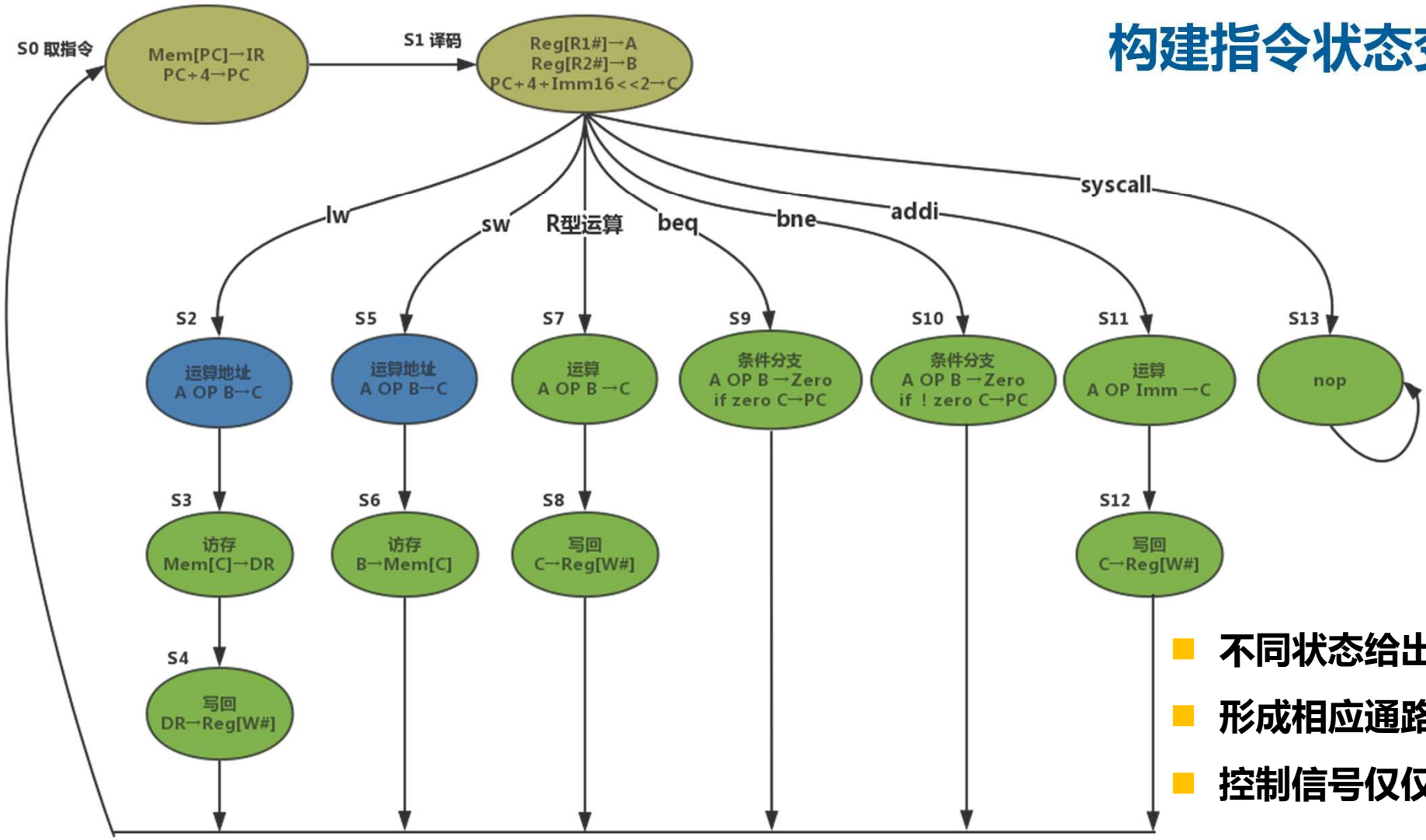
- 输入信号
  - 指令字Opcode，Func字段（12位）
  - 时钟信号、复位信号
- 输出信号
  - 多路选择器选择信号
  - 内存访问控制信号
  - 寄存器写使能信号
  - 运算器控制信号、指令译码信号



控制信号功能说明 （8条核心指令集）

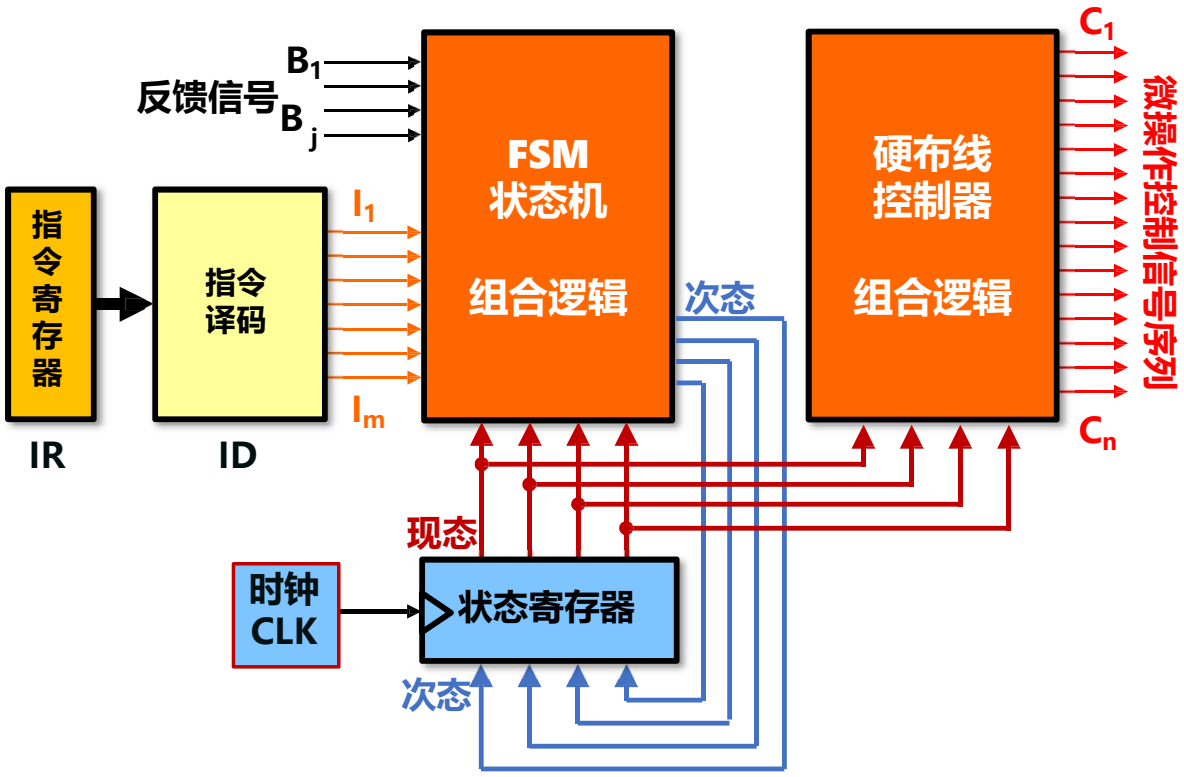
#	控制信号	信号说明	产生条件
1	PCWrite	PC写使能控制	取指令周期，分支指令执行
2	IorD	指令还是数据	0表示指令，1表示数据
3	IRwrite	指令寄存器写使能	高电平有效
4	MemWrite	写内存控制信号	sw指令
5	MemRead	读内存控制信号	lw指令 取指令
6	Beq	Beq指令译码信号	Beq指令
7	Bne	Bne指令译码信号	Bne指令
8	PcSrc	PC输入来源	顺序寻址还是跳跃寻址
9	AluOP	运算器操作控制符 4位	ALU_Control控制，00加，01减，10由Funct定
10	AluSrcA	运算器第一输入选择	
11	AluSrcB	运算器第二输入选择	Lw指令，sw指令，addi
12	RegWrite	寄存器写使能控制信号	寄存器写回信号
13	RegDst	写入寄存器选择控制信号	R型指令
14	MemToReg	写入寄存器的数据来自存储器	lw指令

构建指令状态变换图



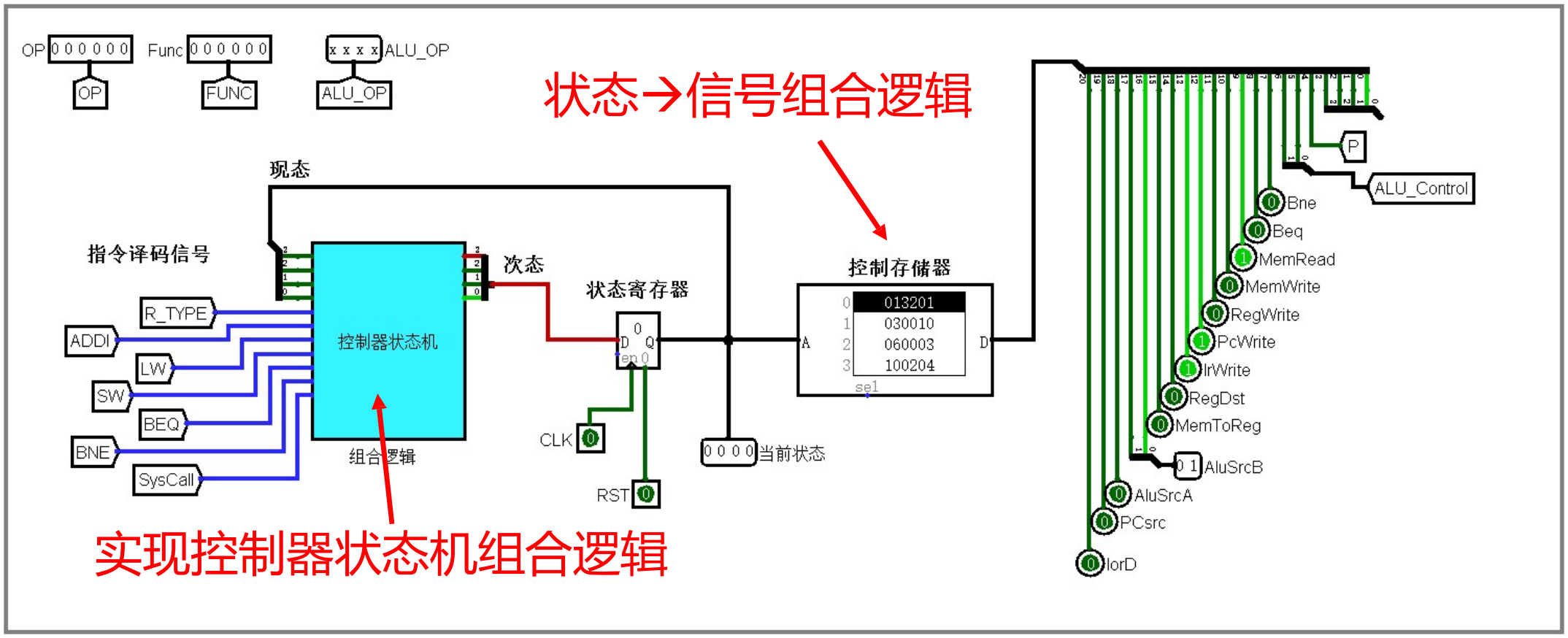
- 不同状态给出不同信号
- 形成相应通路
- 控制信号仅仅与状态有关

设计控制器



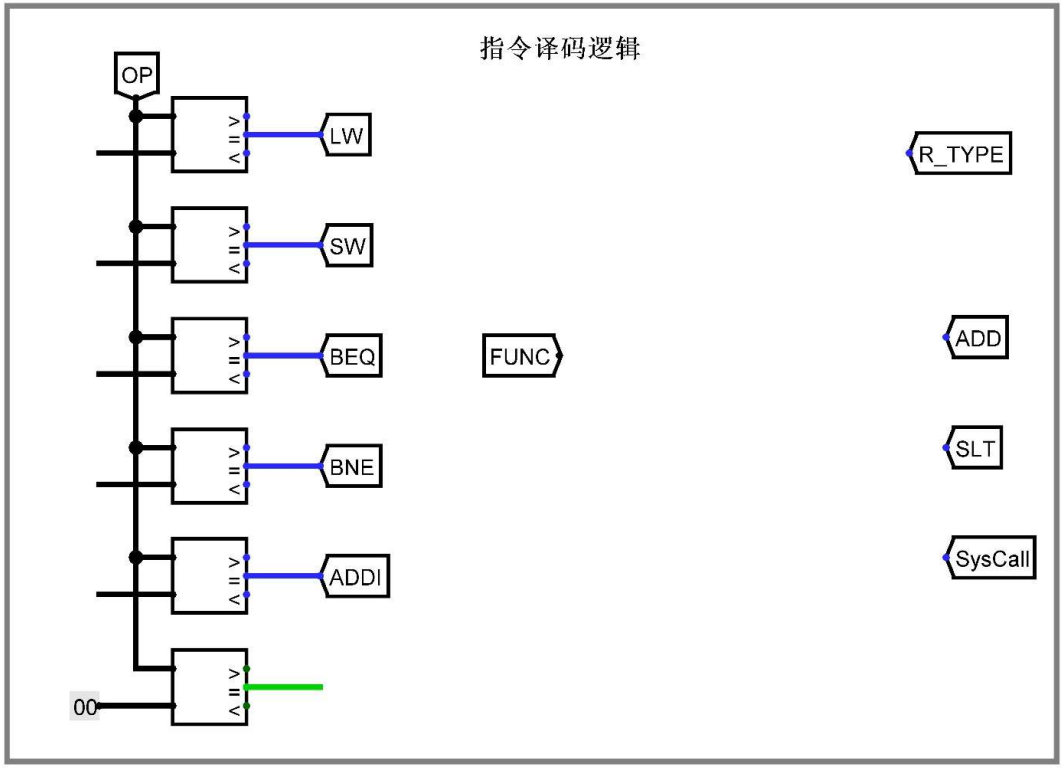
机器指令字 → 控制器信号序列

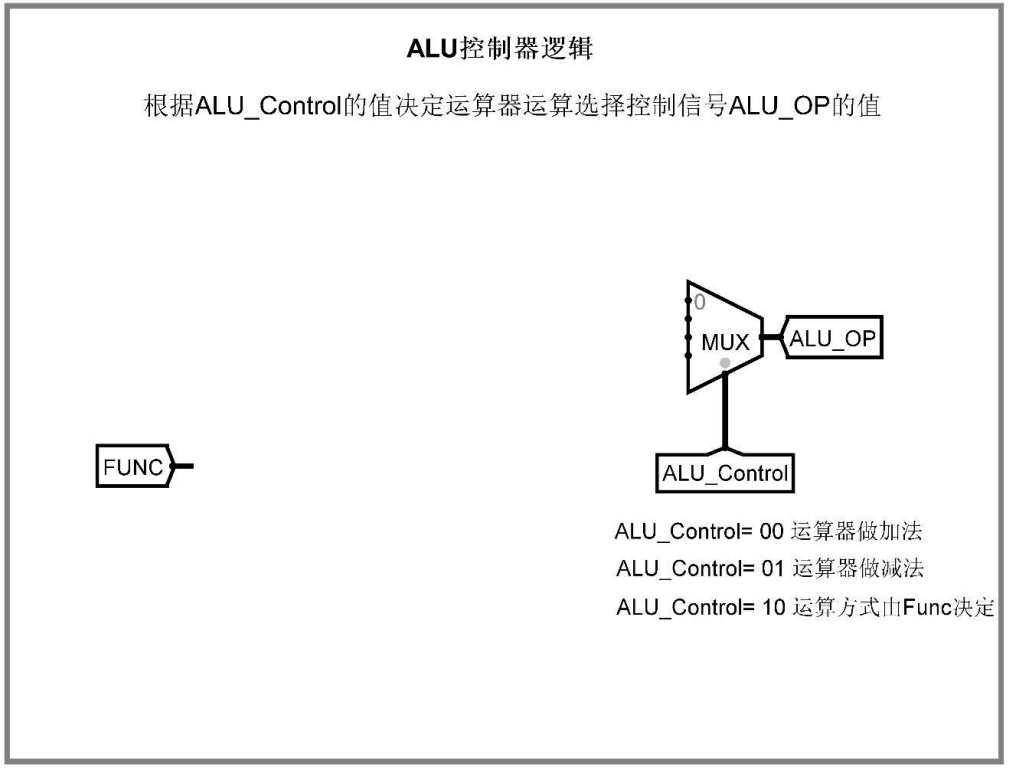
控制器内部架构


OP 0 0 0 0 0 0
OP
Func 0 0 0 0 0 0
FUNC
x x x x ALU_OP
ALU_OP
状态→信号组合逻辑
现态
指令译码信号
R_TYPE
ADDI
LW
SW
BEQ
BNE
SysCall
控制器状态机
组合逻辑
次态
状态寄存器
D Q
en 0
CLK 0
RST 0
0 0 0 0 当前状态
控制存储器
0 013201
1 030010
2 060003
3 100204
A
D
sel
P
ALU_Control
Bne
Beq
MemRead
MemWrite
RegWrite
PcWrite
IrWrite
RegDst
MemToReg
0 1 AluSrcB
AluSrcA
PCsrc
IorD
实现控制器状态机组合逻辑

完善控制器内部逻辑

- 首先完成如下电路逻辑： **指令译码、ALU控制**





状态机逻辑生成

- 填写状态转换表　　（输入：现态，指令译码信号，输出次态）
- 自动生成次态逻辑表达式

S3	S2	S1	S0	最小项表达式	N3	N2	N1	N0
~S3&	~S2&	~S1&	~S0&	~S3&~S2&~S1&~S0&R_Type&SW	~S3&~S2&~S1&~S0&R_Type&SW+			~S3&~S2&~S1&~S0&R_Type&SW+
~S3&	~S2&	~S1&	S0&	~S3&~S2&~S1&S0&R_Type		~S3&~S2&~S1&S0&R_Type+	~S3&~S2&~S1&S0&R_Type+	~S3&~S2&~S1&S0&R_Type+
~S3&	~S2&	~S1&	S0&	~S3&~S2&~S1&S0&LW			~S3&~S2&~S1&S0&LW+	
				逻辑表达式->>>	~S3&~S2&~S1&~S0&R_Type&SW	~S3&~S2&~S1&S0&R_Type	[illegible]	[illegible]

理想指令流水线设计

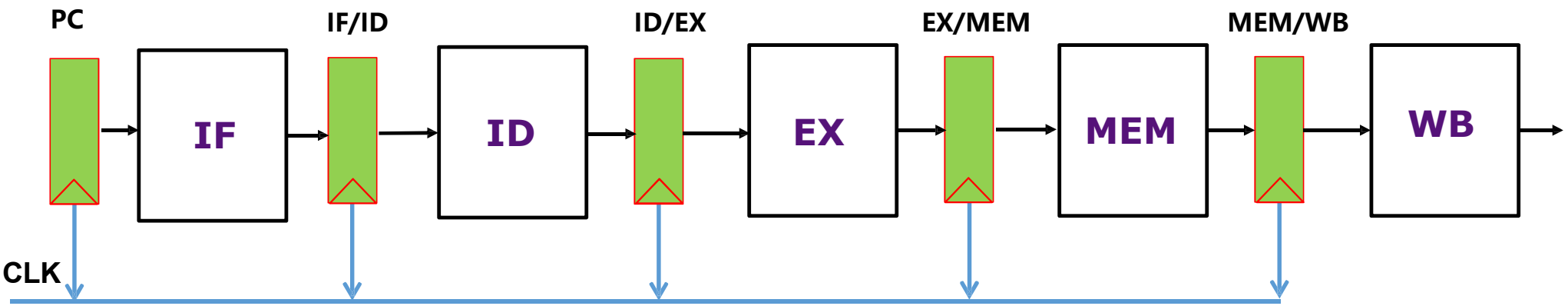
理想指令流水线设计

理想流水线关键问题

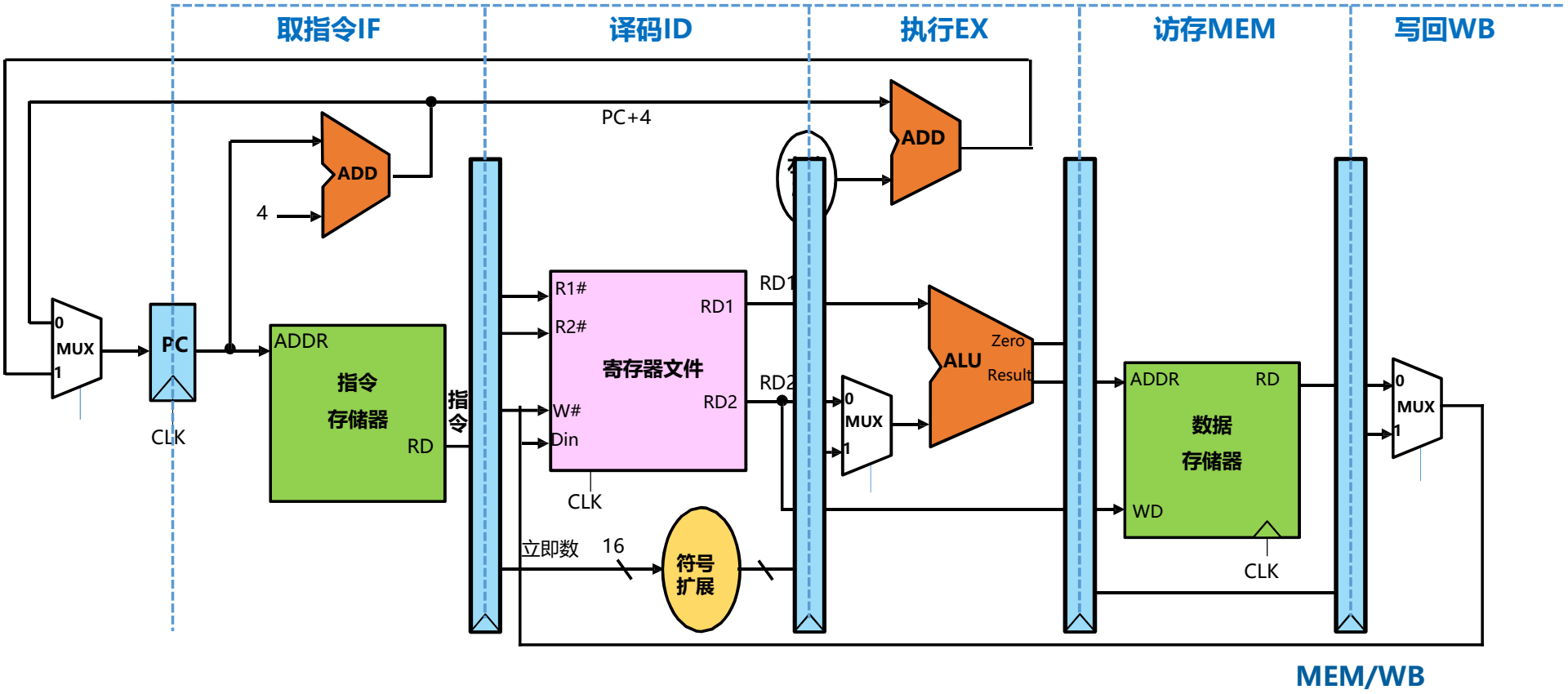
- **阶段数相同**：所有加工对象均通过同样的工序（阶段）
  - **不同指令阶段数不同，让所有指令均经过5个阶段，部分阶段可能不工作**
- **段时延相同**：各段传输延迟一致，不能有等待现象，取最慢的同步
  - **取指，访存段最慢，按访存时延进行流水同步**
- **无资源冲突**：不同阶段之间无共享资源，各段完全并发
  - **内存争用：哈佛结构　运算器争用：增设加法器**
- **无段间互锁**：进入流水线的对象不受其他阶段的影响
  - **多条指令间存在相关和依赖，仅运行无数据相关，无分支相关的程序**

理想流水线构建步骤

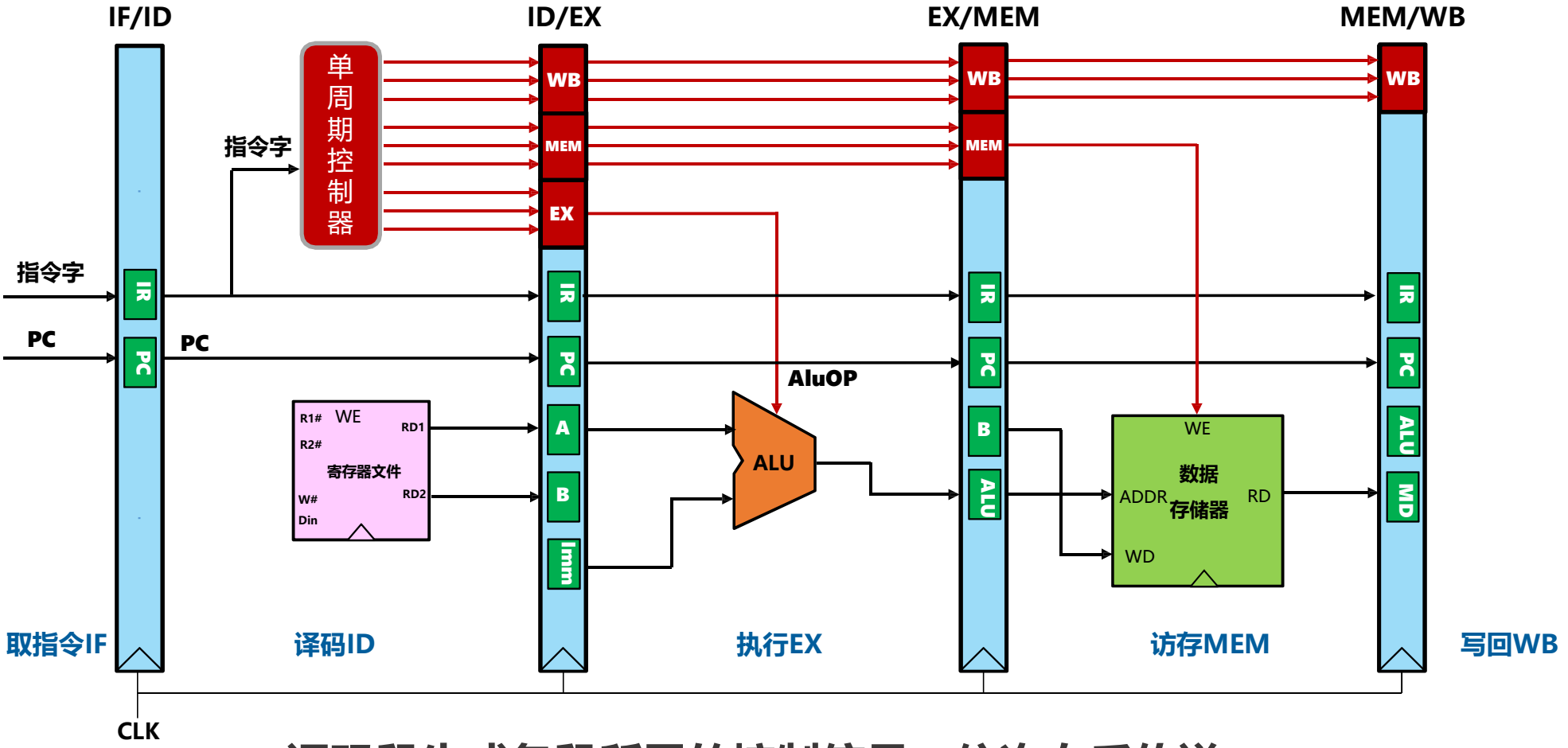
- 将单周期CPU细分成五个阶段：IF、ID、EX、MEM、WB
- 段间设置流水接口部件
  - 接口部件本质是寄存器，锁存前段加工完成的数据
  - 通过接口传递与指令相关的数据、控制、反馈信息
  - 后段对数据的加工处理依赖于前段接口传递过来的信息



单周期CPU → 五段流水线



5段指令流水线数据与信号传递

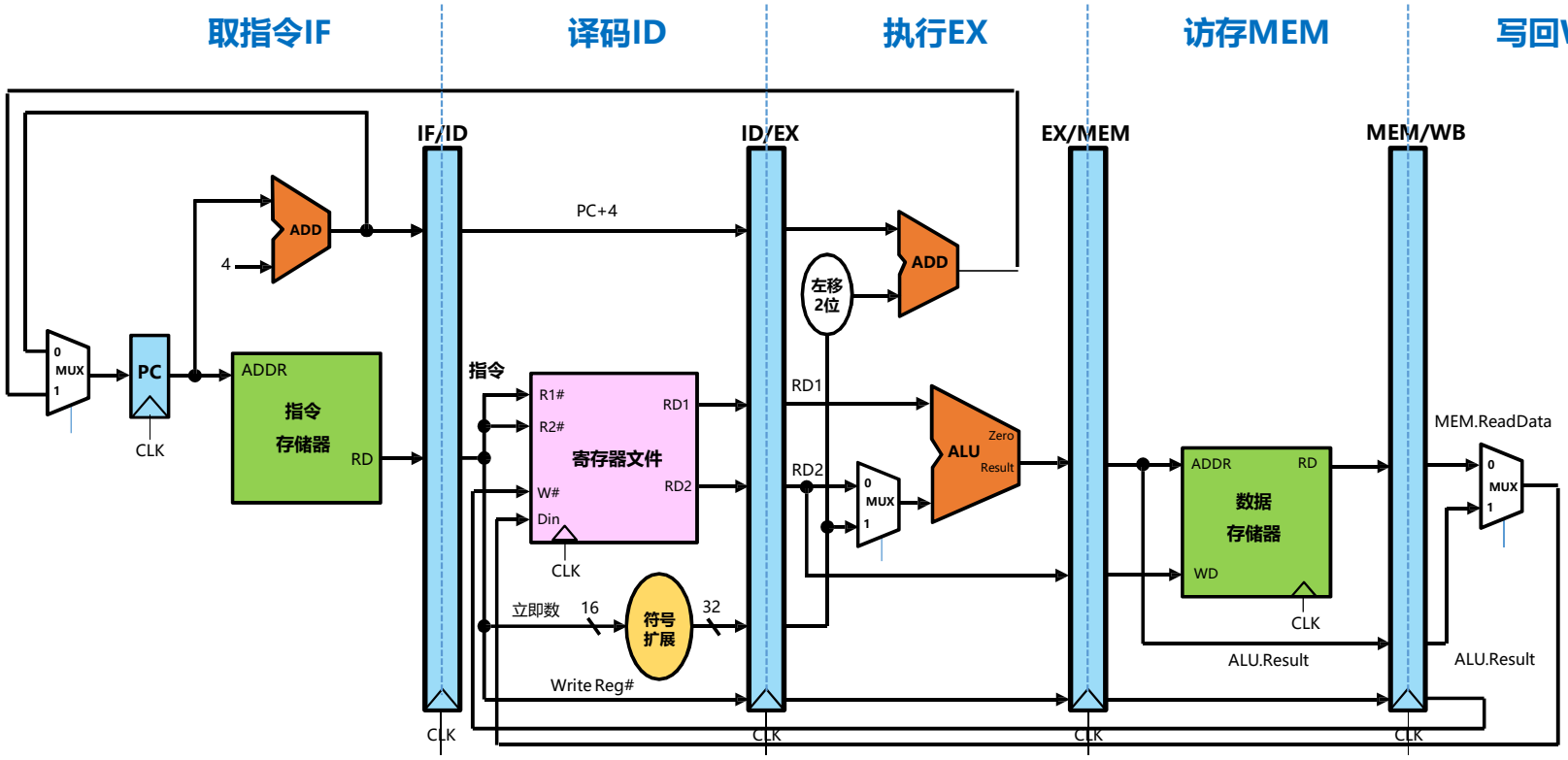


- 译码段生成各段所需的控制信号，依次向后传递

流水接口部件内部结构

- **本质是若干寄存器**
  - 锁存数据与信号
  - 控制端
    - 同步清零（插入气泡），使能端（stall信号）

MIPS五段流水数据通路



- **内存争用**：哈佛结构；　**寄存器争用**：寄存器读，寄存器写？

理想指令流水线设计实验

- **实验任务**
  - **将单周期CPU修改为理想流水线**
  - 设计流水接口子电路　IF/ID、**ID/EX**、EX/MEM、MEM/WB
  - 将单周期数据通路 → 流水数据通路

流水线分支相关处理

指令流水线的相关、冲突、冒险（hazard）

- **资源相关**
  - **争用主存**：IF段取指令、ID段取操作数
  - **争用ALU**：多周期方案中计算PC、分支地址，运算指令
  - **解决方案**：增加部件消除
- **分支相关**
  - 控制IF段进行分支跳转
  - 提前取出的指令作废，流水线清空
- **数据相关**
  - 指令操作数依赖于前一条指令的执行结果
  - 引起流水线停顿直到数据写回

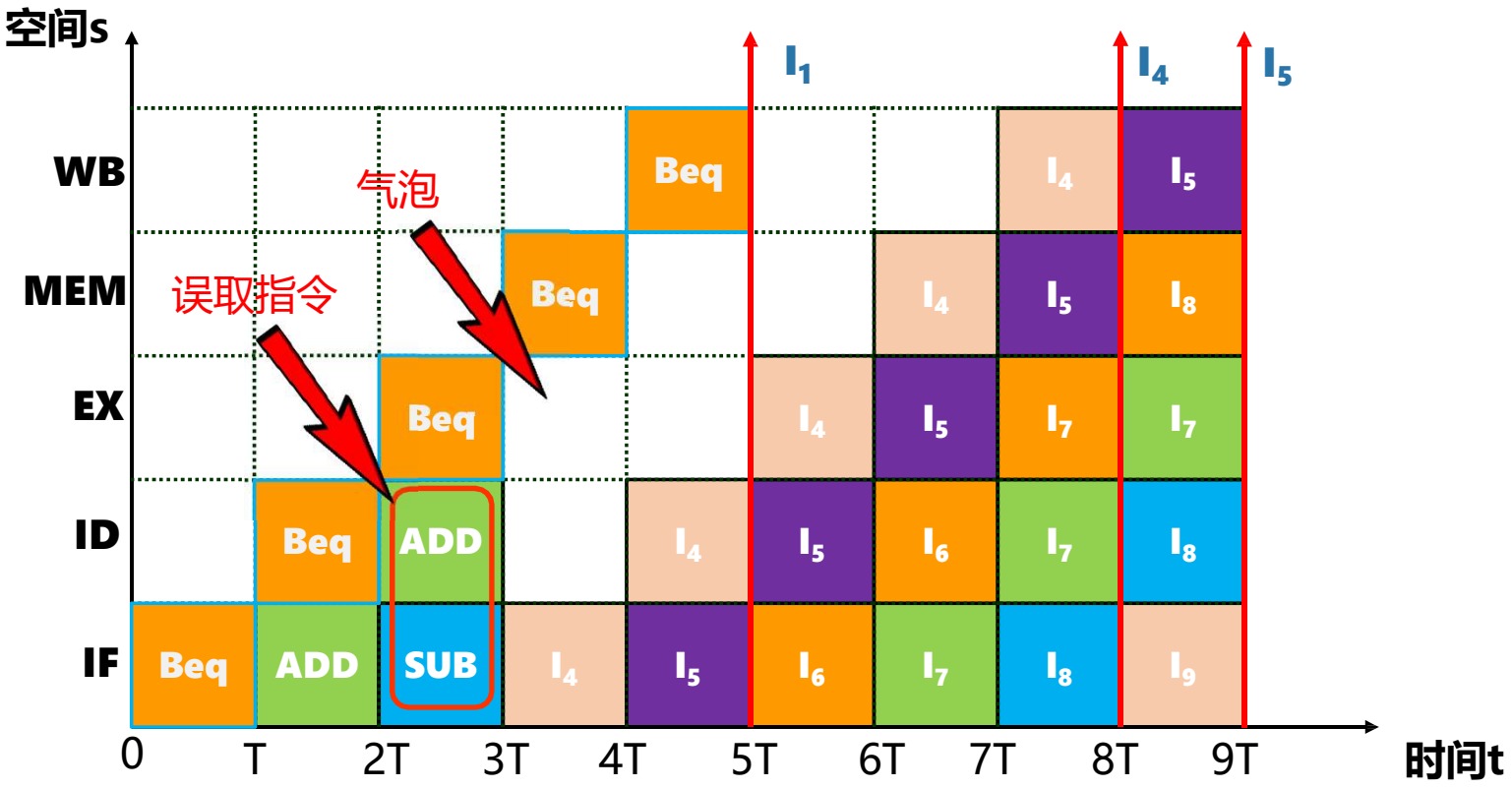
```
ADD $s1, $s2, $s3
ADD $s4, $s1, $s3
```

分支指令

- 无条件分支
- 有条件分支

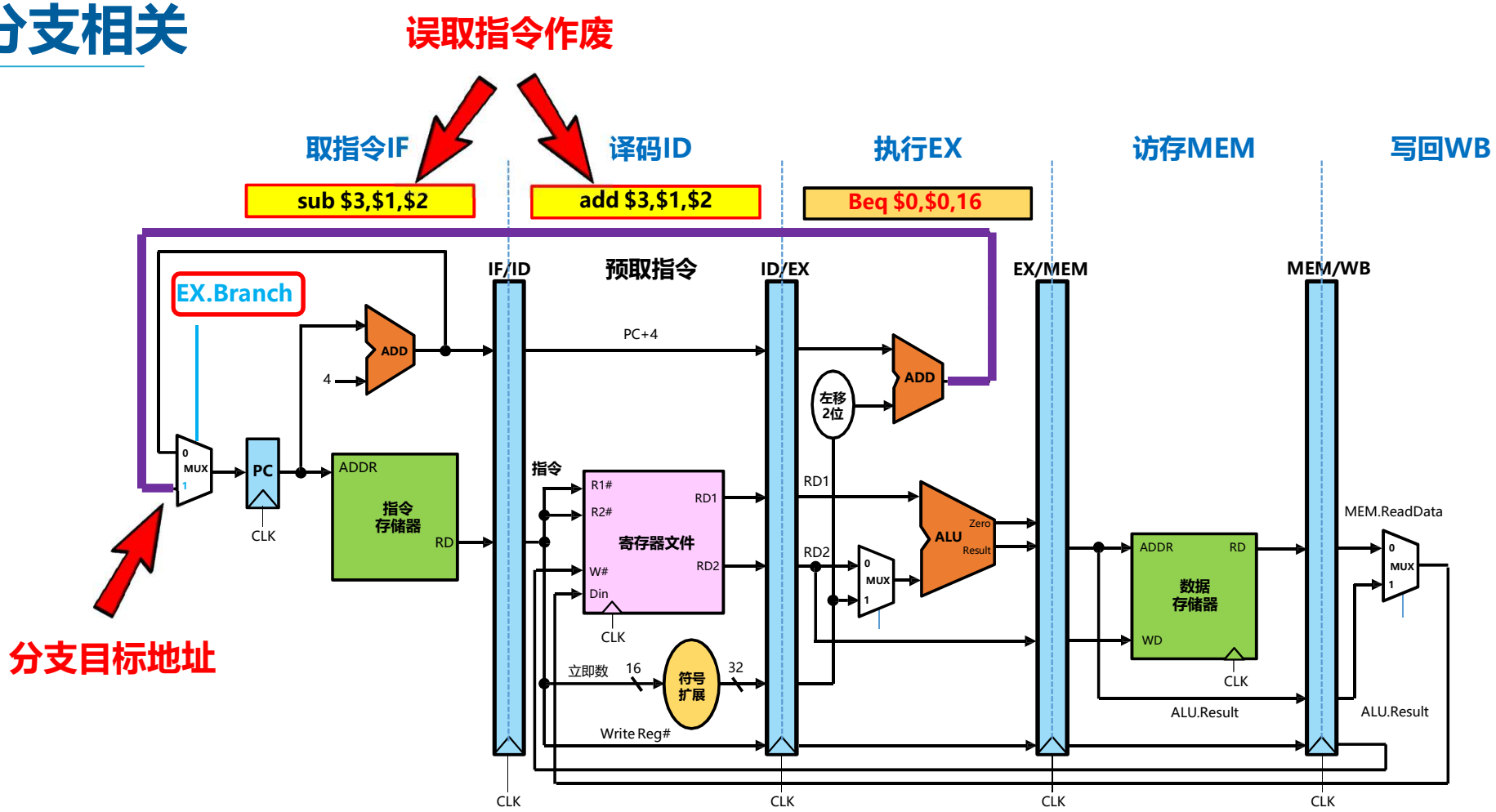
#	指令类型	指令	调用格式
1	R型指令	JR、JALR	#JR rs
2	I型指令	BEQ、BNE BGTZ(>0)，BGEZ(≥0)，BLTZ(<0)，BLEZ（≤0)	#BEQ rs, rt, offset16 #BGEZ rs, offset16
3	J型指令	J、JAL、	#JAL target26

分支相关流水线时空图



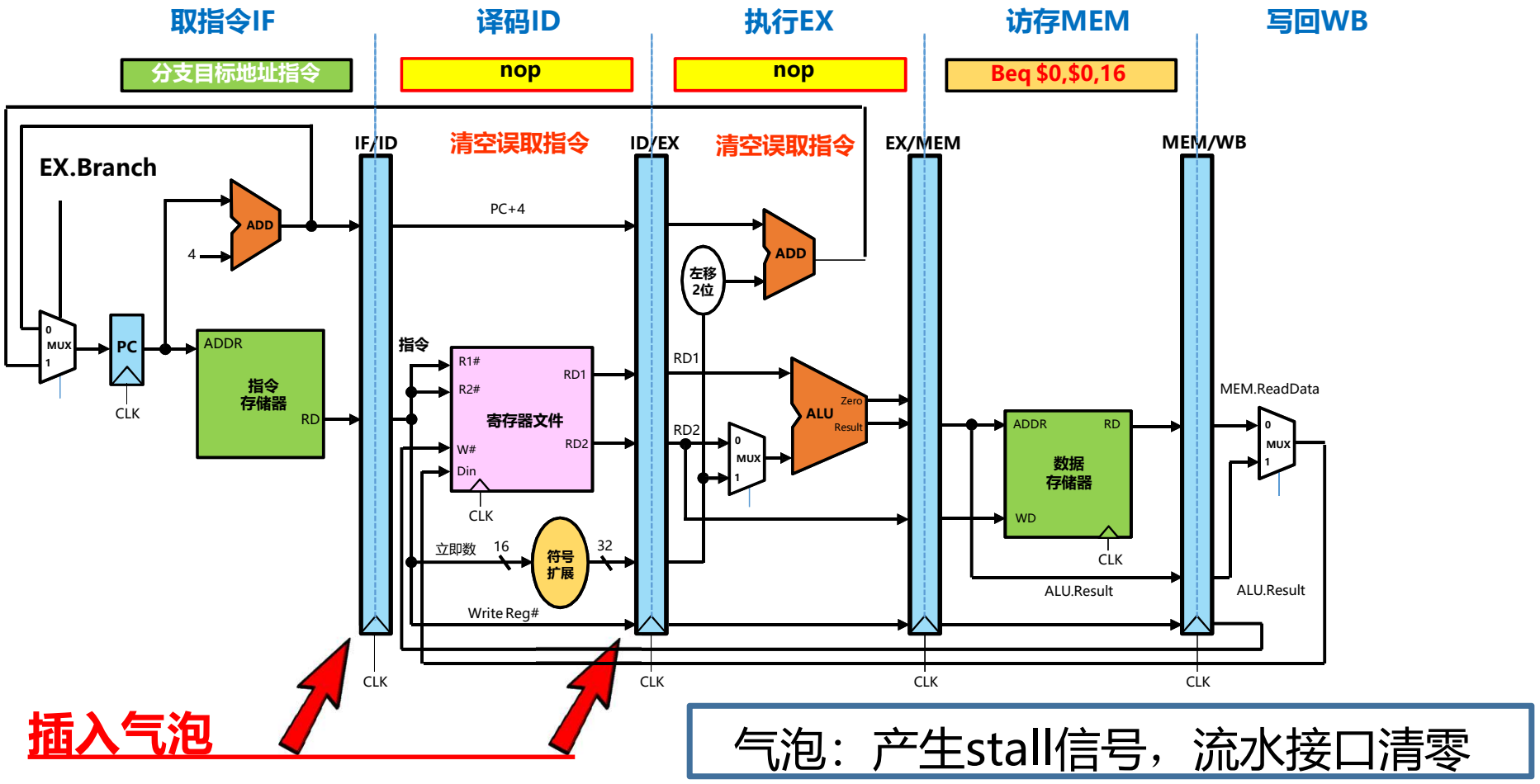
误取两条指令作废，流水线损失了2T

分支相关

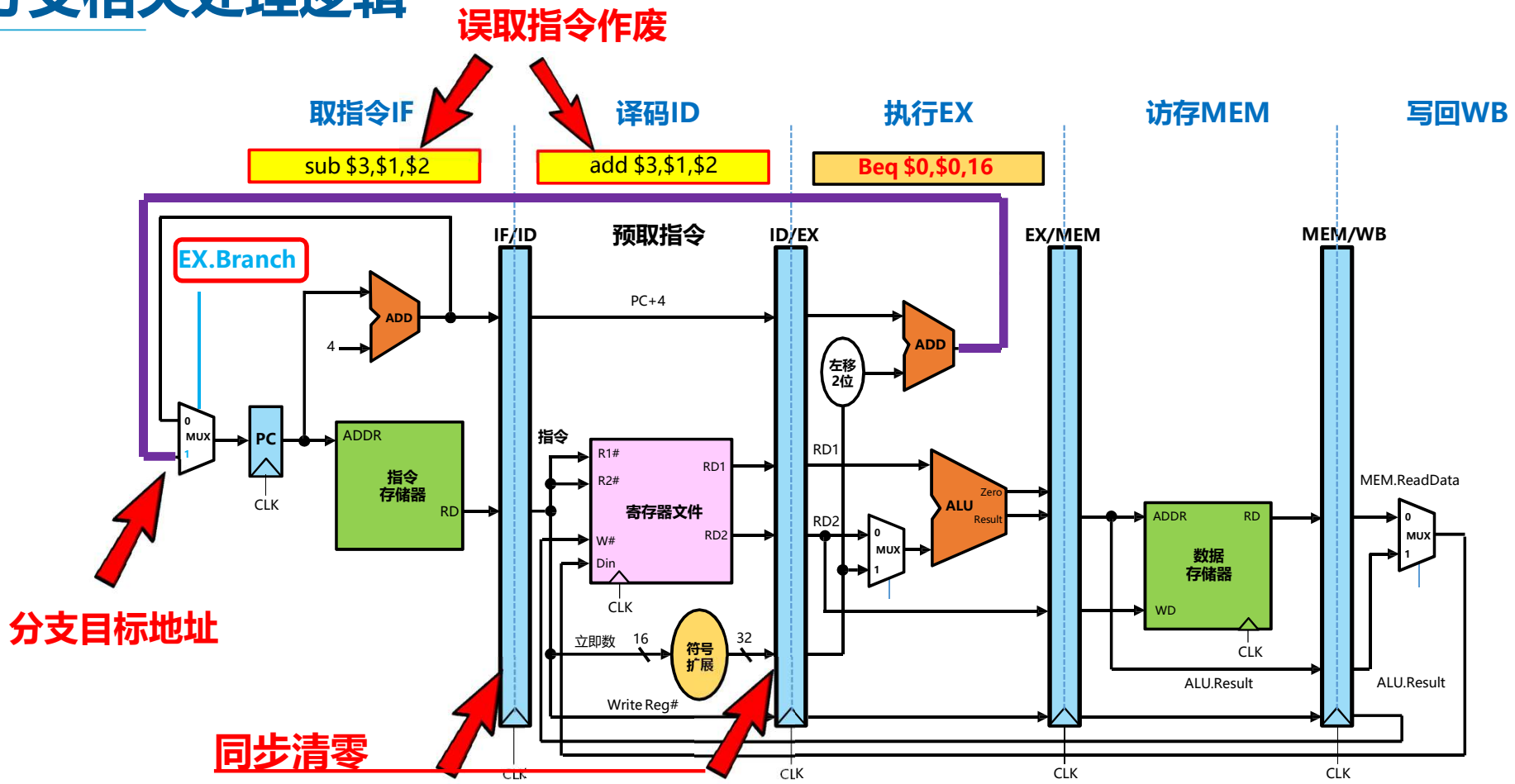


如何清除误取指令?

插入气泡


取指令IF
译码ID
执行EX
访存MEM
写回WB
分支目标地址指令
nop
nop
Beq $0,$0,16
EX.Branch
IF/ID
清空误取指令
ID/EX
清空误取指令
EX/MEM
MEM/WB
ADD
4
PC+4
左移2位
0 MUX 1
PC
CLK
ADDR
指令存储器
RD
指令
R1#
R2#
W#
Din
寄存器文件
RD1
RD2
CLK
RD1
RD2
ALU
Zero
Result
立即数
16
符号扩展
32
Write Reg#
ADDR
RD
数据存储器
WD
CLK
ALU.Result
MEM.ReadData
ALU.Result
CLK
CLK
CLK
CLK
插入气泡
气泡：产生stall信号，流水接口清零

分支相关处理逻辑

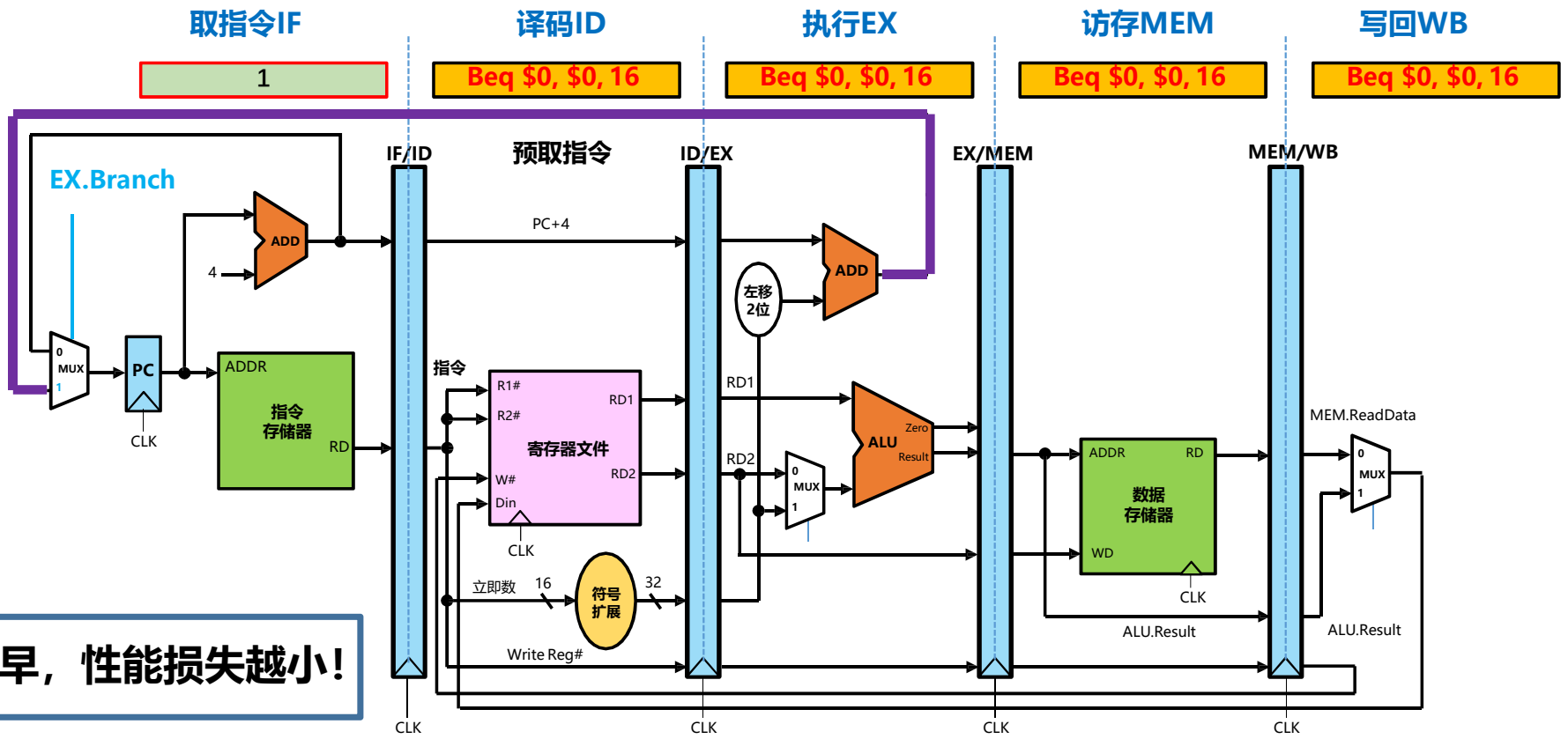


分支逻辑：EX段选择IF段PC地址,并给出IF/ID,ID/EX清零信号

分支相关处理策略

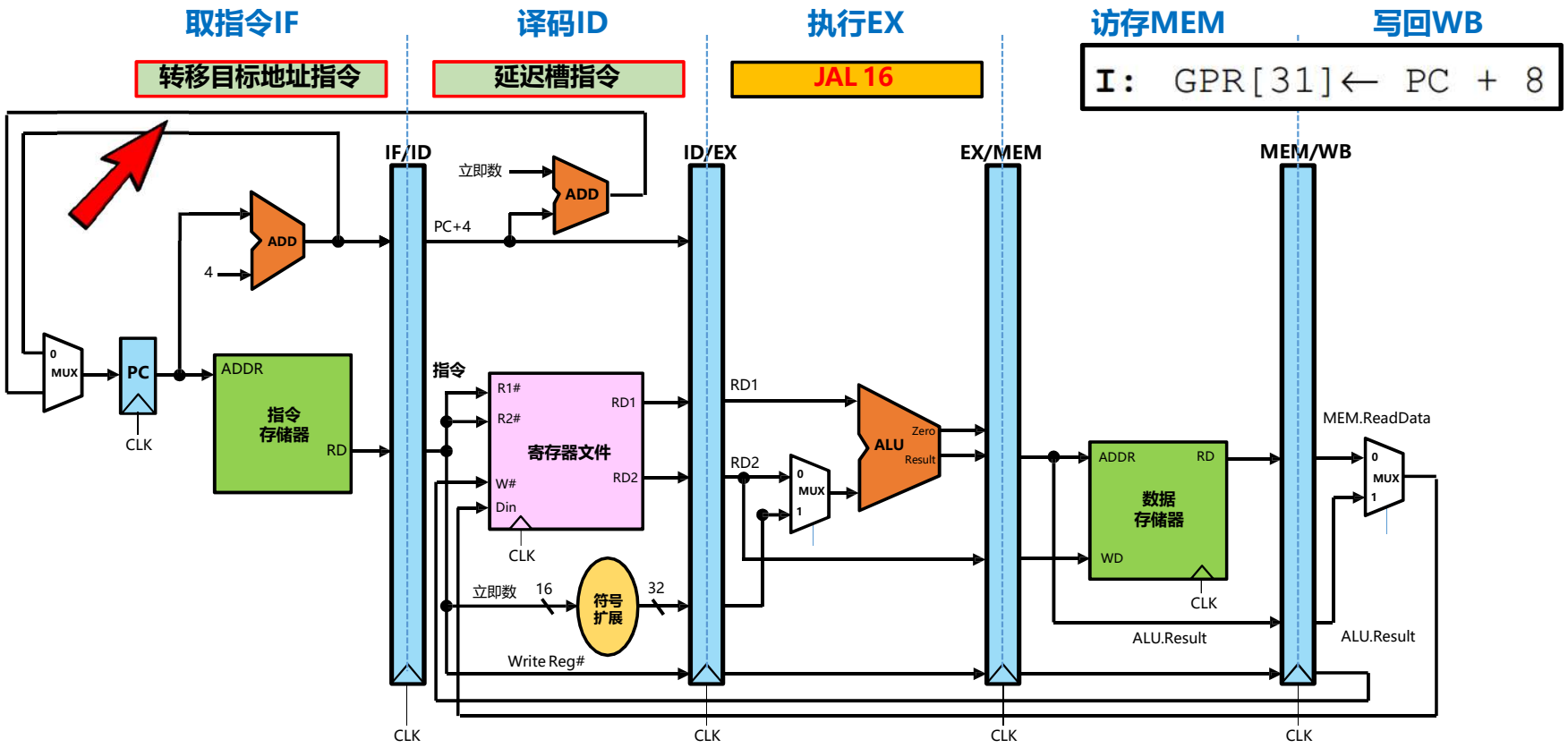
- 分支指令放在那一段执行?

分支指令，是否能在IF段执行?



分支执行越早，性能损失越小!

MIPS延迟槽技术



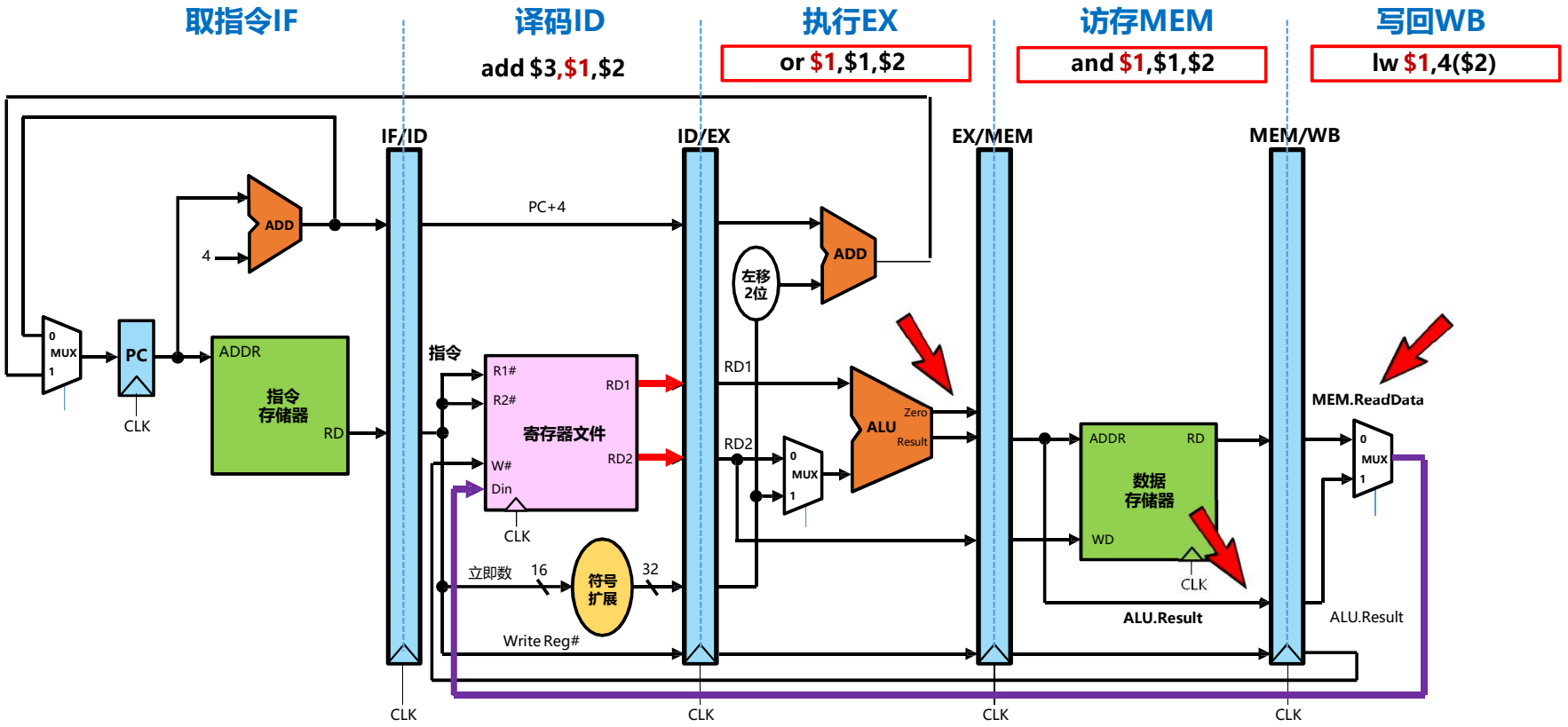
- 分支在ID段执行,相邻指令为延迟槽，无论是否跳转，延迟槽指令必须执行

分支相关总结

- **指令跳转**
  - IF段重新取新的指令
- **清除误取指令**
  - IF/ID、ID/EX段给出同步清零信号
- **分支指令执行阶段?**
  - 越早执行，性能损失越小
  - MIPS中通常为ID段执行，为了简化中断，重定向机制，可在EX段执行
- **分支延迟槽技术**
  - 配合ID段执行分支指令，彻底消除分支带来的流水性能损失
  - 如何将有用的指令载入延迟槽比较关键
  - X86中没有分支延迟槽，采用动态分支预测技术

流水线数据相关处理

数据相关

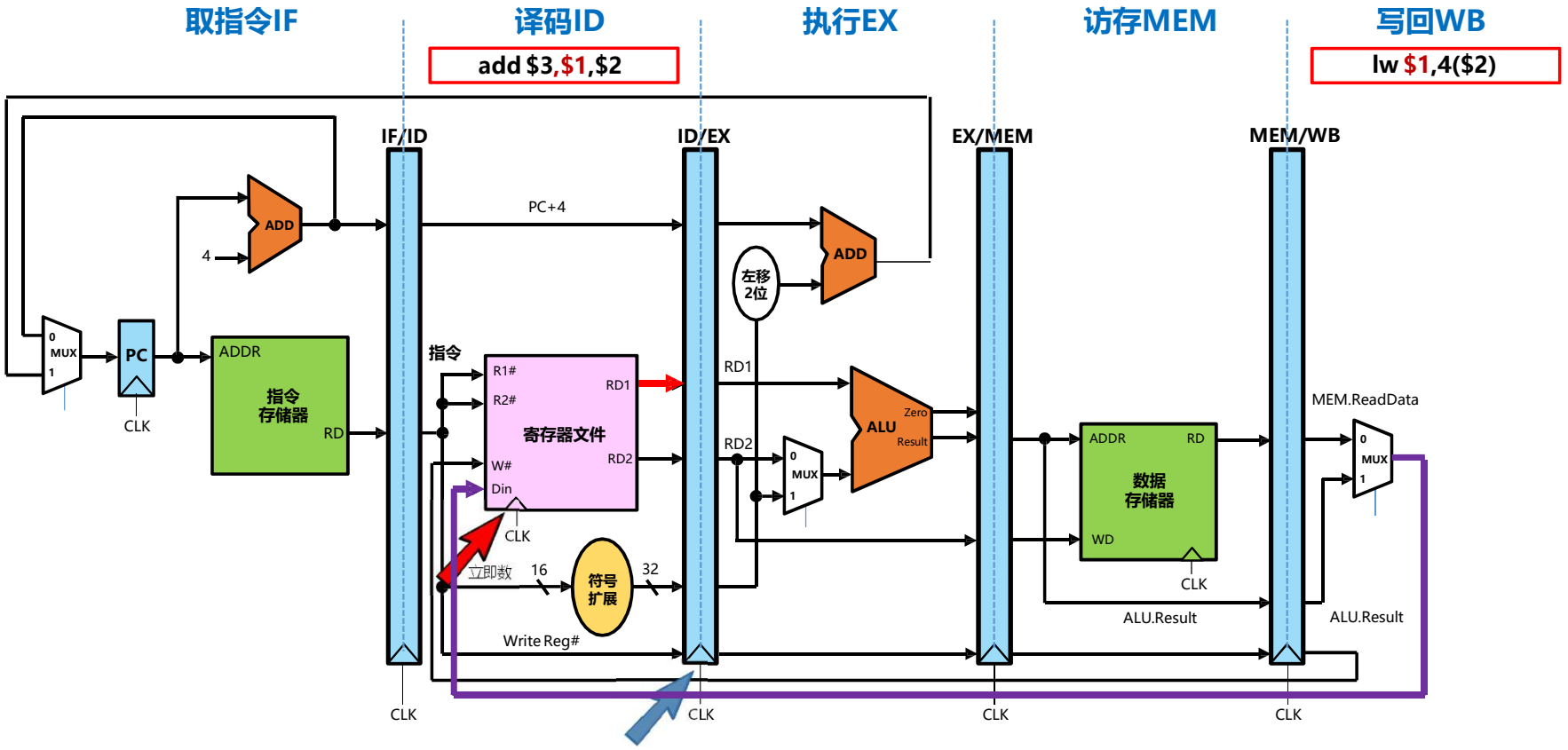


- ID段所需数据可能还未及时写回，涉及EX、MEM、WB段3条指令

数据相关处理机制

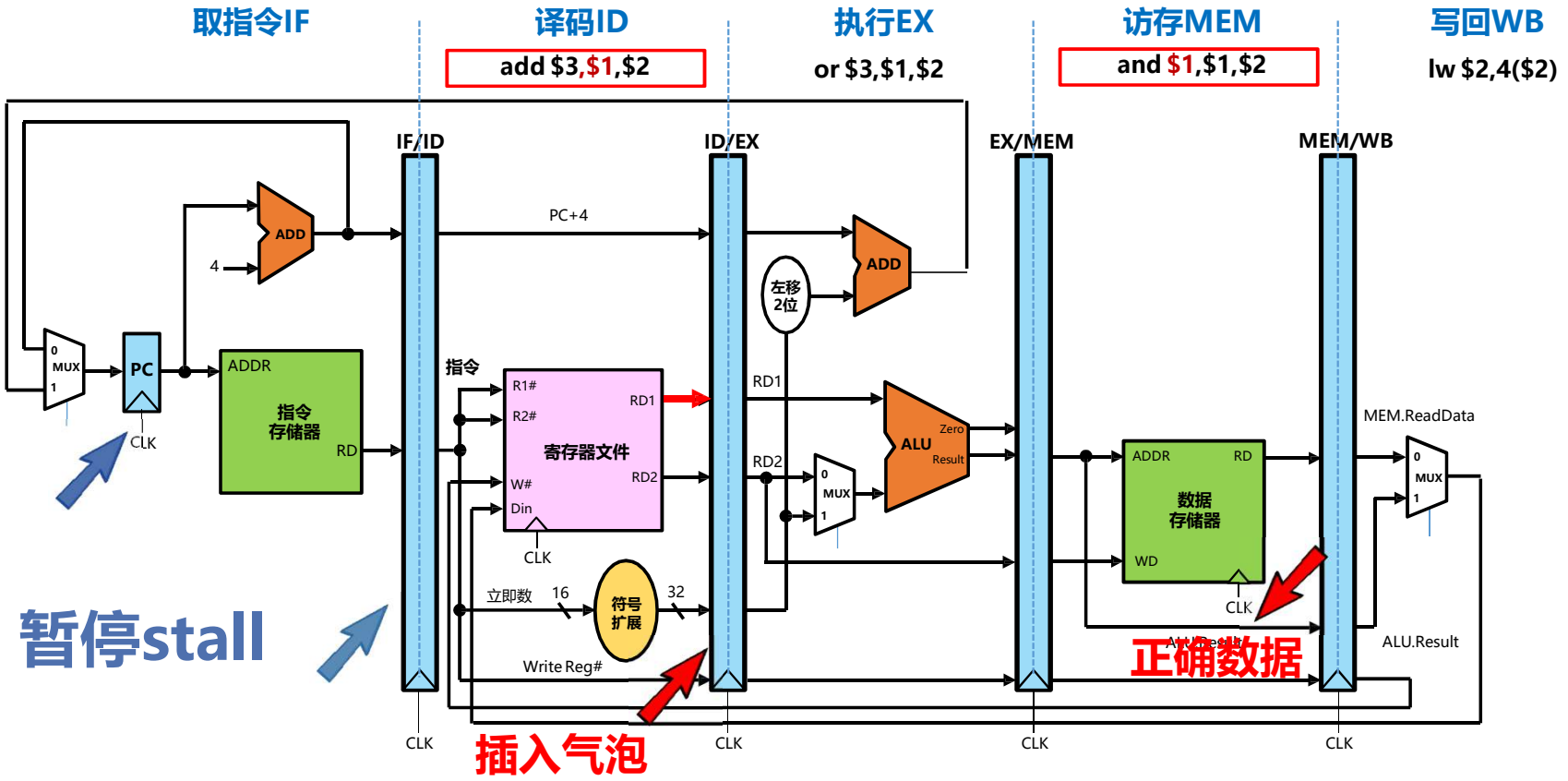
- **软件方法（编译器完成）**
  - 插入空指令
  - 调整程序顺序，使相关性在流水线中消失
- **硬件方法**
  - **插入气泡（空操作）**
    - 向后段插入气泡（接口信号清零）
    - 向前给出阻塞信号（流水线停顿）避免当前指令被新指令取代
  - **数据重定向**bypass（数据旁路）
    - ◆将后端处理后的数据（还没来得及写回）重定向
    - ◆数据在哪就从哪送到运算器

ID段与WB段数据相关消除



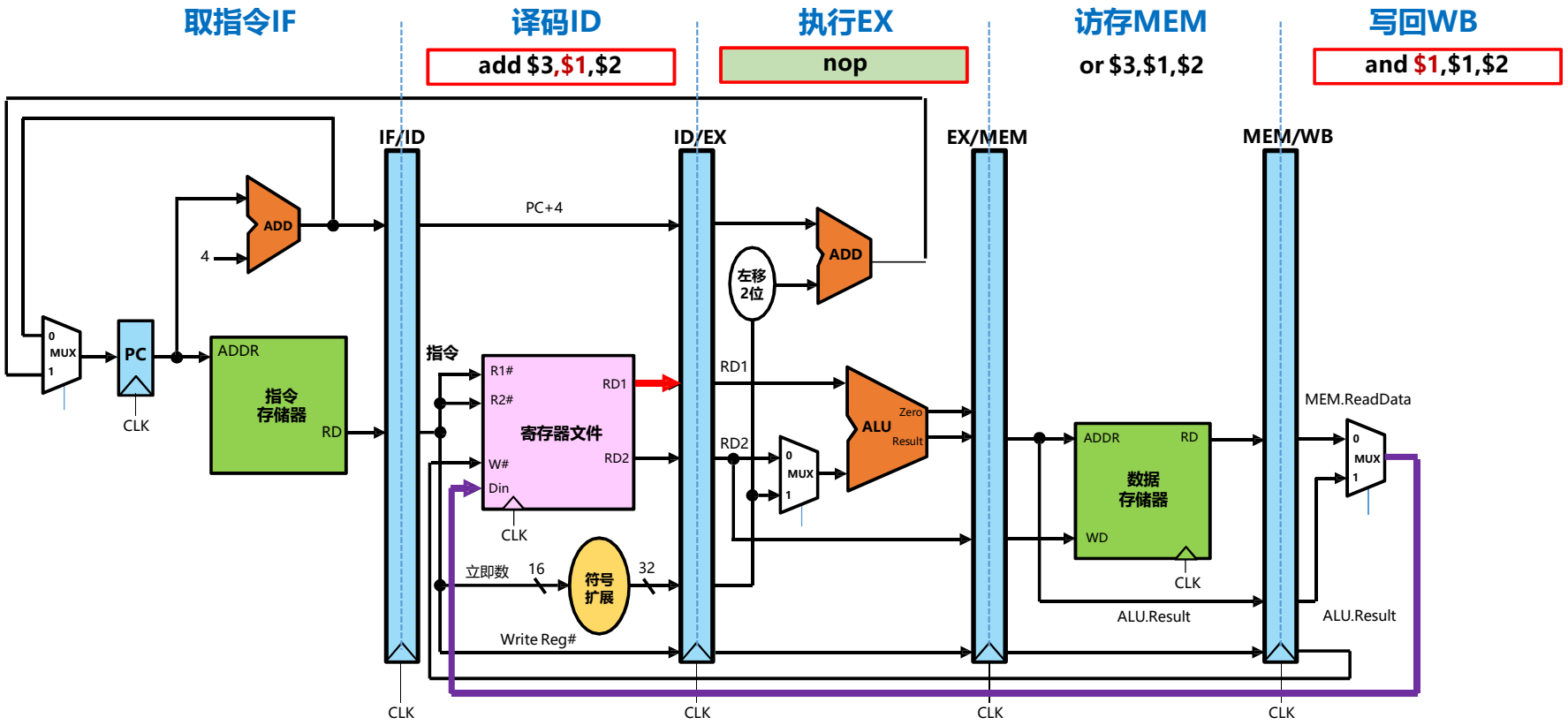
先写后读，寄存器文件下升沿写入，流水接口上降沿有效

ID段与MEM段数据相关



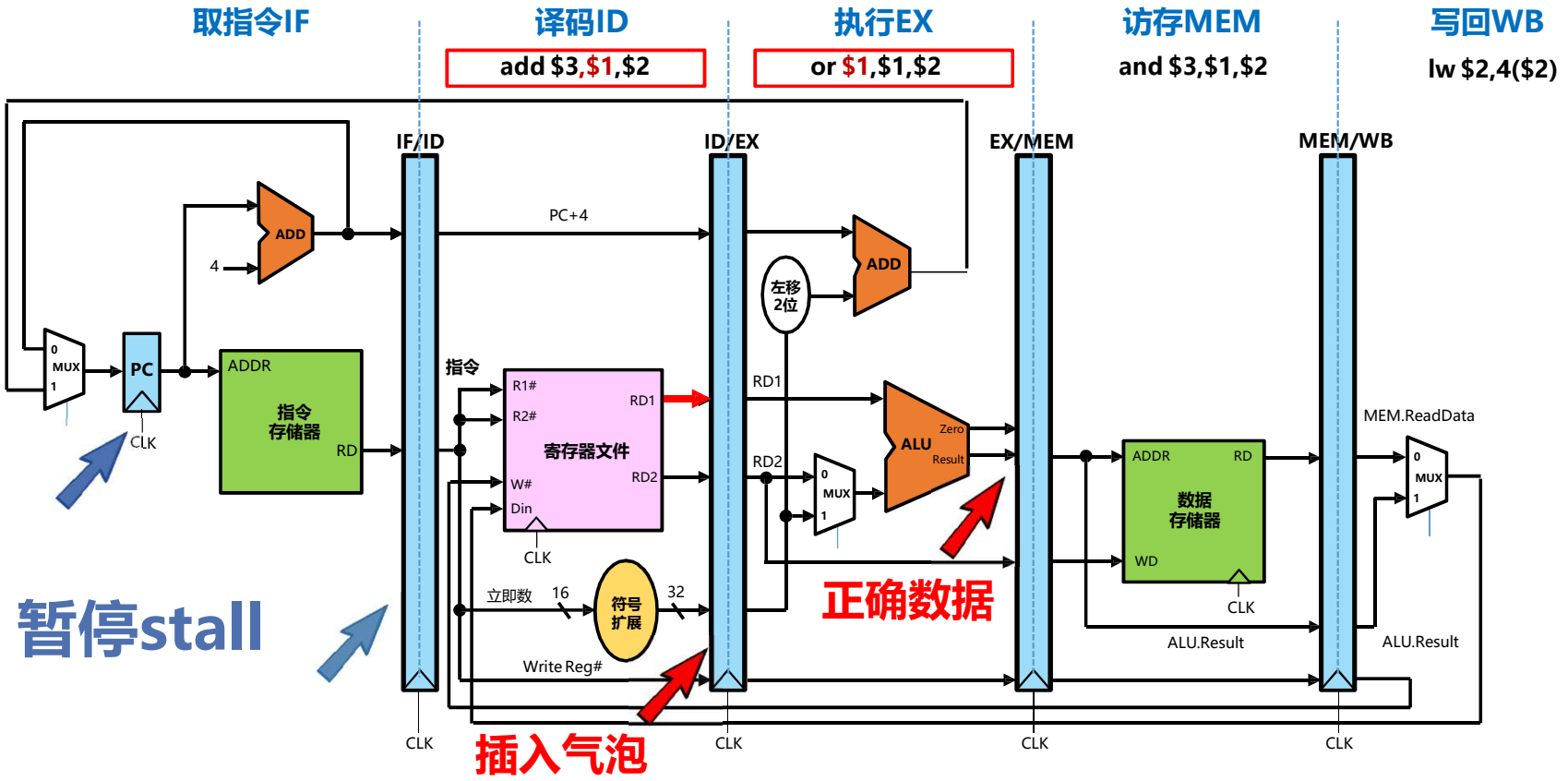
IF段，ID段暂停等待数据写回，EX段插入气泡

ID段与MEM段数据相关



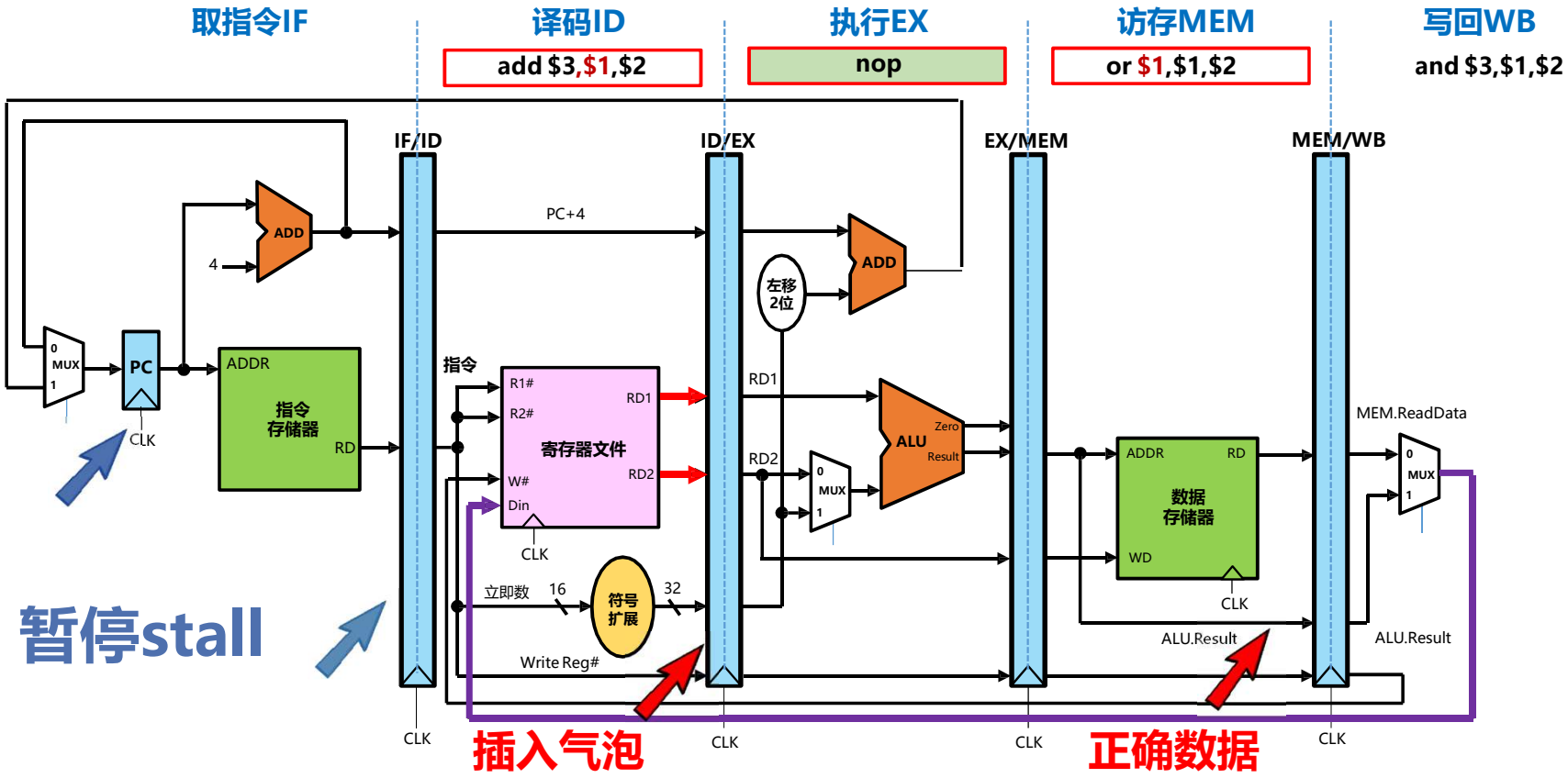
下一时刻数据相关变成与WB段相关

ID段与EX段数据相关



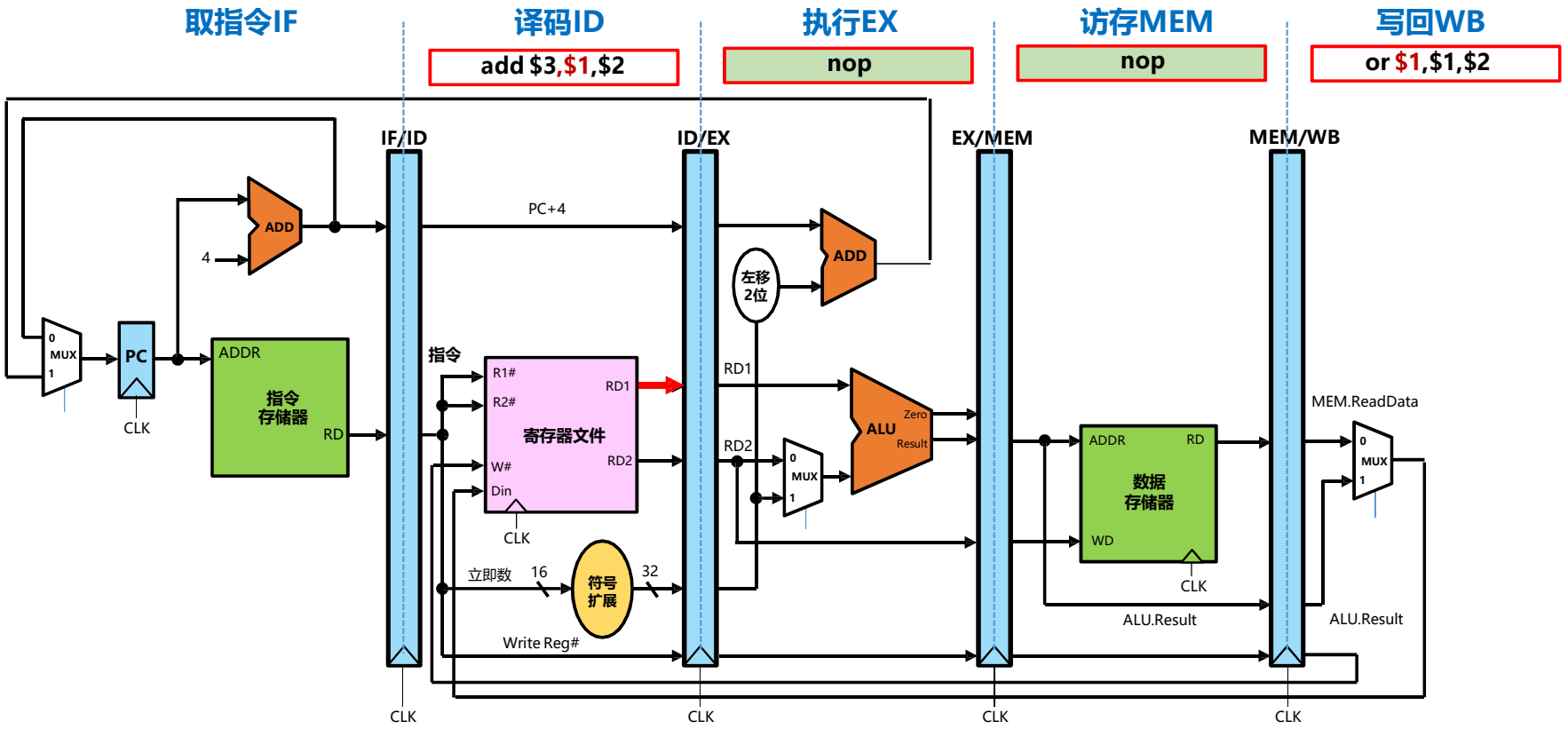
IF段，ID段暂停等待数据写回，EX段插入气泡

ID段与EX段数据相关



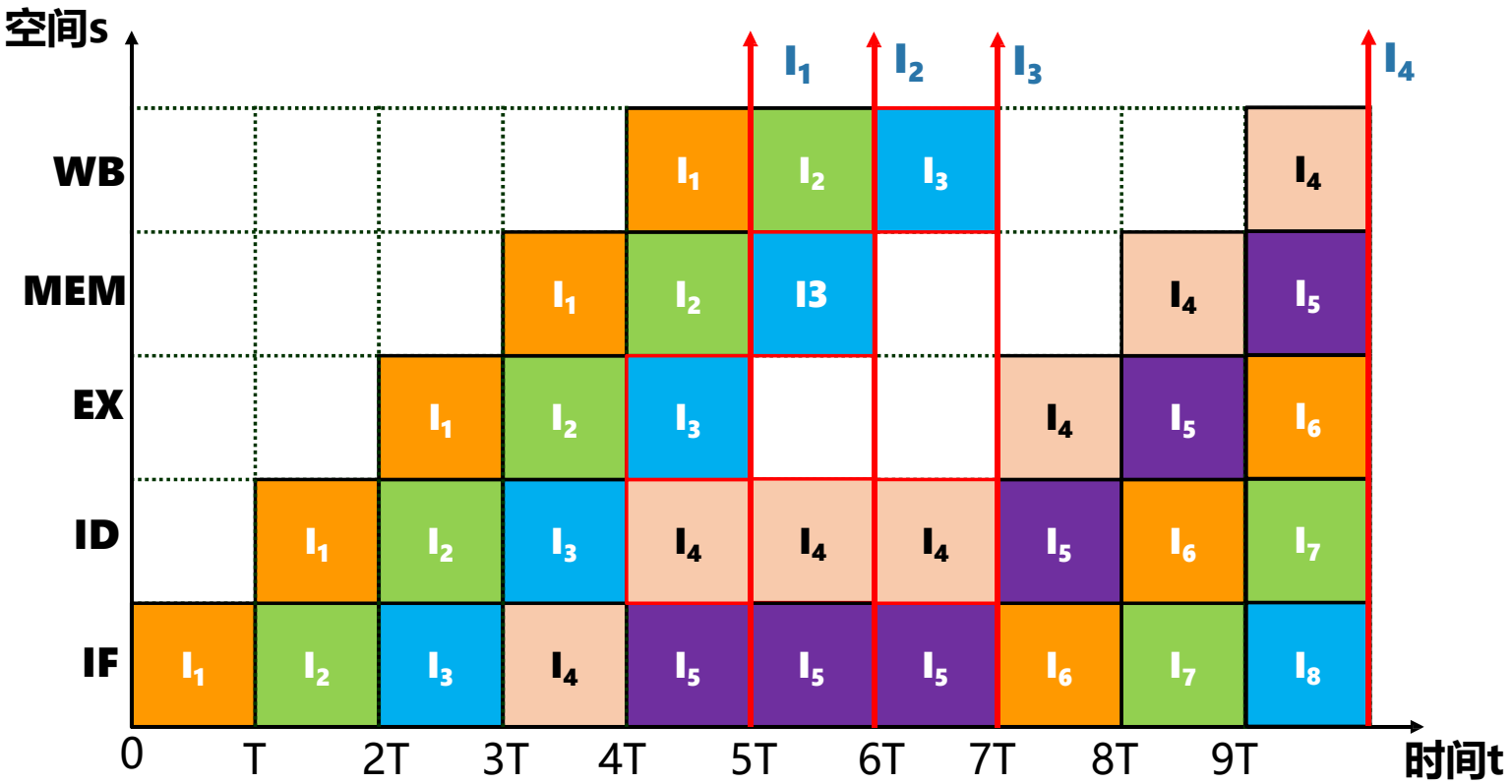
下一时刻相关变成与MEM段相关

ID段与EX段数据相关



再下一时刻相关变成与WB段相关

数据相关流水线时空图



流水线暂停两个时钟周期

数据相关处理总结

- **ID段与WB段相关**
  - 寄存器文件先写后读（下跳沿写入）
- **ID段与EX、MEM段数据相关**
  - 在ID段增加数据相关检测逻辑
  - IF段，ID段暂停，应该给出PC和ID/EX的阻塞信号stall（低电平有效，流水线停顿）
    - ◆控制PC写使能，IF/ID流水接口写使能
    - ◆需要增加IF/ID流水接口的写使能接口
  - ID向EX段插入一个气泡 （给出ID/EX接口同步清零信号）
  - 下一时刻如果相关解除，暂停信号，气泡信号也自然解除

数据相关检测逻辑

- **相关检测逻辑**
  - ID段当前指令读寄存器与后续2条指令写寄存器相同
    - (ID.ReadReg# == EX.WriteReg#) && (EX.RegWrite==1)
    - (ID.ReadReg# == MEM.WriteReg#) && (MEM.RegWrite==1)
    - 0号寄存器不考虑相关性
- **ID段包括0~2个源寄存器：R_1，R_2 是否使用标识**
  - R型指令
    - 运算指令 2个源操作数 R_1#=rs, R_2#=rt
    - syscall指令 2个源操作数 R_1#=v_0, R_2#=a_0
    - Jr 指令 1个源操作数 R_1#=rs
  - I型指令 1~2两个源操作数 R_1#=rs, R_2#=rt
- **不同类型指令包含写入0~1个写入寄存器**

数据相关处理实验

- **实现数据相关检测逻辑**
  - 构建源寄存器使用情况子电路
    - ◆输入：OP，Funct，输出：R_1_Used， R_2_Used，控制器真值表生成
  - 构建数据相关检测逻辑子电路
    - ◆输入：R_1 _Used, R_2 _Used，R_1#，R_2#，EX.WriteReg#, MEM.WriteReg#，RegWrite
    - ◆输出：数据相关信号
- **实现IF，ID段暂停逻辑**
  - 利用数据相关信号控制对应部件写使能，低电平有效
- **实现EX段插入气泡逻辑**
  - 利用数据相关信号控制ID/EX接口的同步清零信号
- **测试联调**