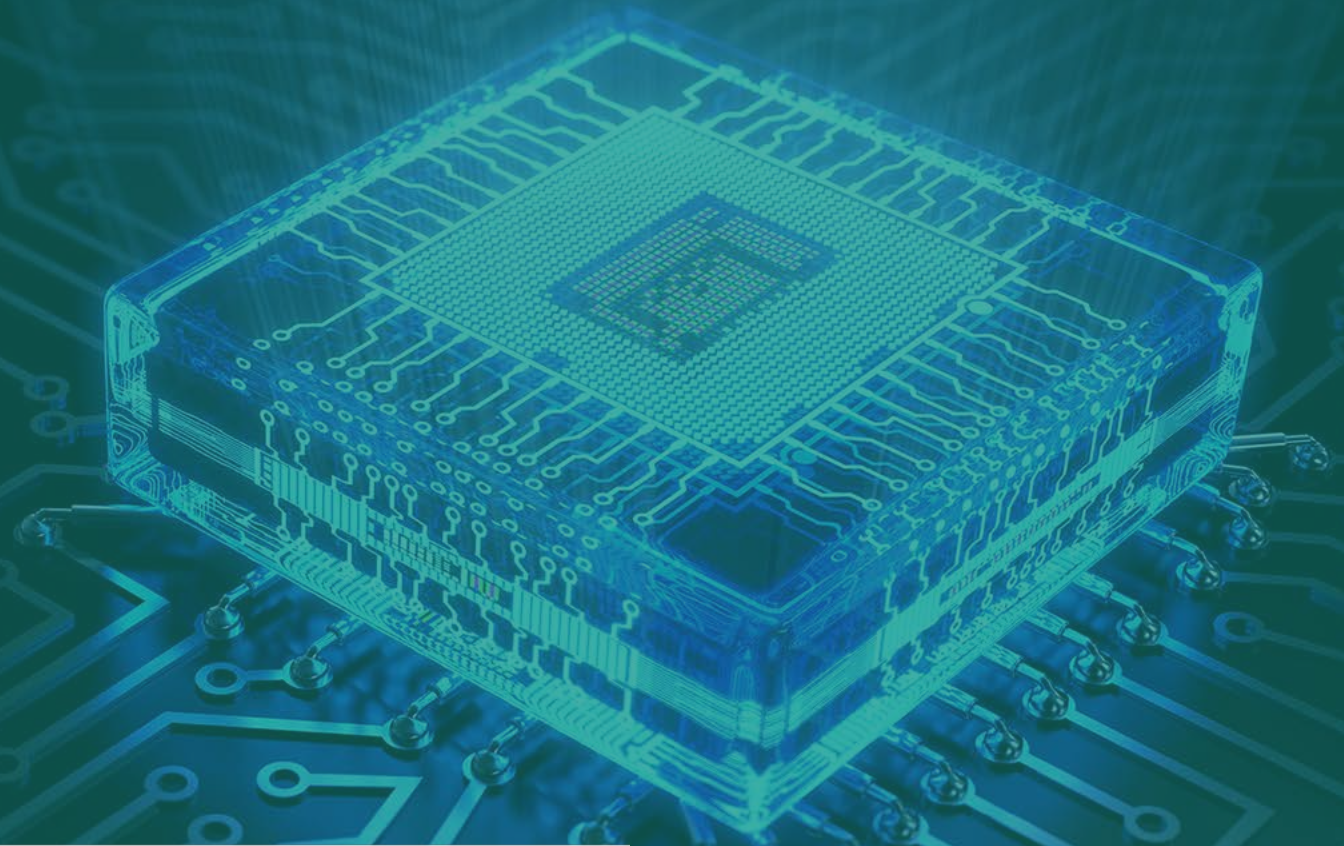
華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST



计算机组成原理

计算机组成原理

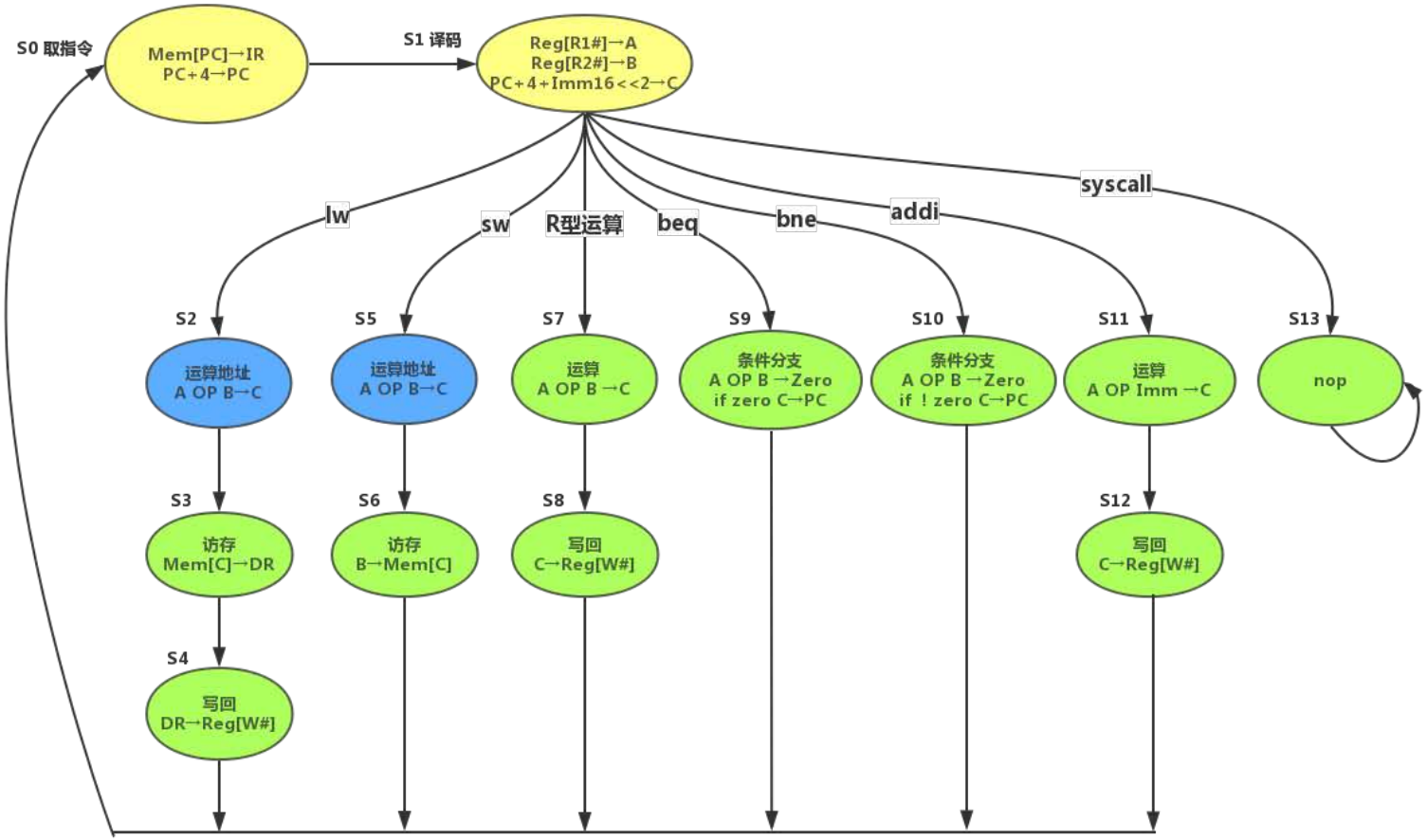
七、指令流水线

本章主要内容

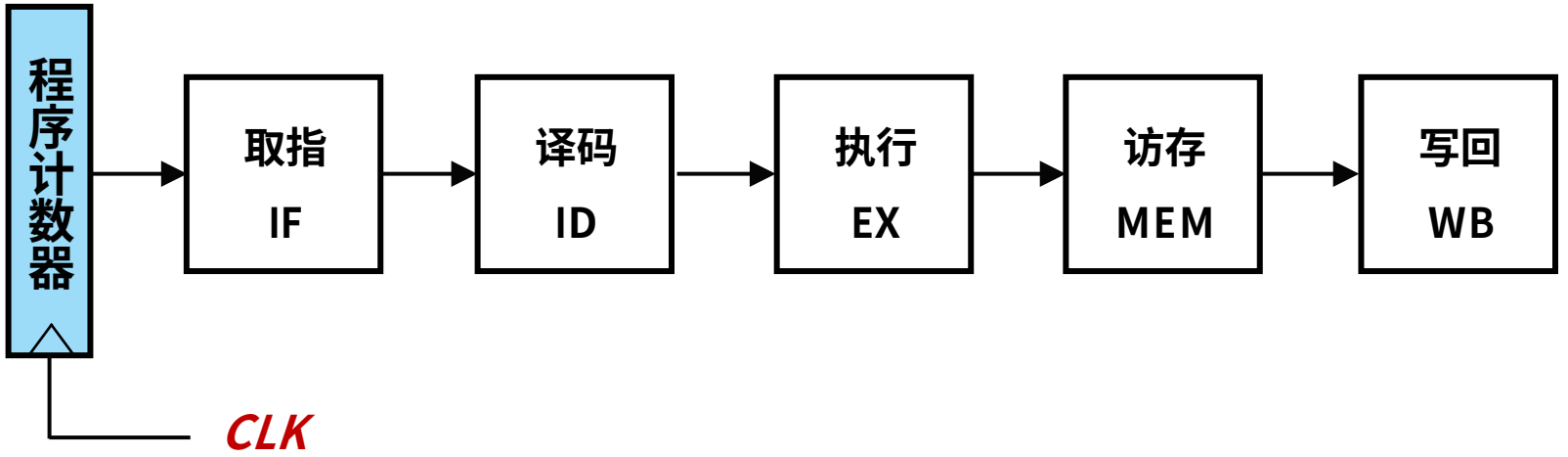
- 7.1 流水线概述
- 7.2 流水线数据通路
- 7.3 流水线冲突与处理
- 7.4 流水线的异常与中断
- 7.5 指令集并行技术

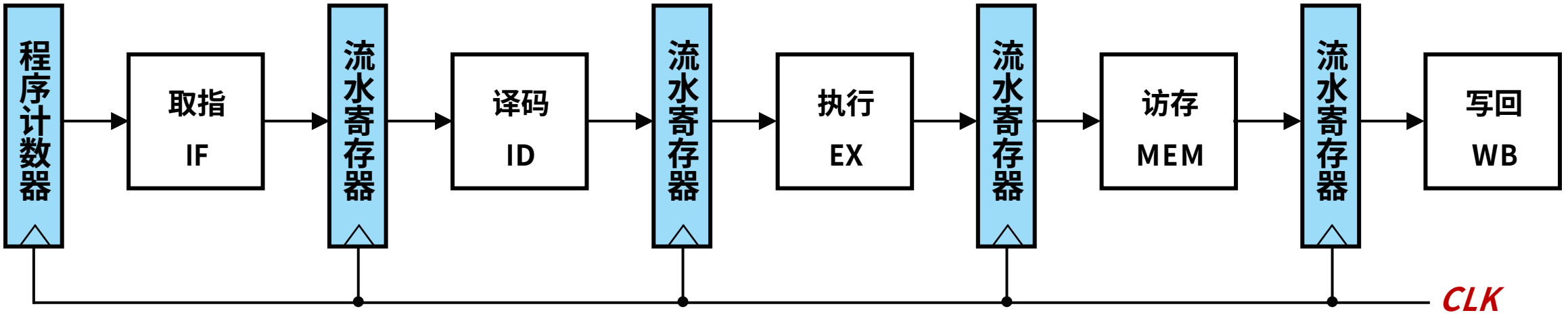
MIPS CPU实现方案

- 单周期方案
  - 性能受限于最慢的指令
  - 结构简单，实现容易
- 多周期方案
  - 传统多周期
    - 提升性能，复用器件
    - 异步控制，变长指令周期
  - 指令流水线
    - 多指令并行，提升性能
    - 部件并发



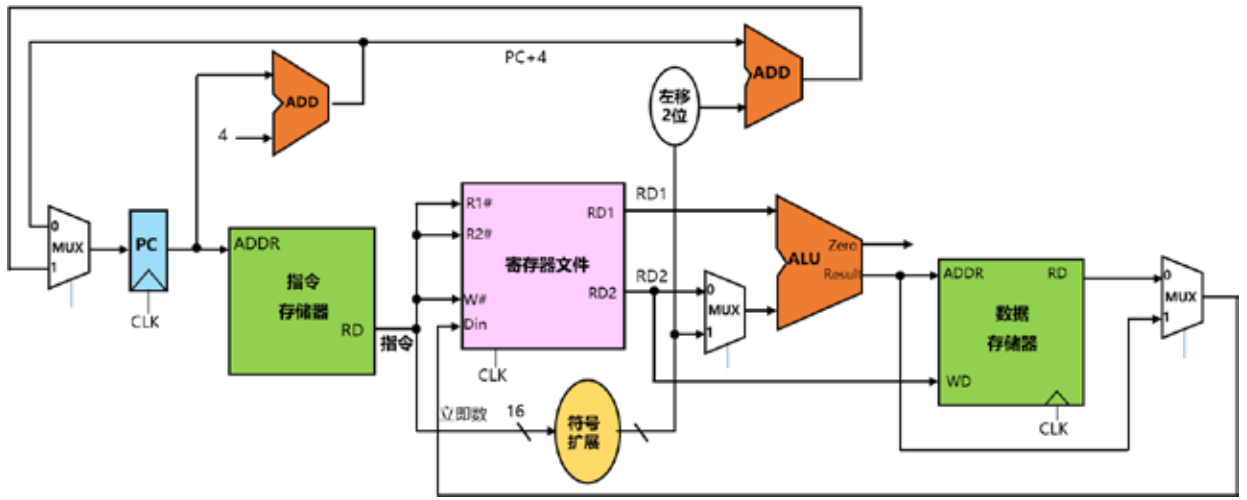
MIPS单周期与流水线

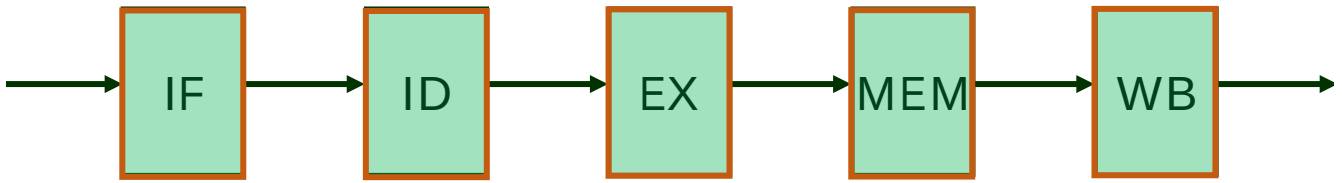




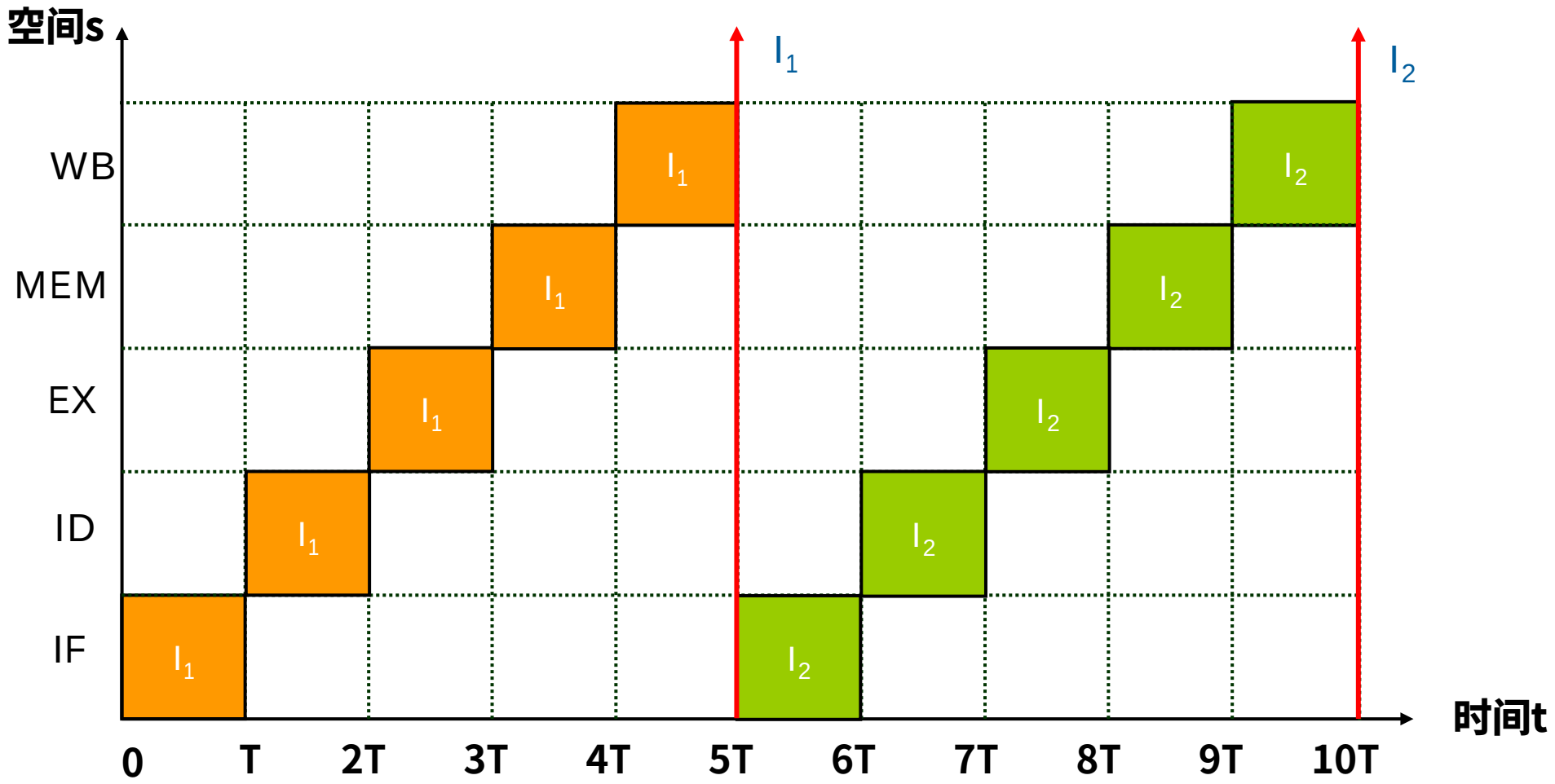
单周期指令运行动态

- 数据通路细分为5段，总时长为1个时钟周期=5T
  - 取指令　IF　(Instruction Fetch)
  - 指令译码　ID　(Instruction Decode)
  - 执行运算　EX　(Execution)
  - 访存阶段　MEM
  - 结果写回　WB　(Write Back)





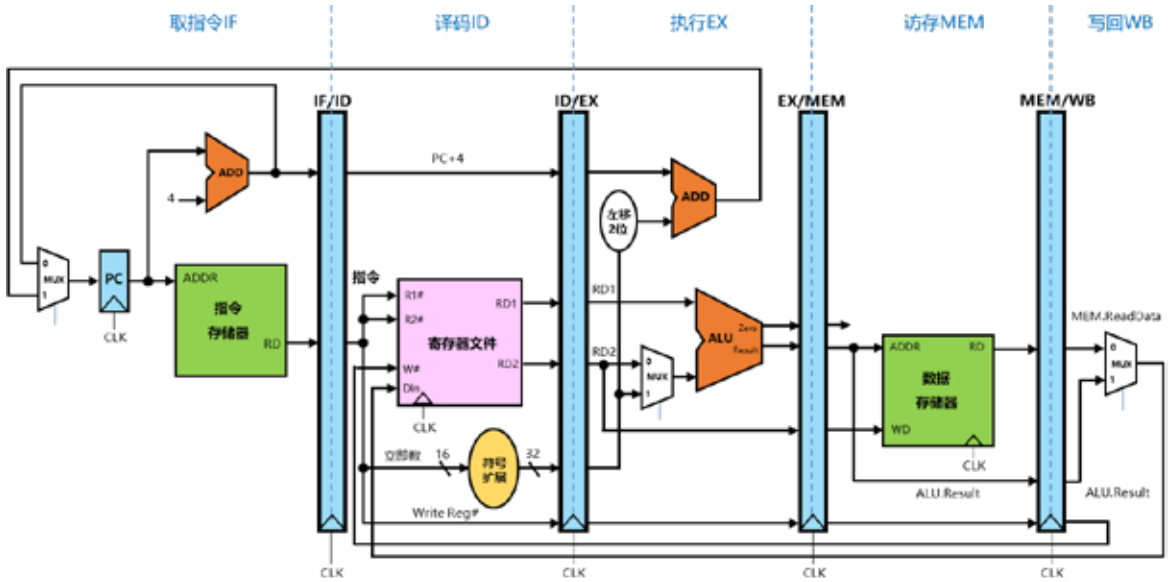
单周期时空图

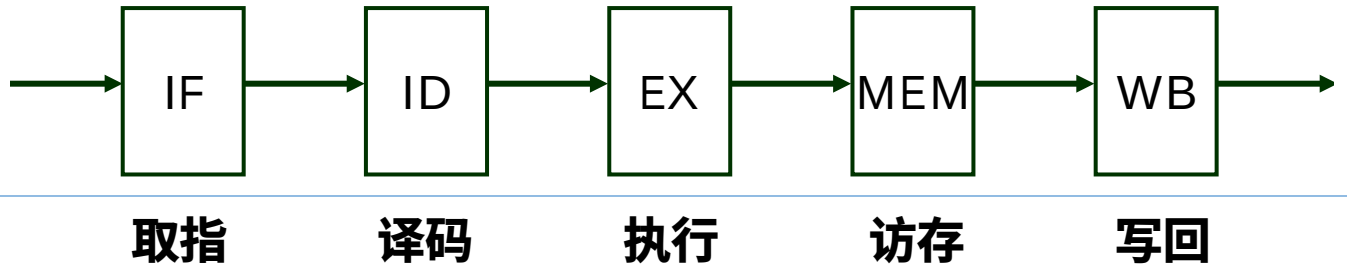


完成n条指令需要 5nT

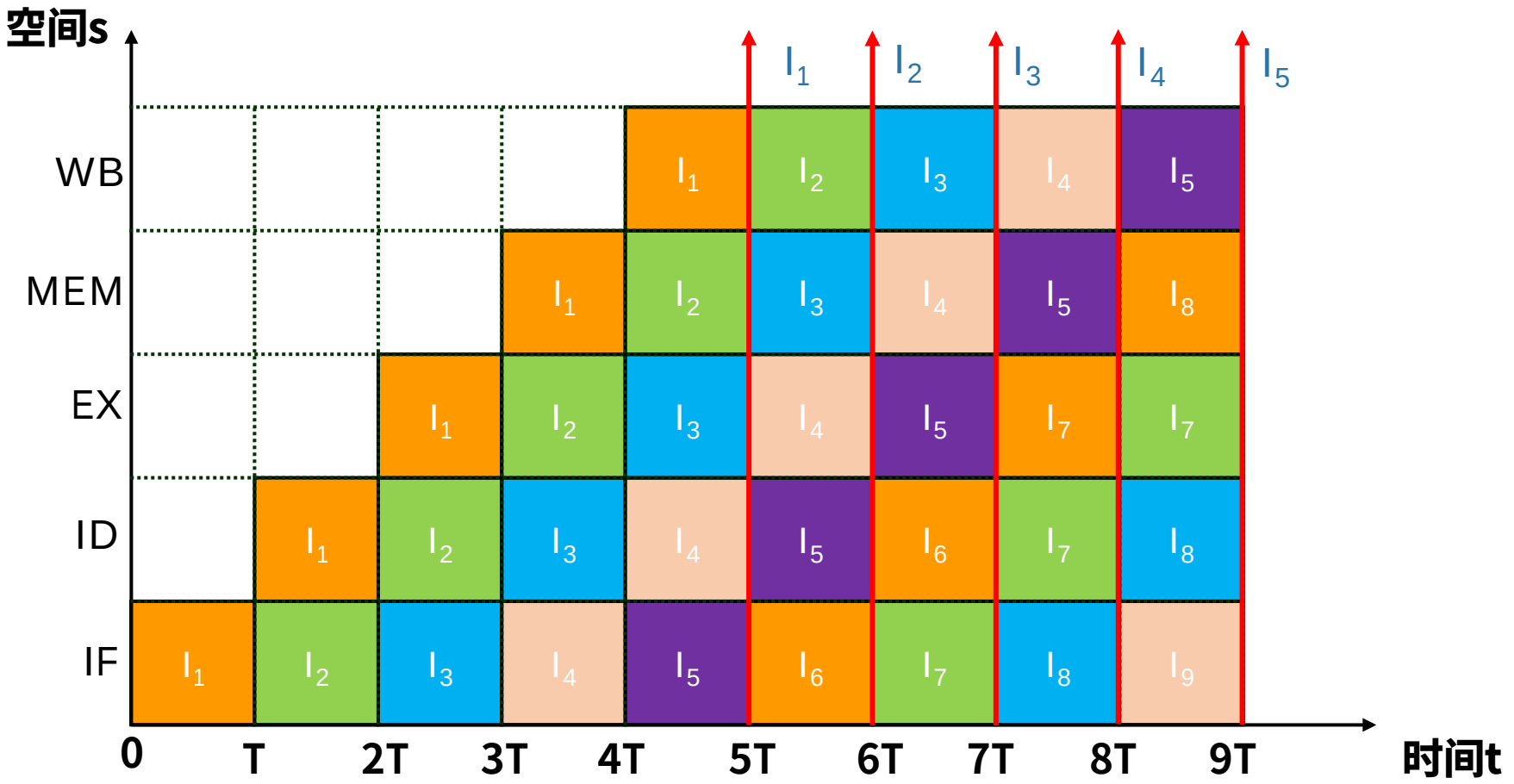
流水线指令运行动态

- 数据通路细分为5段，**各段完全并发**
  - 取指令　IF　(Instruction Fetch)
  - 指令译码　ID　(Instruction Decode)
  - 执行运算　EX　(Execution)
  - 访存阶段　MEM
  - 结果写回　WB　(Write Back)





指令流水线时空图



完成n条指令的时间=完成第一条指令时间5T+(n-1)*T=(n+4)T

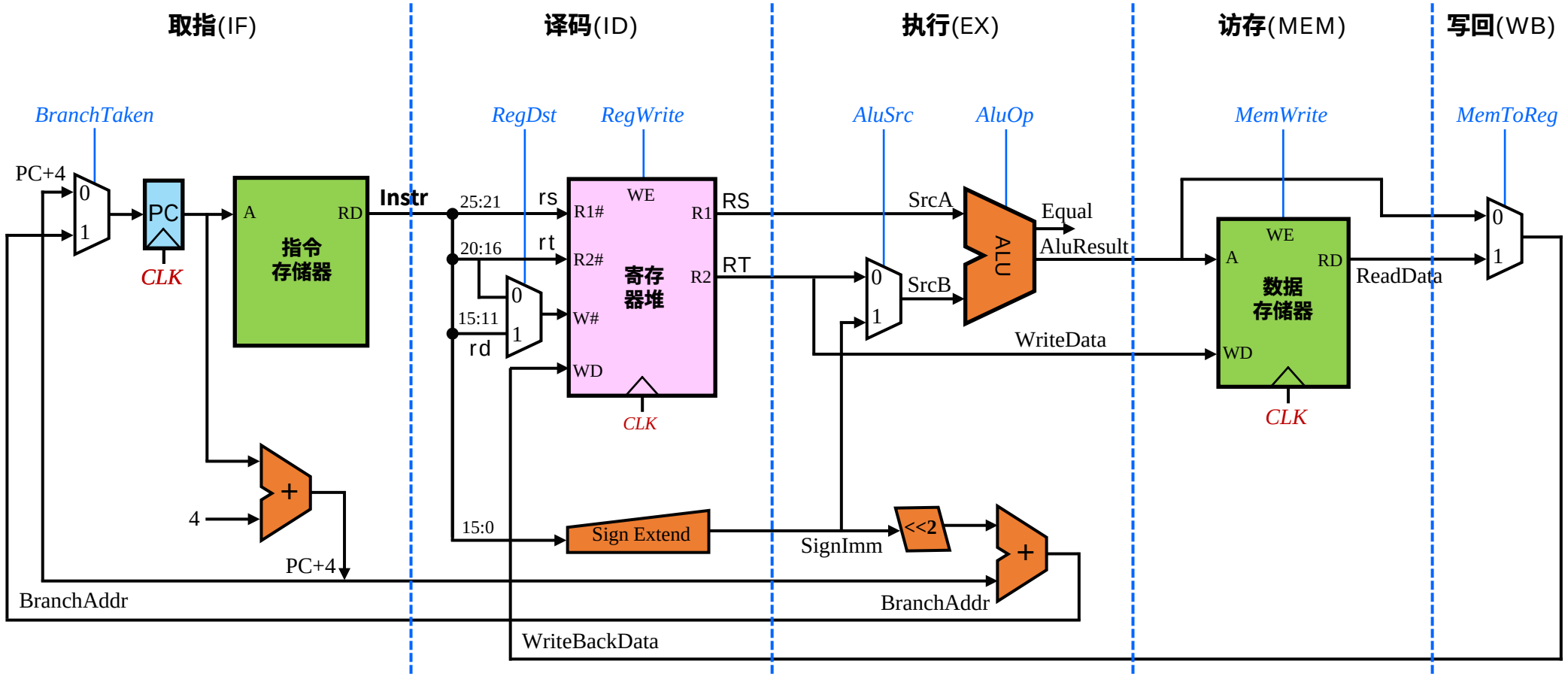
理想流水线

- 适用范围
  - 工业自动化流水线
  - 指令流水线？
- 理想流水线特征
  - 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
  - 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
  - 无资源冲突：不同阶段之间无共享资源，各段完全并发　　取指令、取数存在内存争用
  - 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
    - Microprocessor without interlocked piped stages（MIPS）

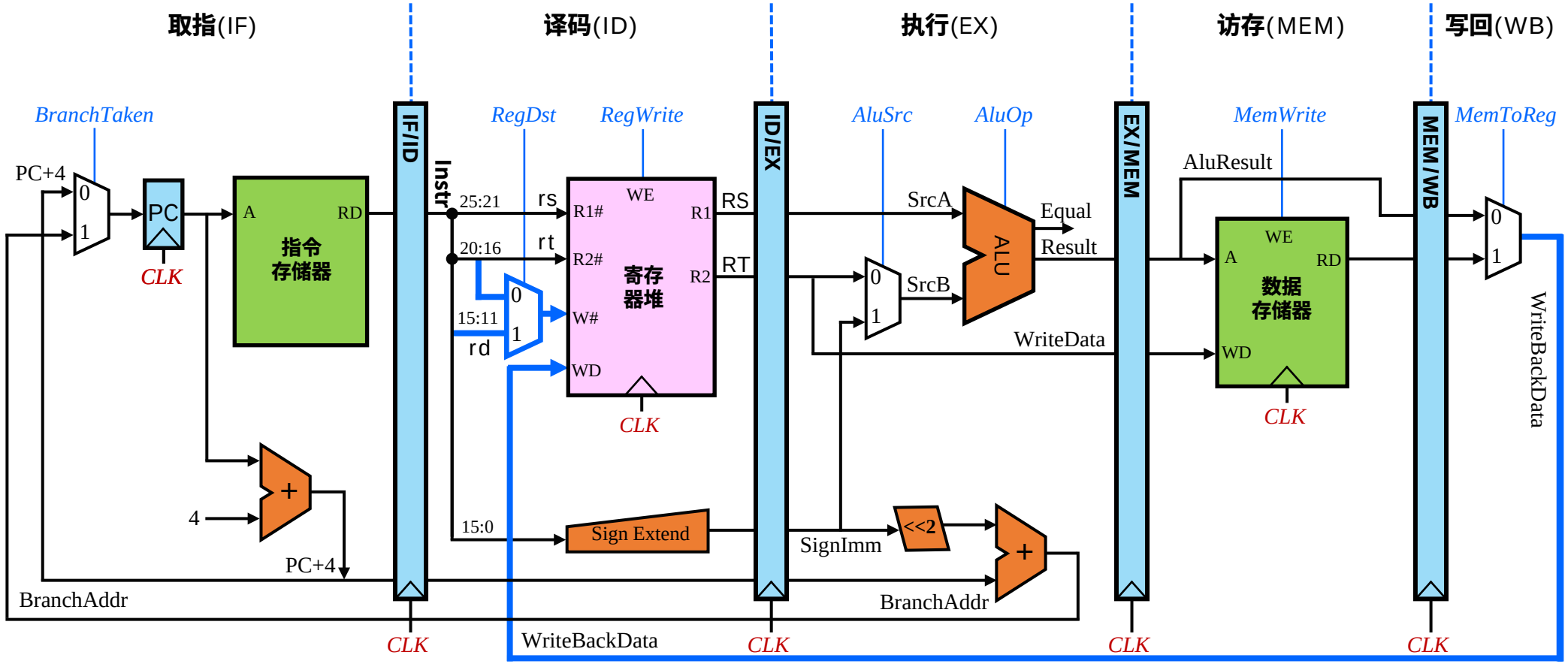
本章主要内容

- 7.1 流水线概述
- **7.2 流水线数据通路**
- 7.3 流水线冲突与处理
- 7.4 流水线的异常与中断
- 7.5 指令集并行技术

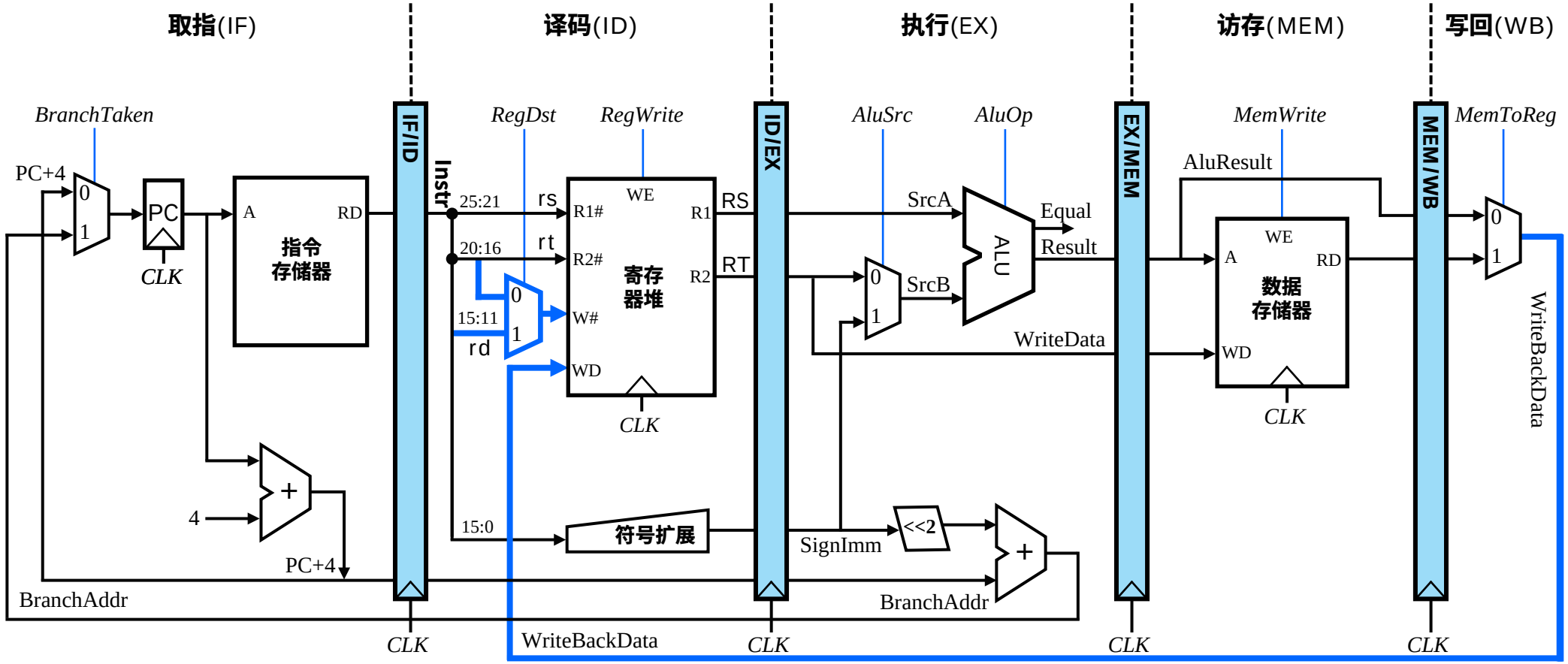
单周期MIPS处理器数据通路



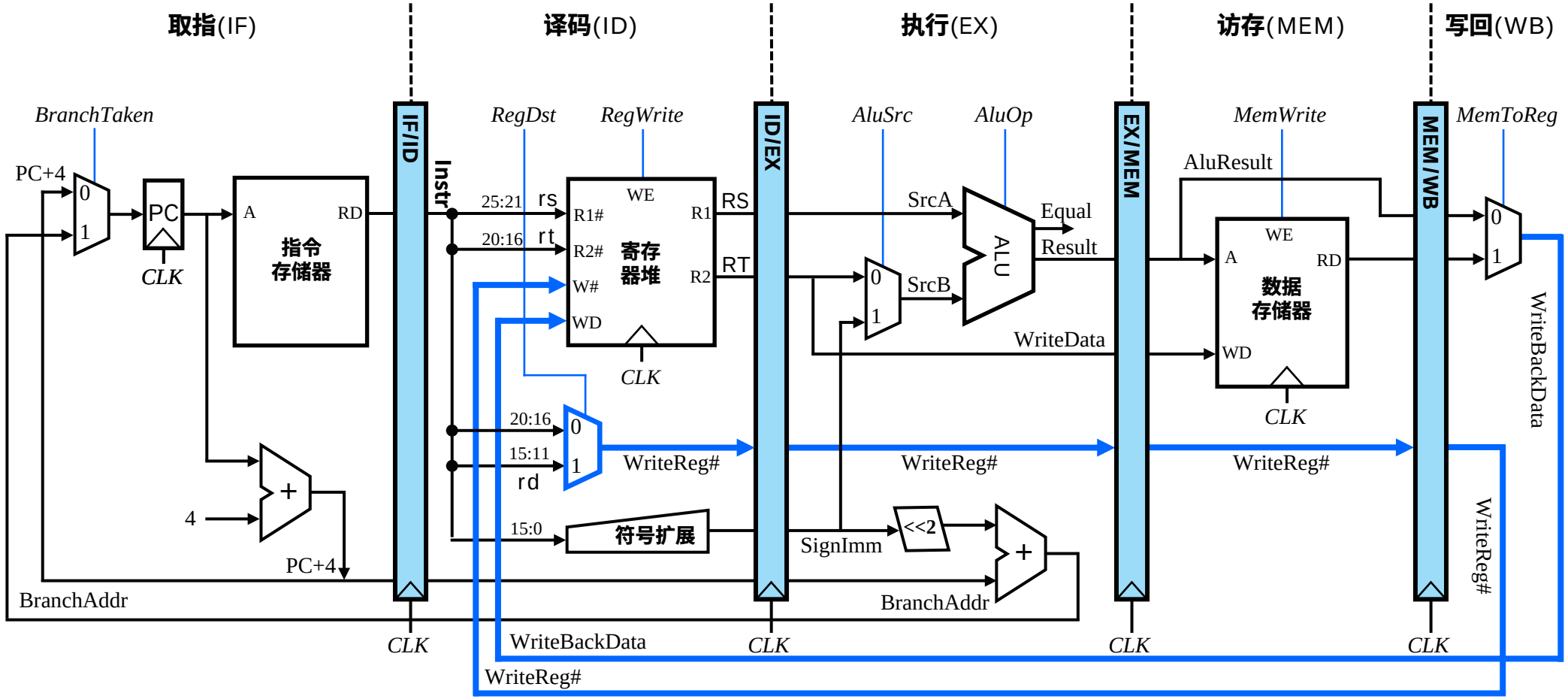
单周期MIPS数据通路流水线改造



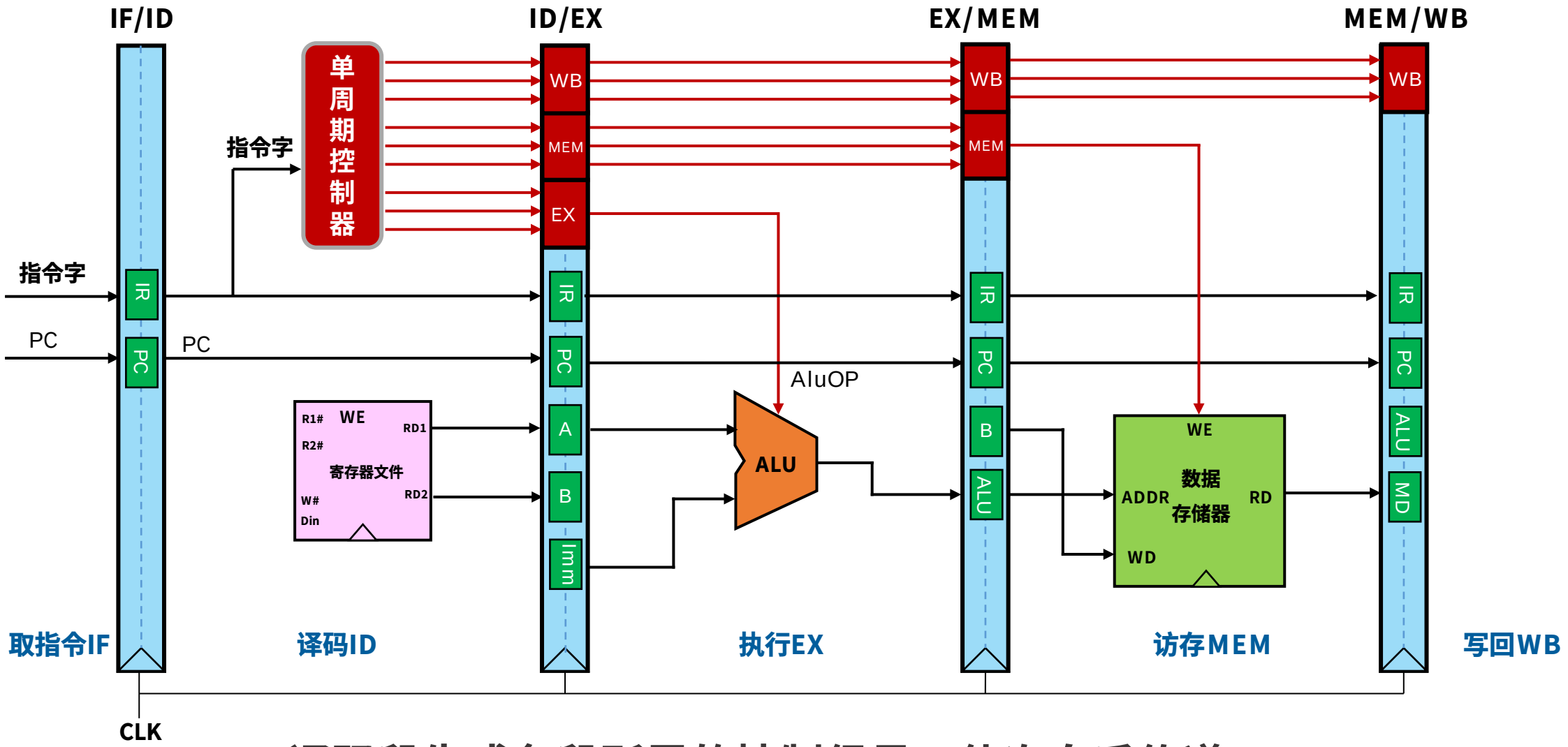
单周期MIPS数据通路流水线改造



流水线中写回数据通路改造



5段指令流水线数据与信号传递

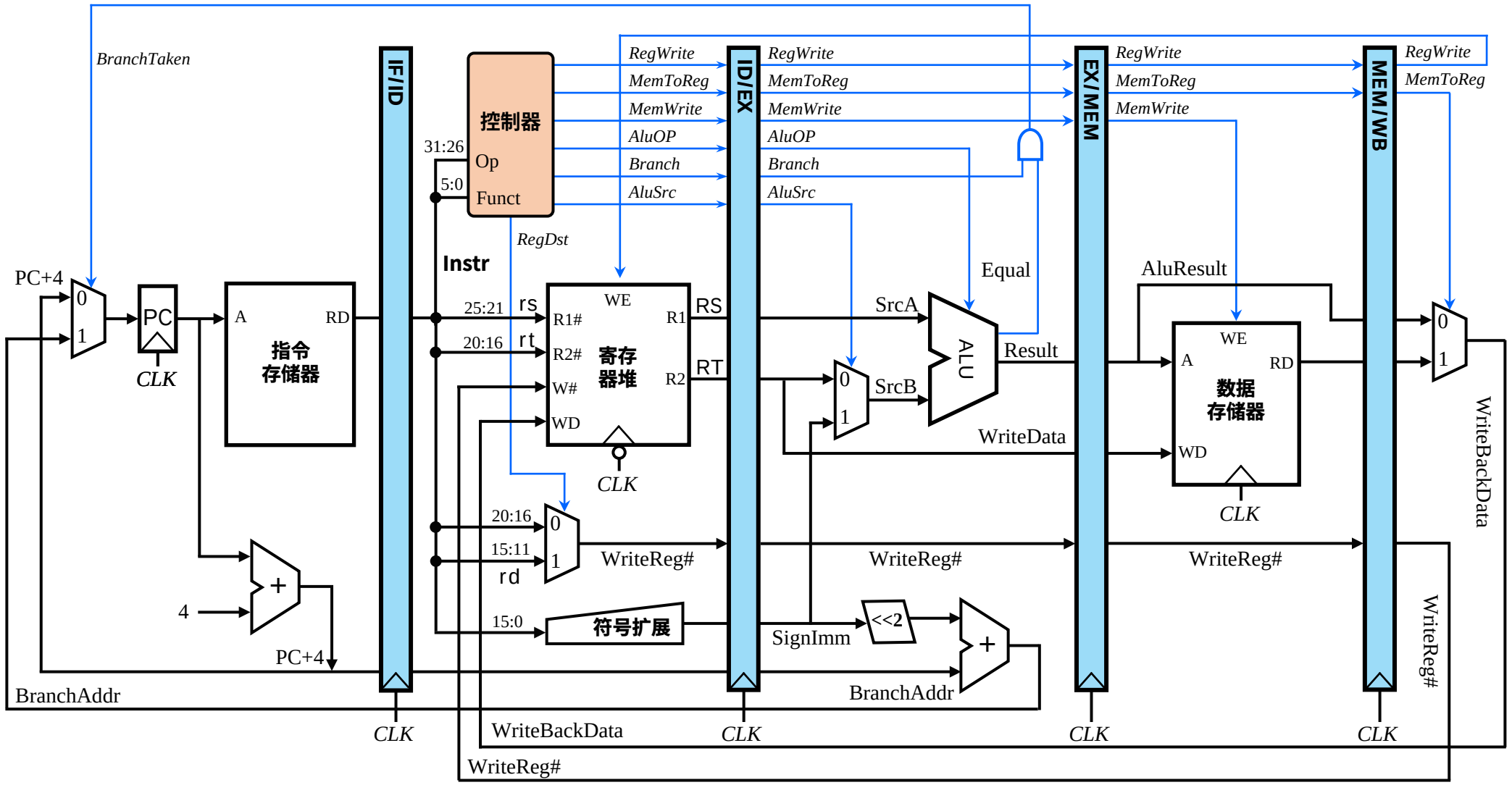


n 译码段生成各段所需的控制信号，依次向后传递

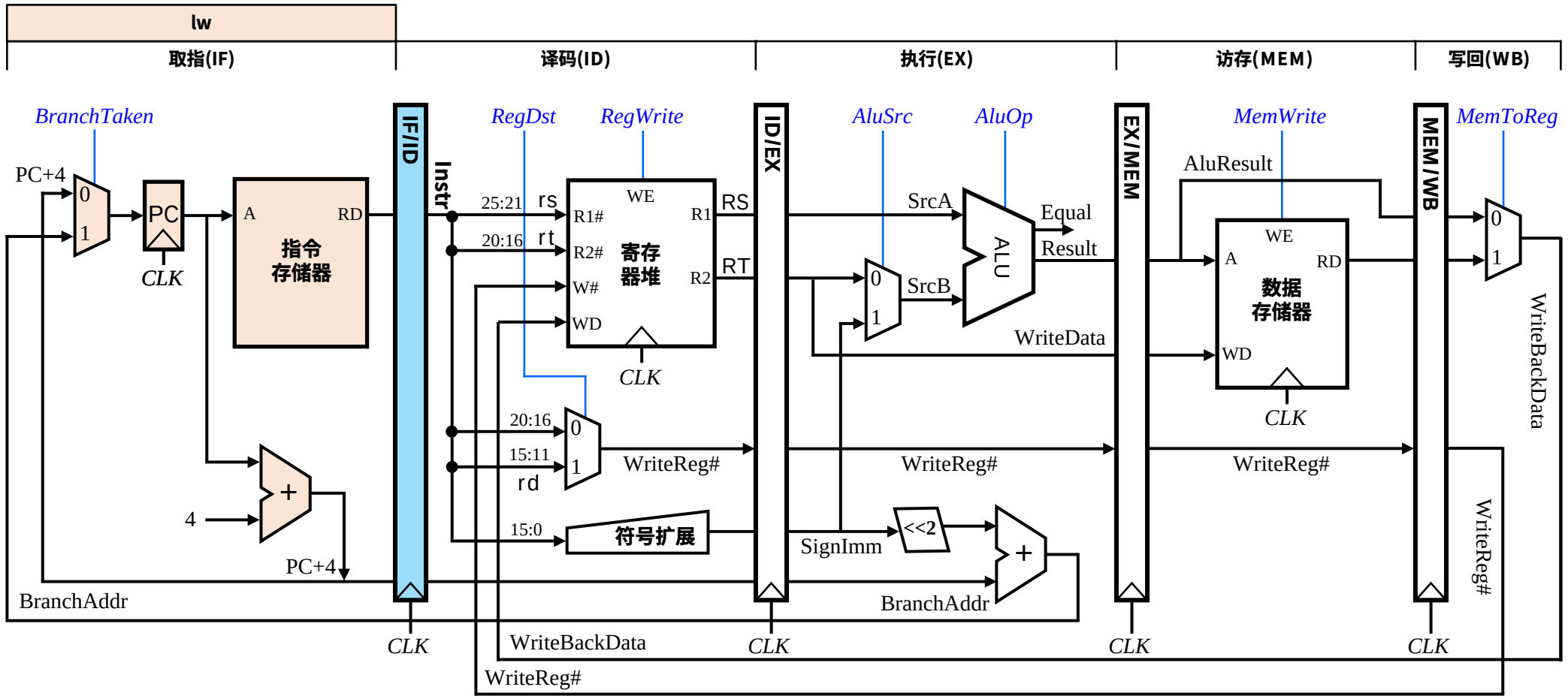
控制信号分类

控制信号	位置	来源	功能说明
BranchTaken	**IF**	**EX**	分支跳转信号，为 1 表示跳转，由 EX 段的 Branch 信号与 equal 标志进行逻辑与生成
RegDst	ID	ID	写入目的寄存器选择，为 1 时目的寄存器为 rd 寄存器，为 0 时为 rt 寄存器
RegWrite	**ID**	**WB**	控制寄存器堆写操作，为 1 时数据需要写回寄存器堆中的指定寄存器
AluSrc	EX	EX	ALU 的第二输入选择控制，为 0 时输入寄存器 rt，为 1 时输入扩展后的立即数
AluOp	EX	EX	控制 ALU 进行不同运算，具体取值和位宽与 ALU 的设计有关
MemWrite	MEM	MEM	控制数据存储器写操作，为 0 时进行读操作，为 1 时进行写操作
MemToReg	WB	WB	为 1 时将数据存储器读出数据写回寄存器堆，否则将 ALU 运算结果写回

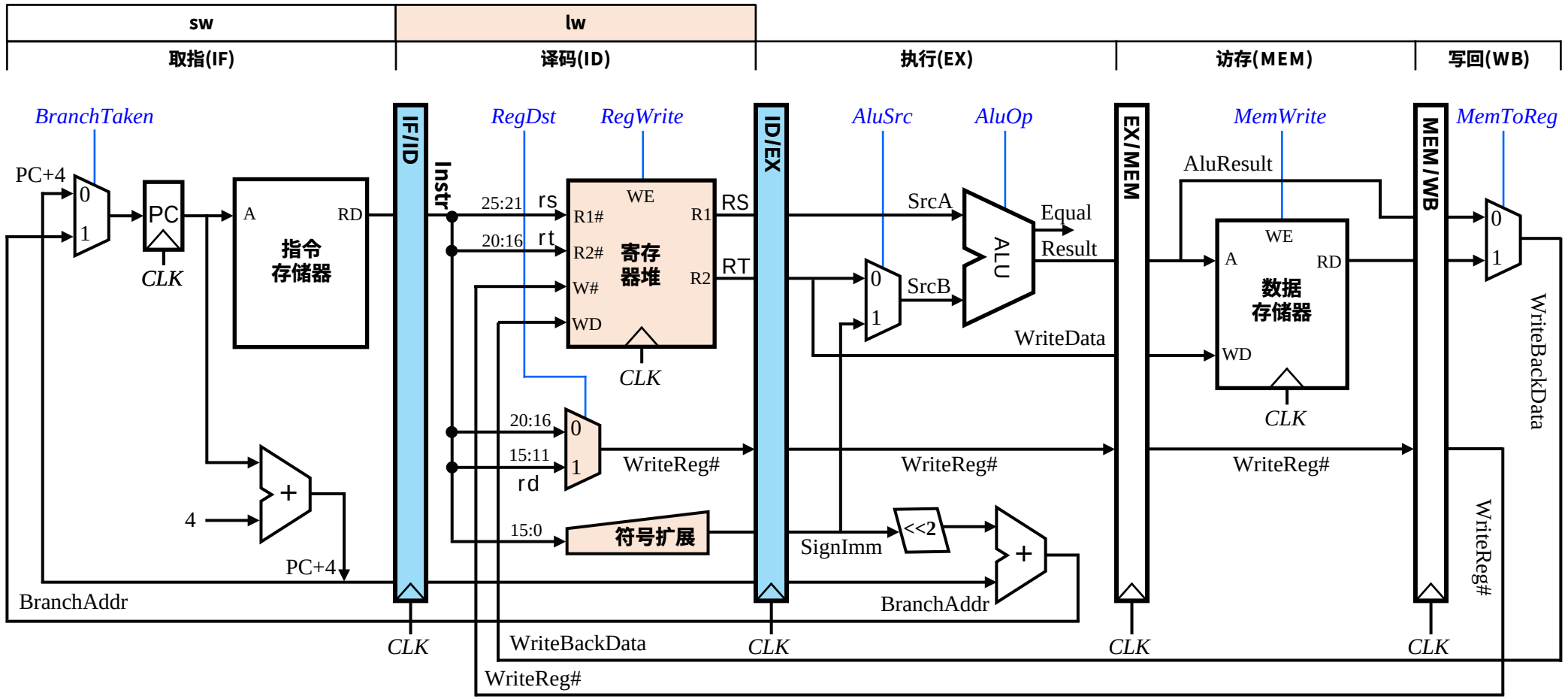
5段流水线控制信号与传递



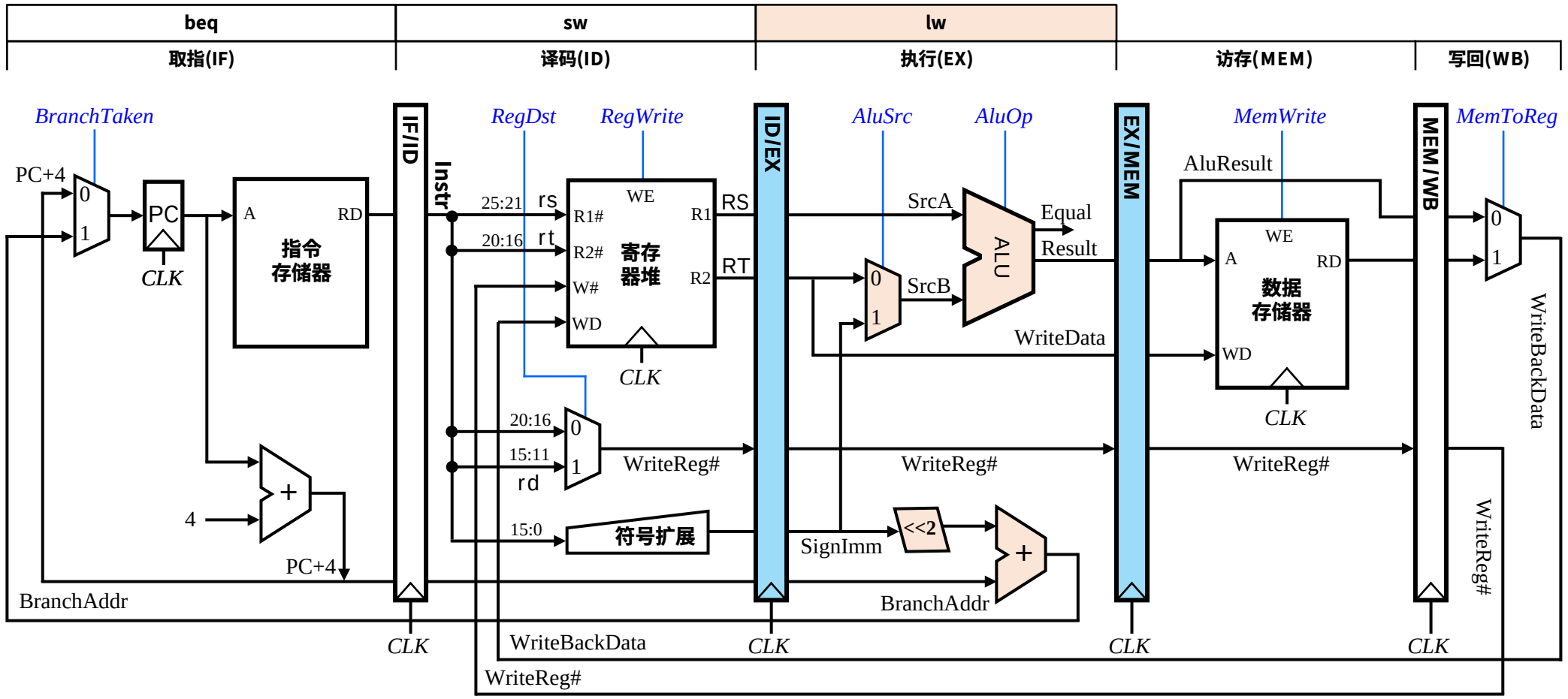
7.2.3 指令在流水线中的执行过程



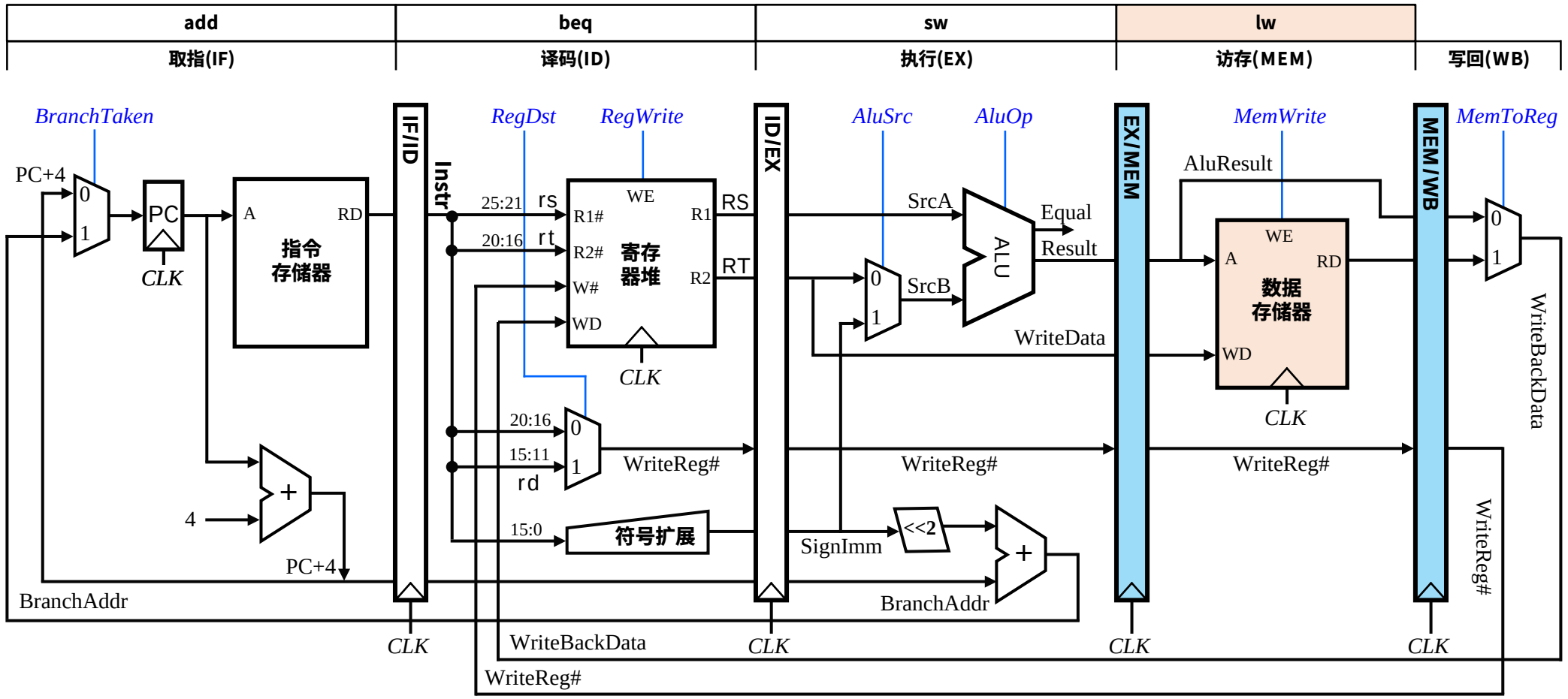
7.2.3 指令在流水线中的执行过程



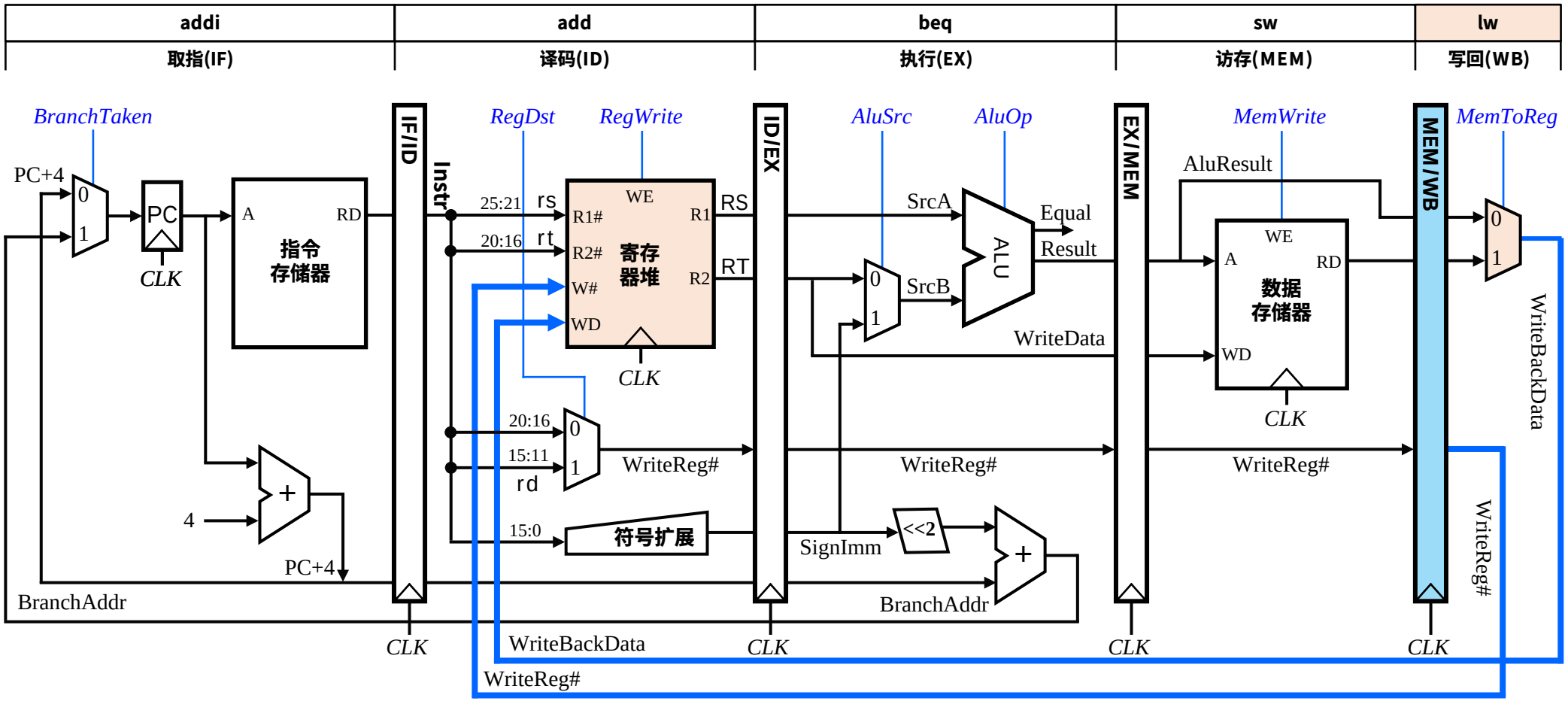
7.2.3 指令在流水线中的执行过程



7.2.3 指令在流水线中的执行过程



7.2.3 指令在流水线中的执行过程



本章主要内容

- 7.1 流水线概述
- 7.2 流水线数据通路
- **7.3 流水线冲突与处理**
- 7.4 流水线的异常与中断
- 7.5 指令集并行技术

指令流水线的冲突、相关、冒险（hazard）

- 结构冲突
  - 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
- 控制冲突
  - 提前取出的指令作废，流水线清空
  - 流水线发生中断
- 数据冲突
  - 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

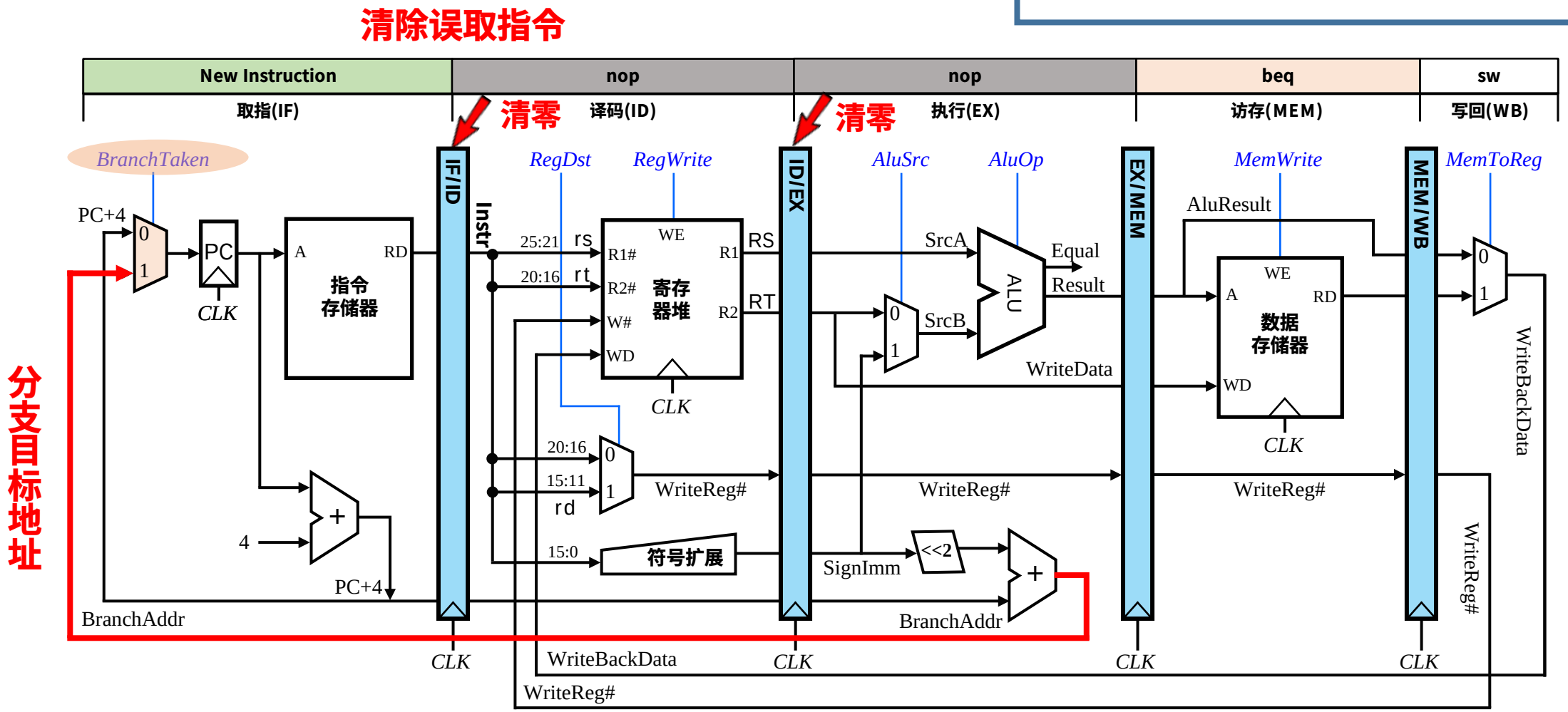
```
ADD $s1, $s2, $s3
ADD $s4, $s1, $s3
```

控制冲突总结

- 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

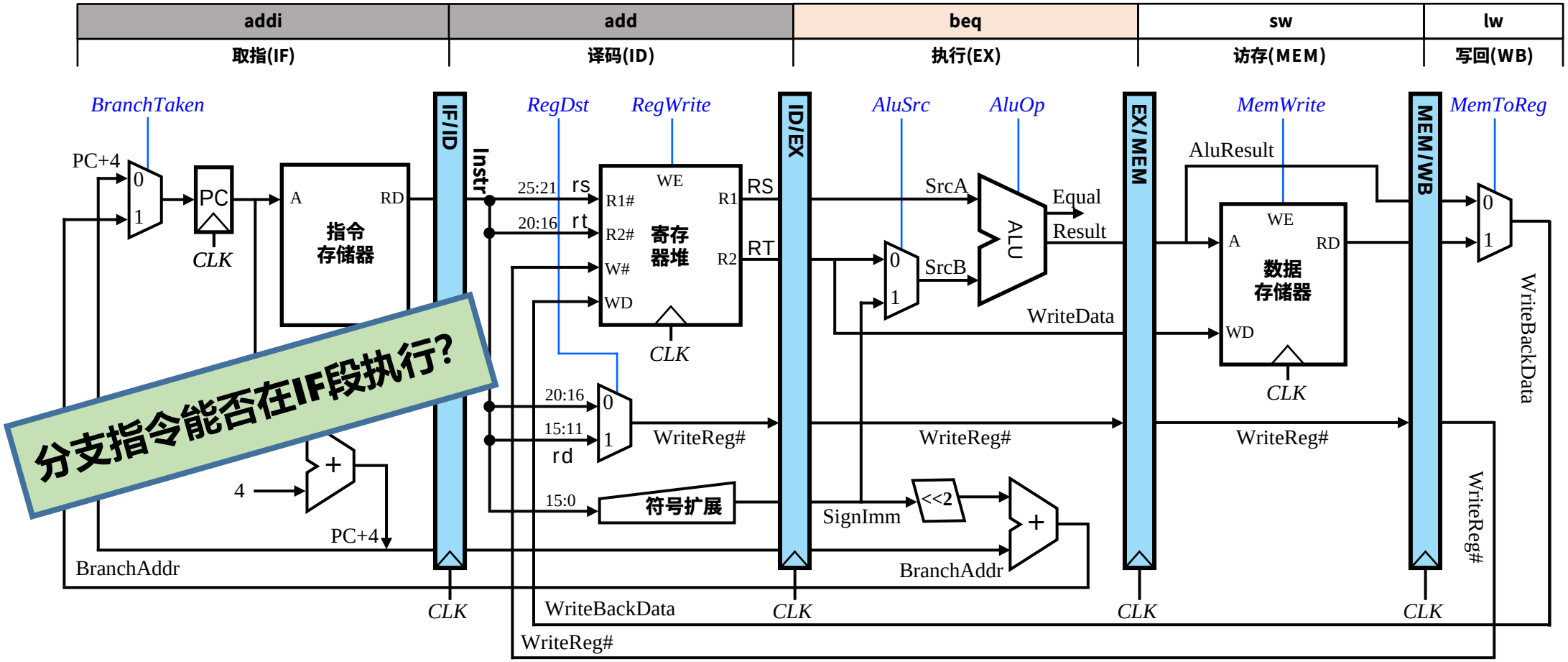
分支相关

流水接口清零 同步/异步？



分支逻辑：IF段根据EX段分支目标地址取指令，清除误取指令

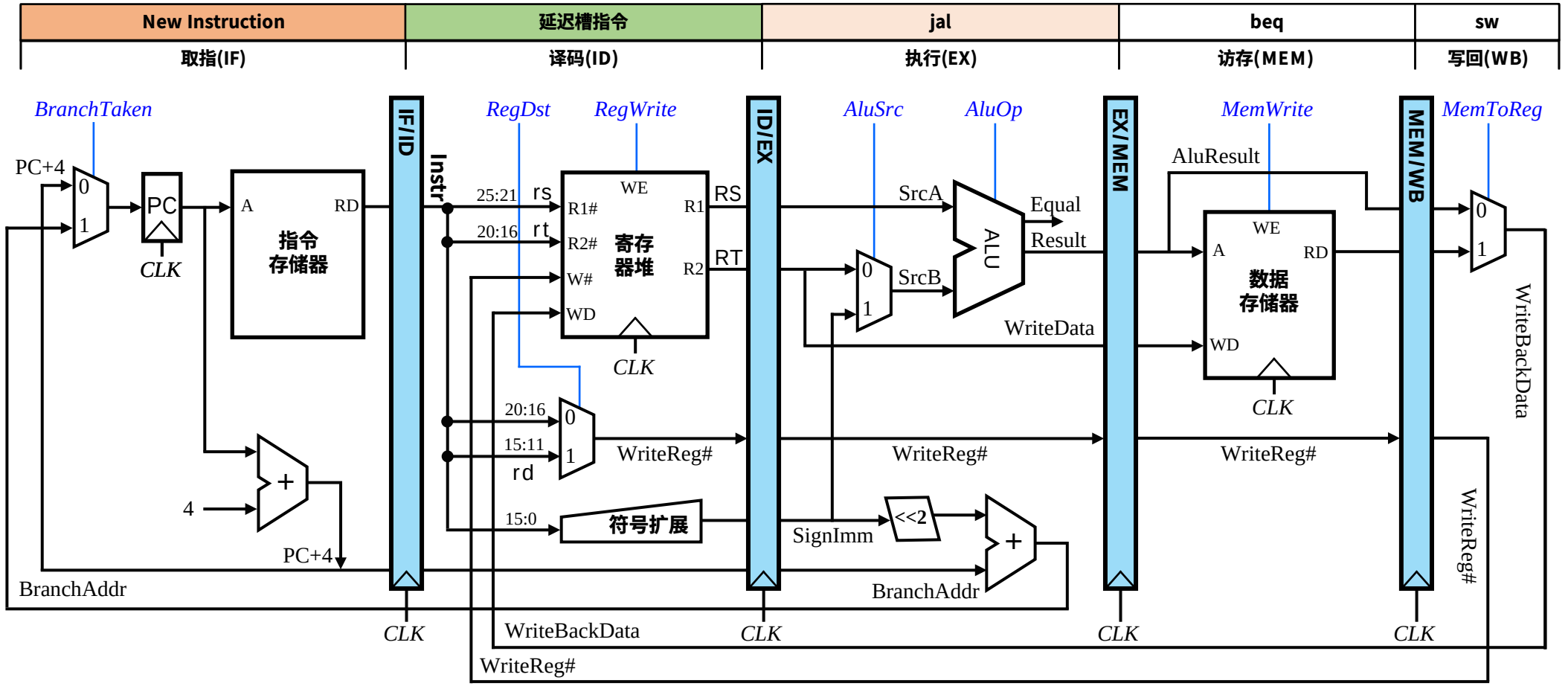
分支指令执行时机？



分支执行越早，性能损失越小！

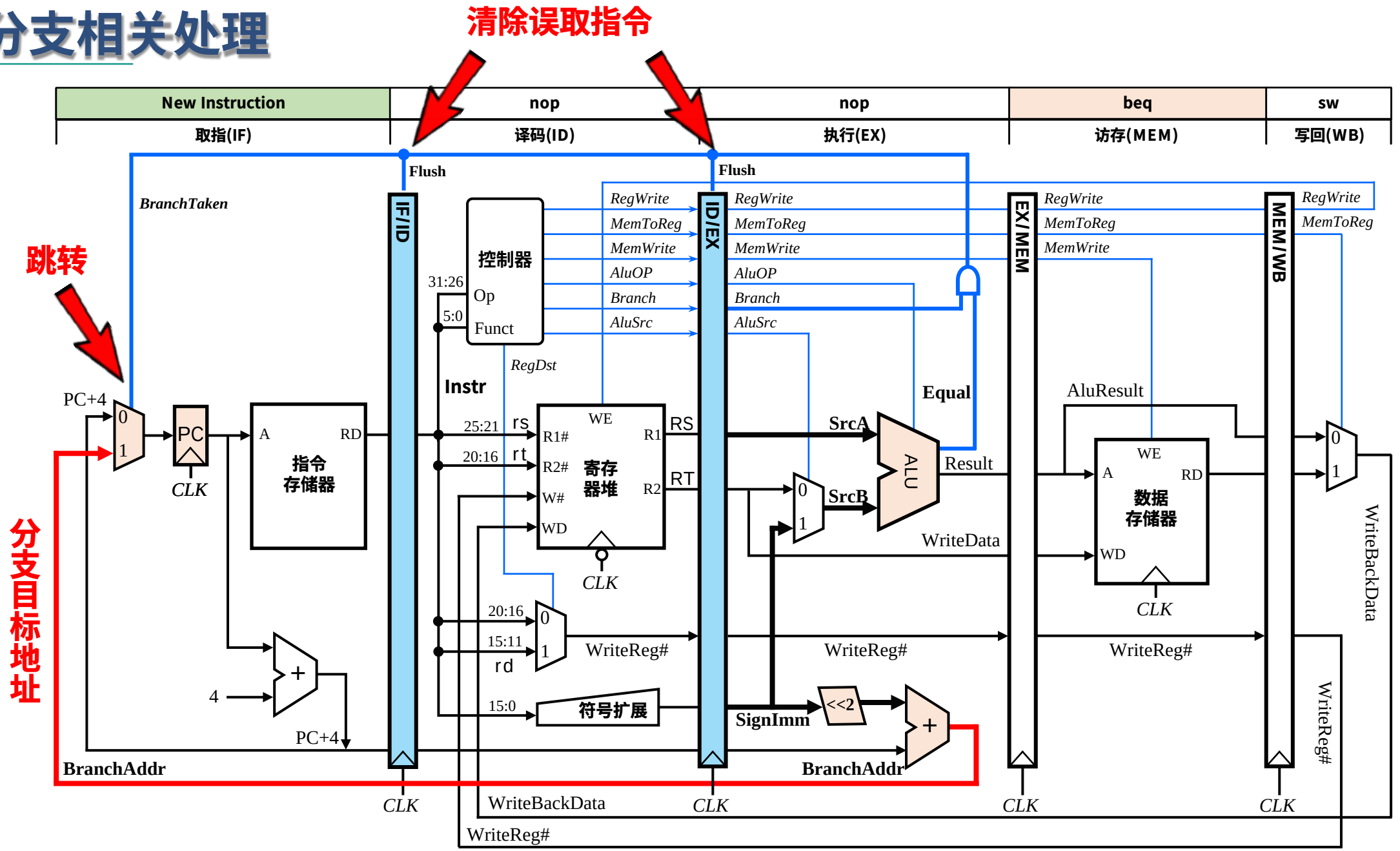
MIPS延迟槽技术

```
I:  GPR[31]← PC + 8
```



- 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

分支相关处理


清除误取指令
New Instruction
nop
nop
beq
sw
取指(IF)
译码(ID)
执行(EX)
访存(MEM)
写回(WB)
Flush
Flush
BranchTaken
跳转
分支目标地址
IF/ID
ID/EX
EX/MEM
MEM/WB
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RegDst
控制器
31:26
Op
5:0
Funct
Instr
PC+4
PC
CLK
指令
存储器
A
RD
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
寄存
器堆
R1
R2
RS
RT
CLK
Equal
SrcA
SrcB
ALU
Result
WriteData
AluResult
数据
存储器
20:16
15:11
rd
WriteReg#
15:0
符号扩展
SignImm
<<2
4
PC+4
BranchAddr
WriteBackData
WriteReg#

分支相关执行动态

```
1  beq  $0, $0, 2          # 跳转到第 4 条指令，分支目标地址为 PC+4+8
2  add  $1, $2, $3         # 不应执行
3  addi $4, $5, $6         # 不应执行
4  bne  $1, $2, 2          # 跳转到第 7 条指令，假设两寄存器不相等
5  lw   $5, 4($1)          # 不应执行
6  sw   $6, 8($1)          # 不应执行
```

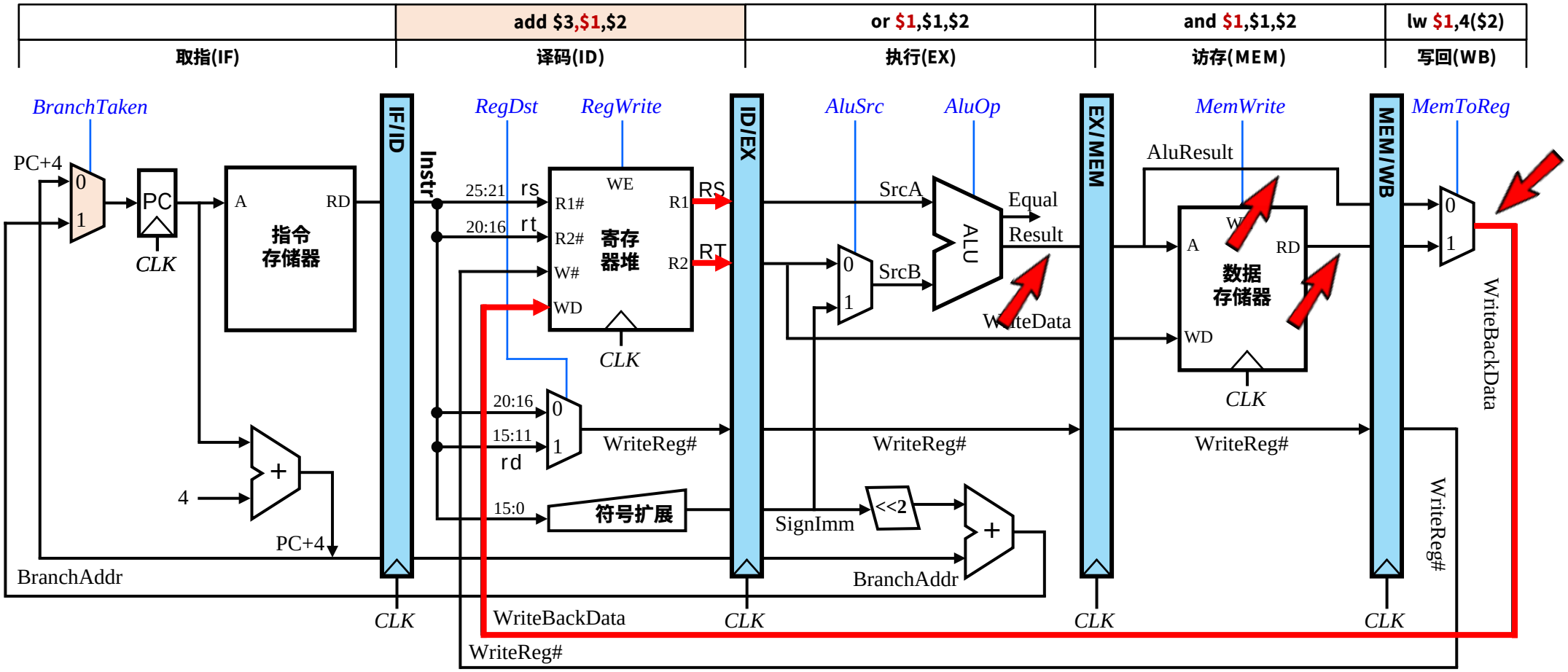
clks	IF	ID	EX	MEM	WB
1	beq		EX段执行分支		
2	add	beq			
3	addi	add	beq		
4	bne			beq	
5	lw	bne			beq
6	sw	lw	bne		
7	New			bne	

clks	IF	ID	EX	MEM	WB
1	beq		ID段执行分支		
2	add	beq			
3	bne		beq		
4	lw	bne		beq	
5	New		bne		beq
6					
7					

数据相关处理机制

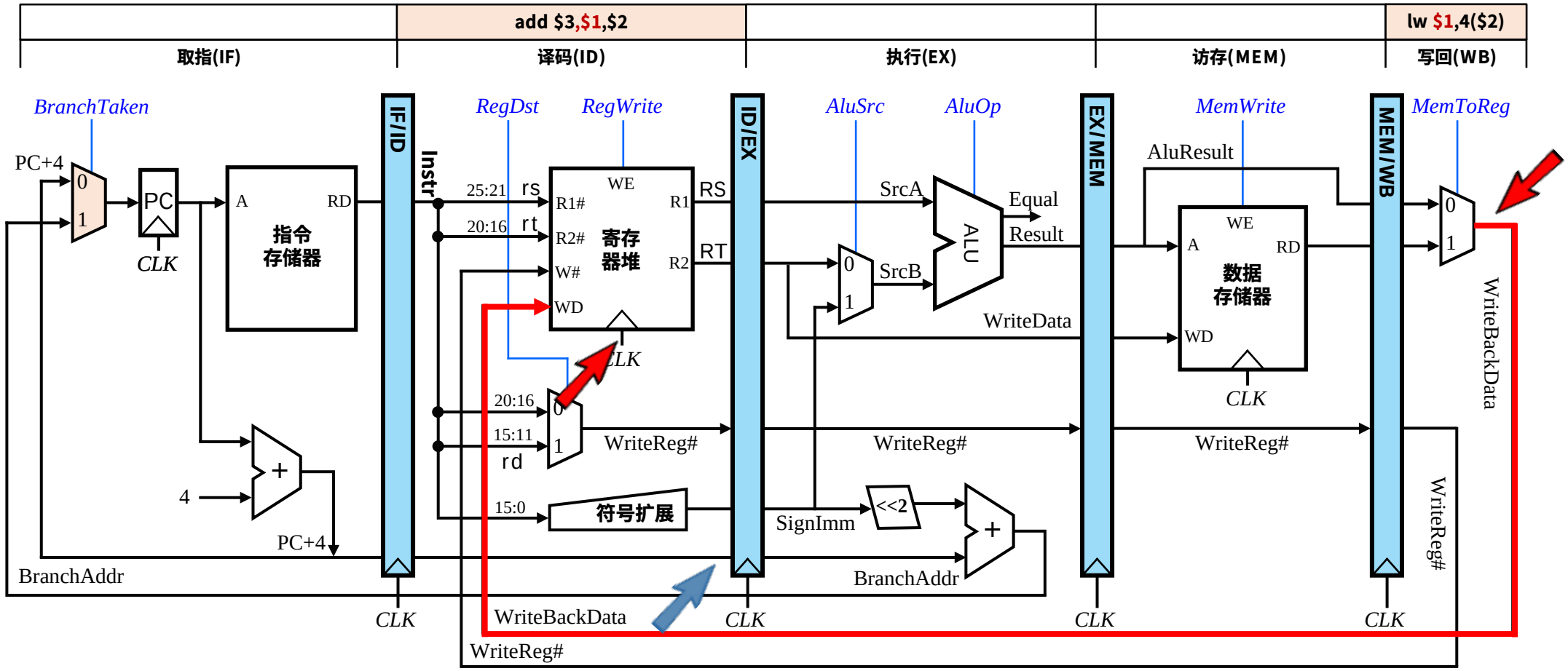
- 软件方法（编译器完成）
  - 插入空指令
  - 调整程序顺序，使相关性在流水线中消失
- 硬件方法
  - 插入气泡（空操作）
    - 向后段插入气泡（接口信号清零）
    - 向前给出阻塞信号（流水线停顿）避免当前指令被新指令取代
  - 数据重定向bypass（数据旁路）
    - 将后端处理后的数据（还没来得及写回）重定向
    - 数据在哪就从哪送到运算器

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

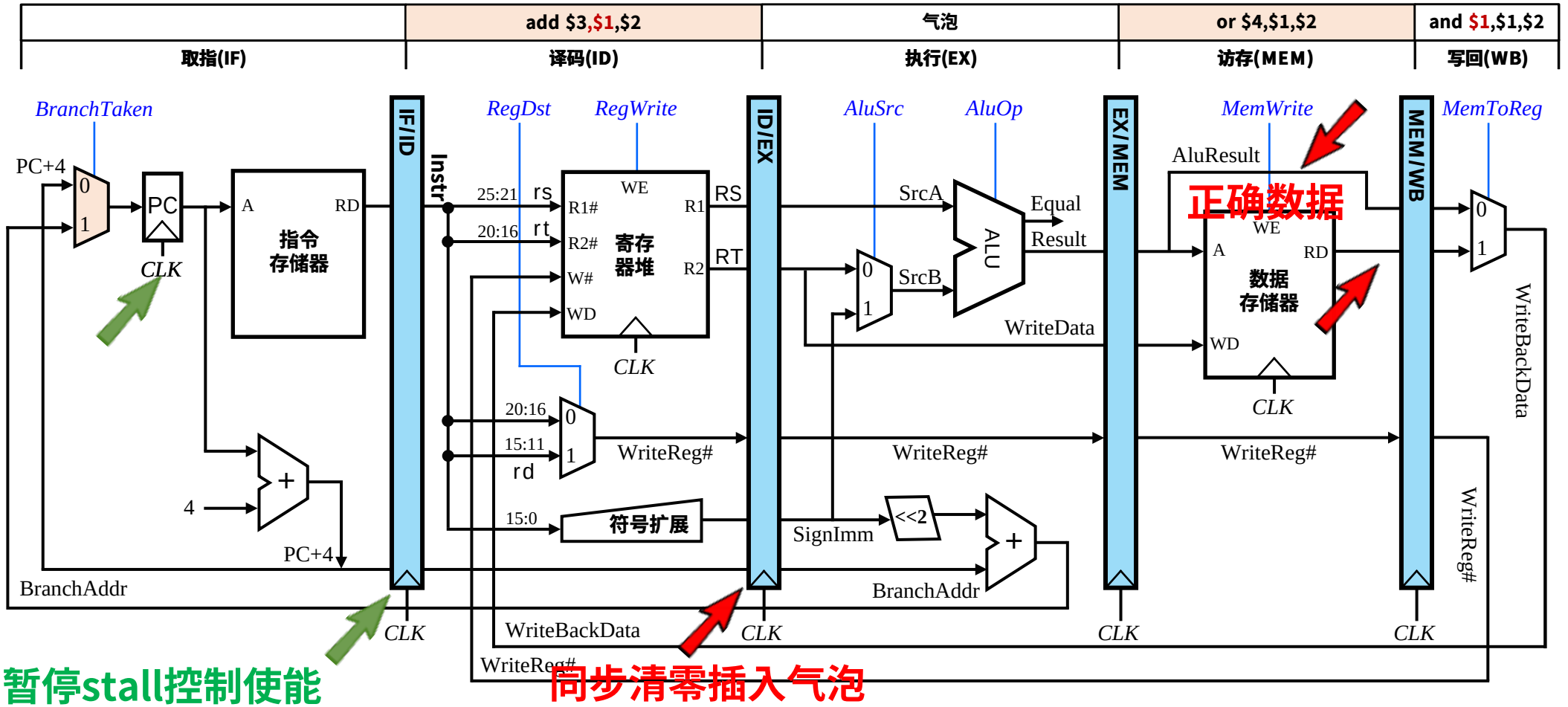
ID段与WB段数据相关消除



先写后读，寄存器文件下跳沿写入，流水接口上跳沿有效

ID段与MEM段数据相关消除

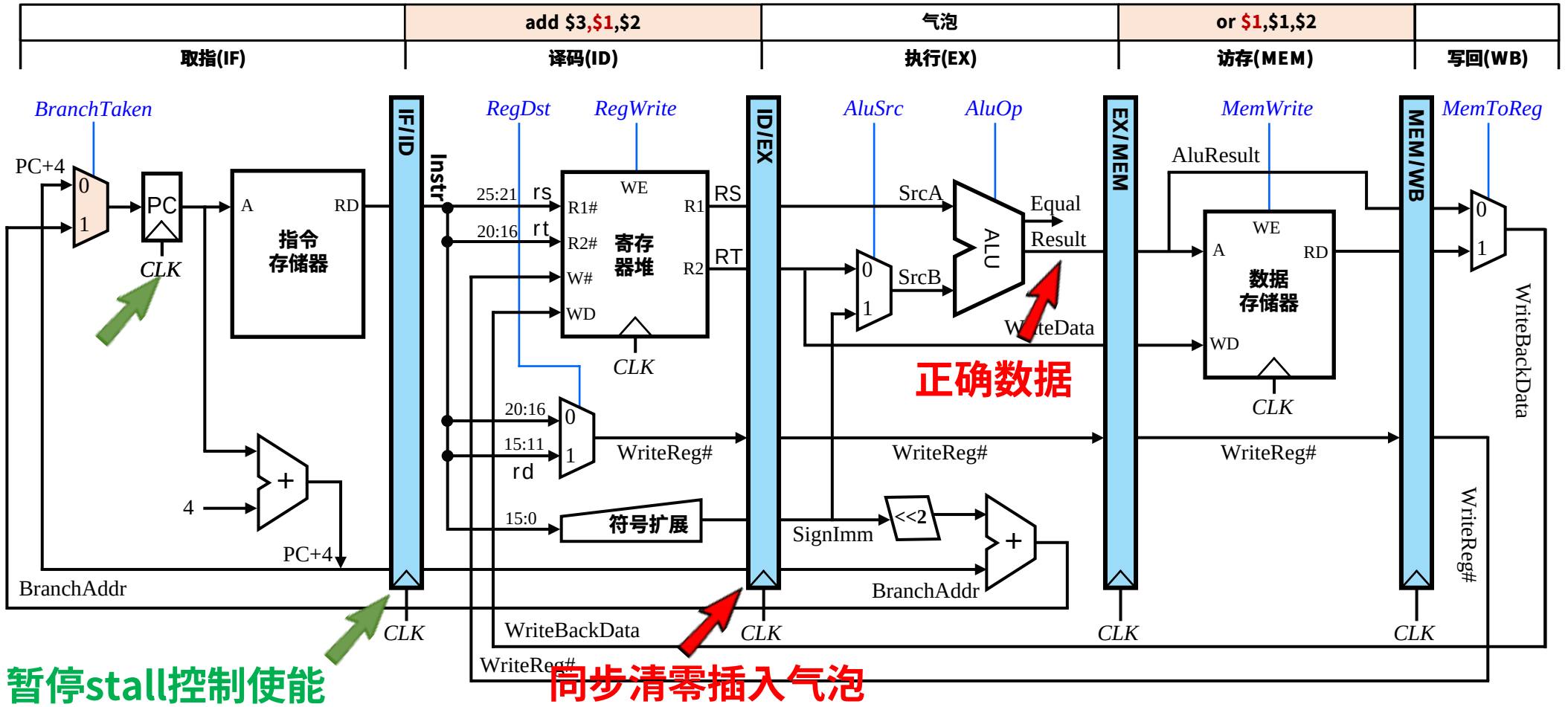
下一个时钟相关性？



IF、ID段暂停等待数据写回，EX段同步插入气泡

ID段与EX段数据相关消除

下一个时钟相关性？



IF、ID段暂停等待数据写回，EX段同步插入气泡

数据相关执行动态（插入气泡）

```
1  lw  $5, 4($1)        # $5 为目的寄存器
2  add $6, $5, $7       # $5 依赖第 1 条指令的访存结果
3  sub $1, $2, $3       # 无数据相关
4  or  $7, $6, $7       # $6 依赖第 2 条指令的运算结果
5  and $9, $7, $6       # $7 依赖第 4 条指令的运算结果
```

clks	IF	ID	EX	MEM	WB
3	sub $1, $2, $3	add $6, **$5**, $7	lw **$5**, 4($1)		
4	sub $1, $2, $3	add $6, **$5**, $7		lw **$5**, 4($1)	
5	sub $1, $2, $3	add $6, **$5**, $7			lw **$5**, 4($1)
6	or $7, $6, $7	sub $1, $2, $3	add $6, **$5**, $7		
7	and $9, $7, $6	or $7, **$6**, $7	sub $1, $2, $3	add $6, **$5**, $7	
8	and $9, $7, $6	or $7, **$6**, $7		sub $1, $2, $3	add $6, **$5**, $7
9	Next Instr	and $9, **$7**, $6	or **$7**, **$6**, $7		sub $1, $2, $3
10	Next Instr	and $9, **$7**, $6		or **$7**, **$6**, $7	
11	Next Instr	and $9, **$7**, $6			or **$7**, **$6**, $7

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

```
DataHazzard = RsUsed & (rs≠0) & EX.RegWrite  & (rs==EX.WriteReg#)
            + RtUsed & (rt≠0) & EX.RegWrite  & (rt==EX.WriteReg#)
            + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#)
            + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)
```

stall = DataHazzard

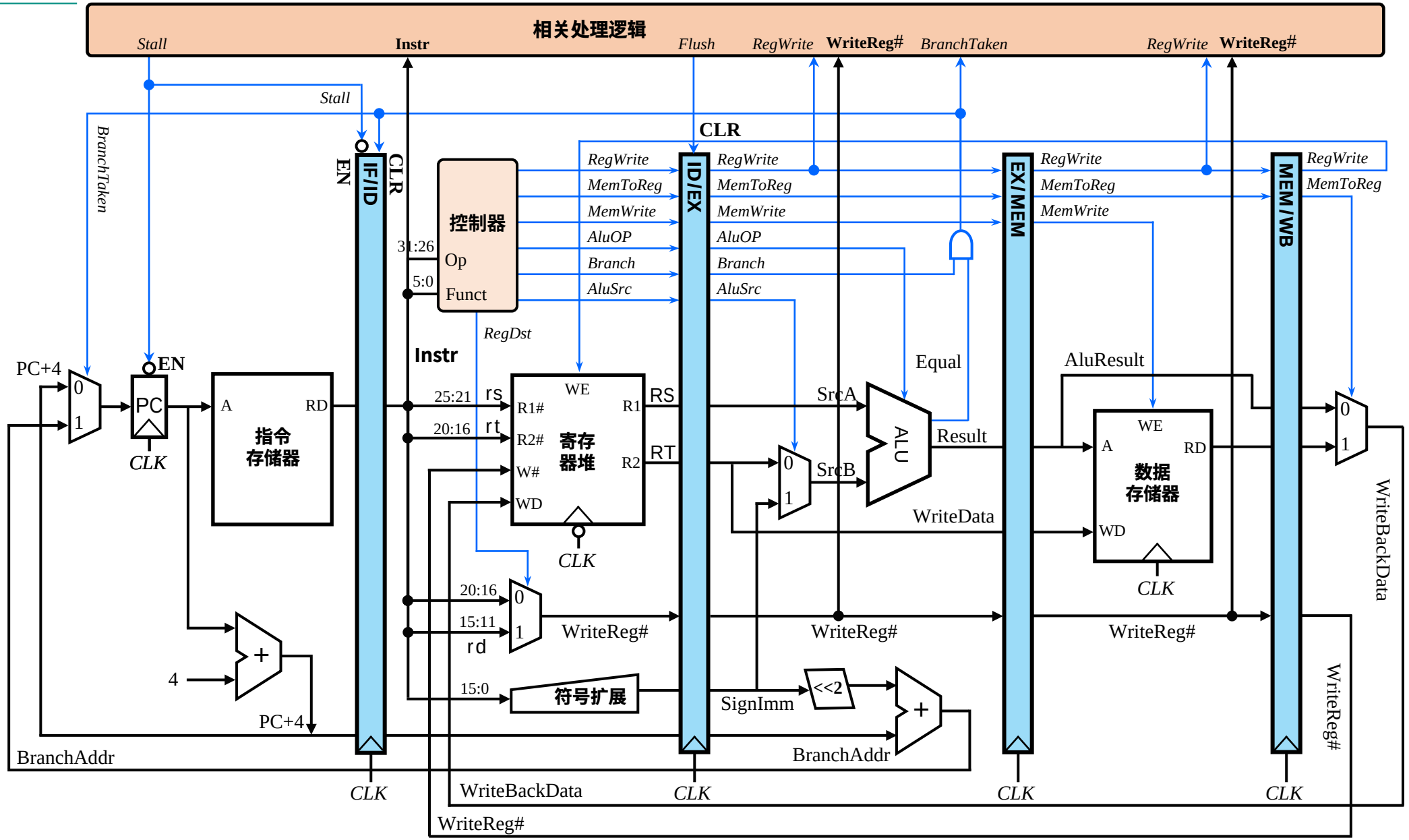
PC.EN = ~Stall

IF/ID.EN = ~Stall

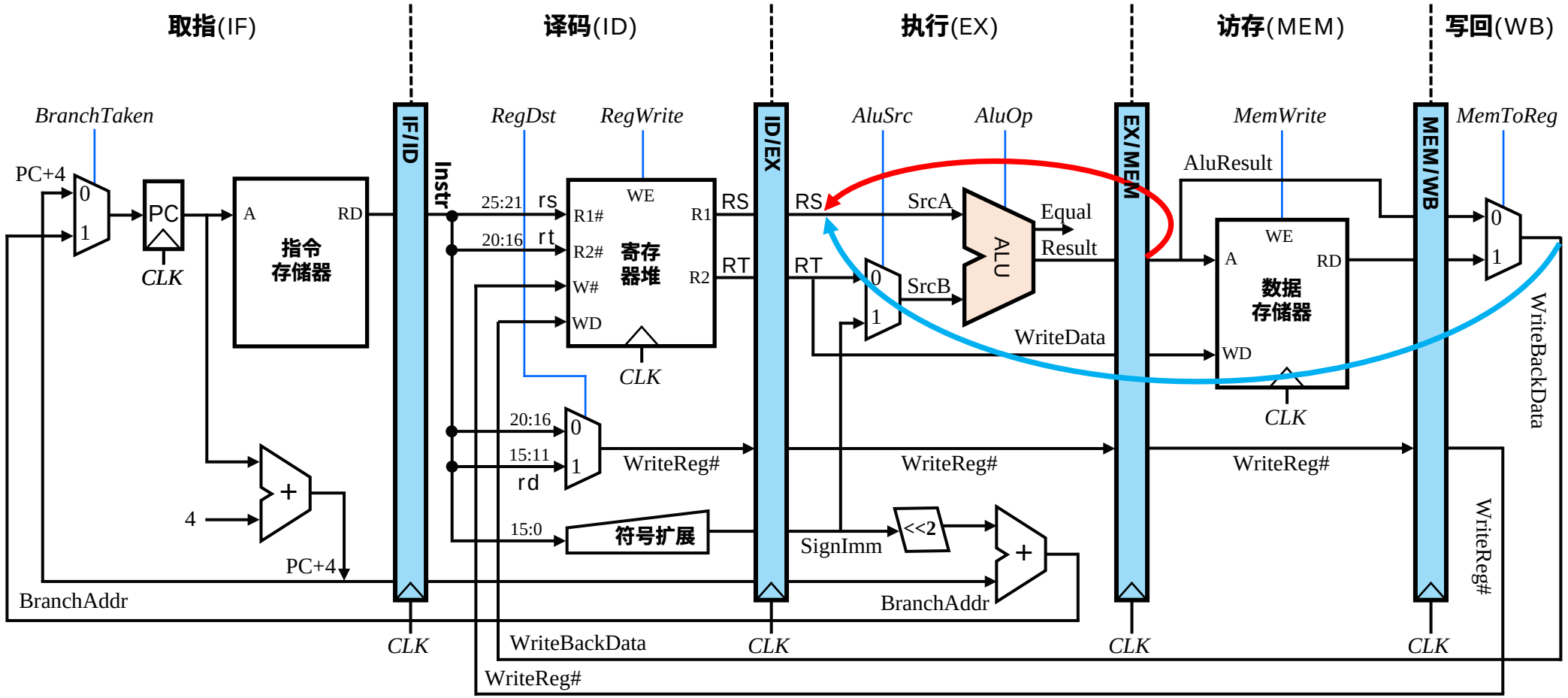
IF/ID.CLR = BranchTaken

ID/EX.CLR = Flush = BranchTaken + DataHazzard

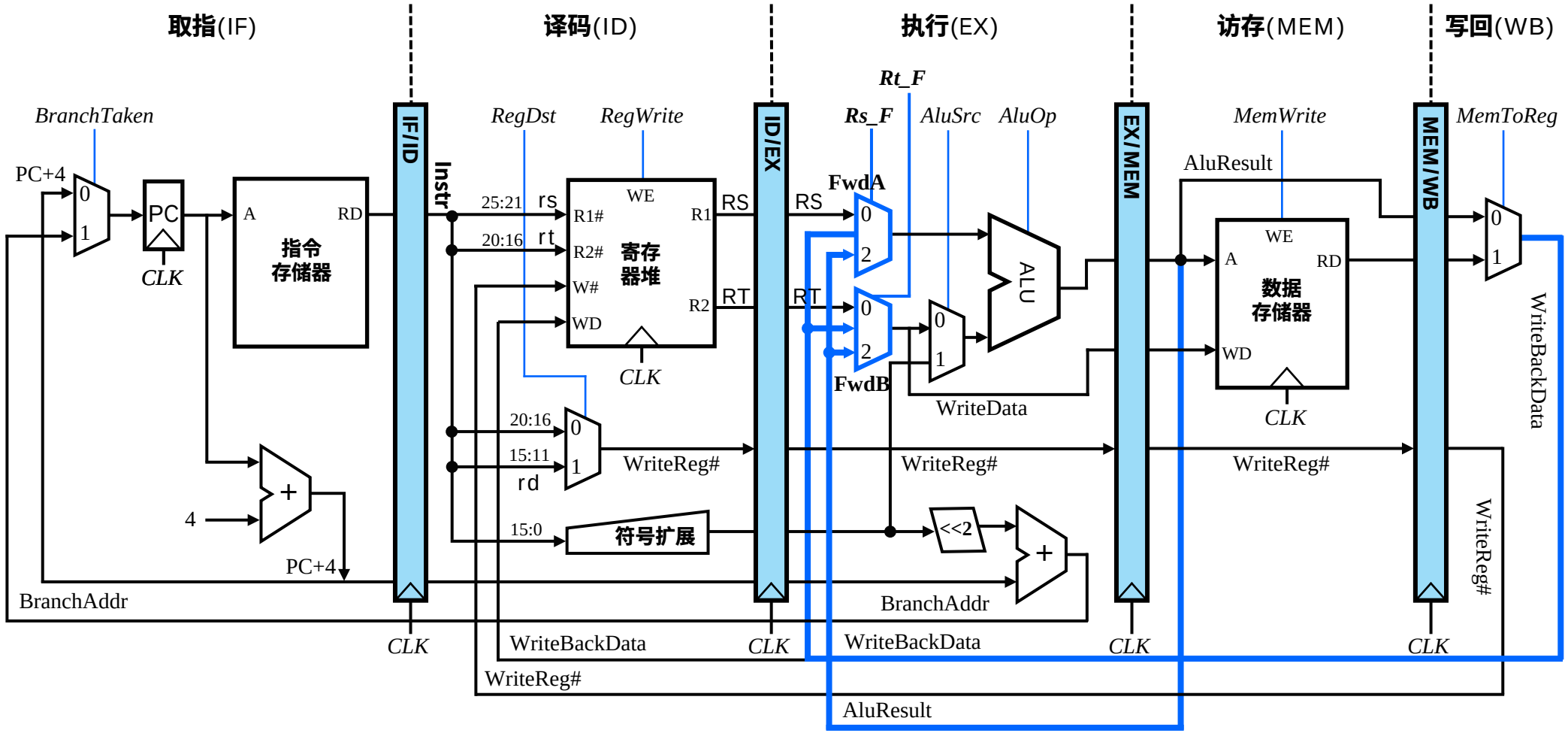
气泡流水线顶层视图


相关处理逻辑
Stall
Instr
Flush
RegWrite
WriteReg#
BranchTaken
RegWrite
WriteReg#
Stall
BranchTaken
CLR
EN
IF/ID
CLR
控制器
31:26
Op
5:0
Funct
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RegDst
ID/EX
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
EX/MEM
RegWrite
MemToReg
MemWrite
MEM/WB
RegWrite
MemToReg
Instr
PC+4
0
1
EN
PC
CLK
A
RD
指令
存储器
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
R1
R2
寄存
器堆
RS
RT
CLK
Equal
SrcA
SrcB
ALU
Result
WriteData
AluResult
WE
A
RD
数据
存储器
WD
CLK
20:16
15:11
rd
0
1
WriteReg#
WriteReg#
WriteReg#
4
+
PC+4
15:0
符号扩展
SignImm
<<2
+
BranchAddr
BranchAddr
WriteBackData
WriteBackData
WriteReg#
CLK
CLK
CLK
CLK

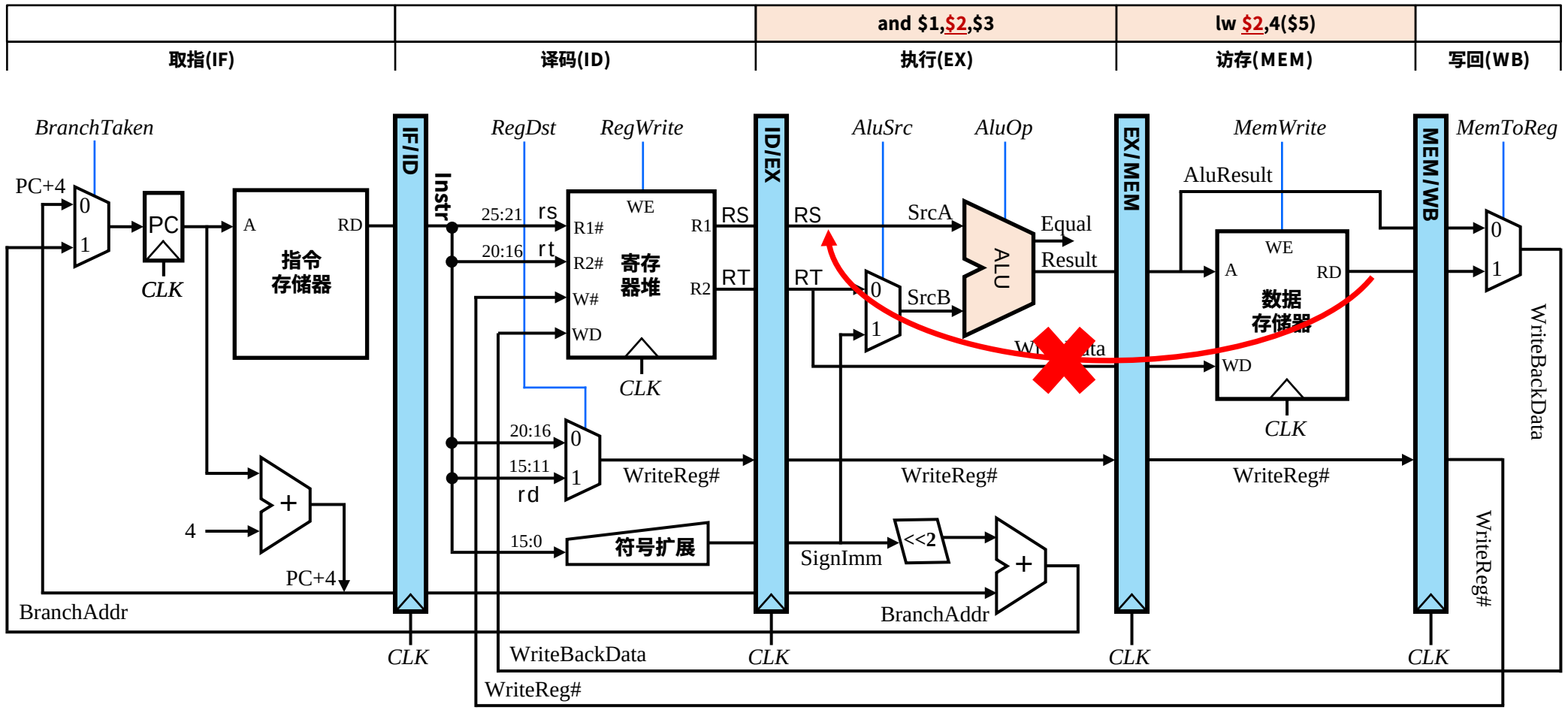
数据重定向


取指(IF)
译码(ID)
执行(EX)
访存(MEM)
写回(WB)
IF/ID
ID/EX
EX/MEM
MEM/WB
BranchTaken
PC+4
PC
CLK
指令
存储器
A
RD
Instr
RegDst
RegWrite
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
寄存
器堆
R1
R2
RS
RT
20:16
15:11
rd
WriteReg#
15:0
符号扩展
4
PC+4
BranchAddr
WriteBackData
AluSrc
AluOp
SrcA
SrcB
ALU
Equal
Result
WriteData
SignImm
<<2
MemWrite
AluResult
数据
存储器
MemToReg
WriteBackData
WriteReg#

数据重定向数据通路

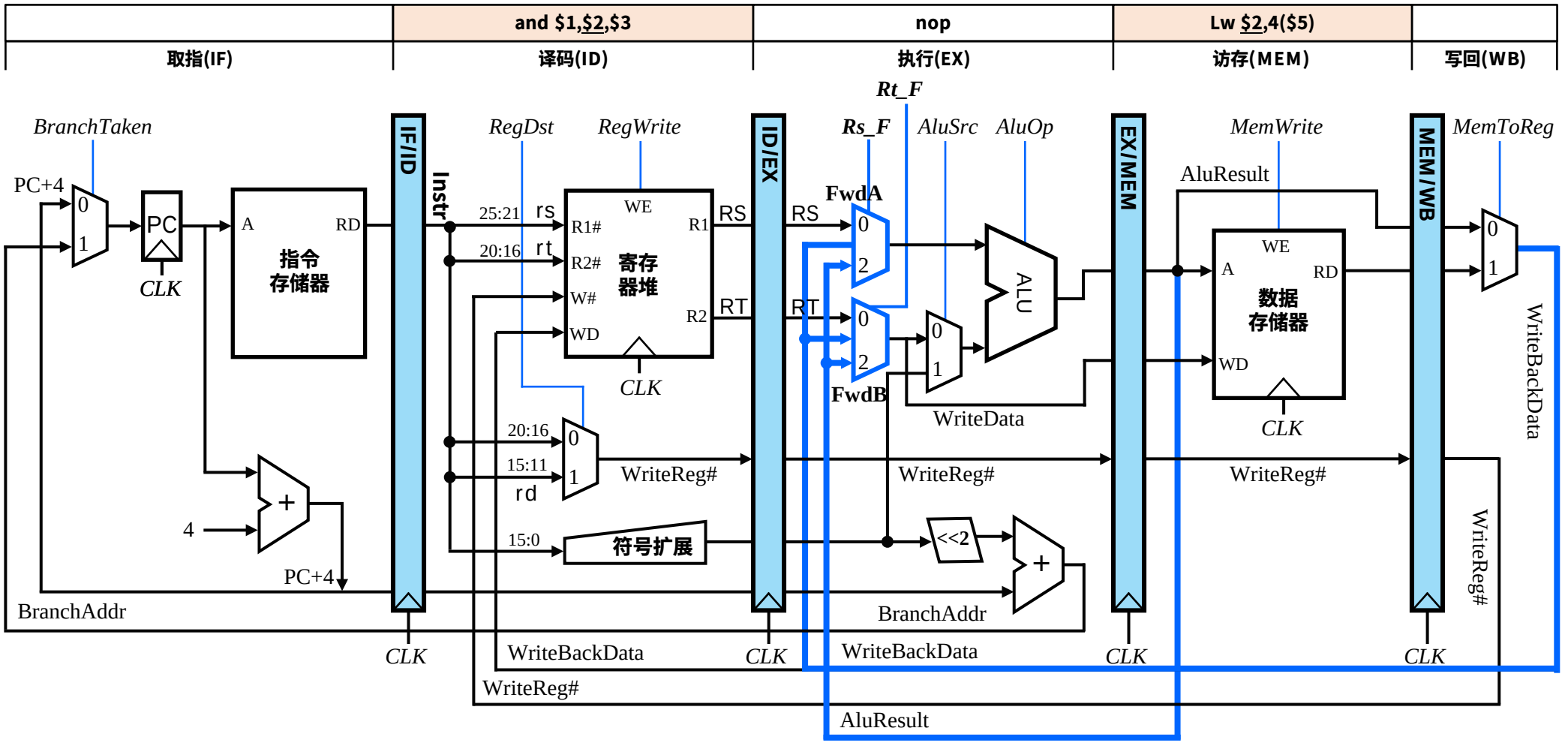

取指(IF)
译码(ID)
执行(EX)
访存(MEM)
写回(WB)
BranchTaken
PC+4
PC
CLK
指令
存储器
A
RD
IF/ID
Instr
RegDst
RegWrite
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
寄存
器堆
R1
R2
RS
RT
ID/EX
Rt_F
Rs_F
AluSrc
AluOp
FwdA
FwdB
ALU
EX/MEM
MemWrite
MemToReg
AluResult
数据
存储器
MEM/WB
WriteBackData
WriteReg#
WriteData
20:16
15:11
rd
15:0
符号扩展
<<2
4
+
PC+4
BranchAddr
WriteBackData
WriteReg#
AluResult

Load-Use相关

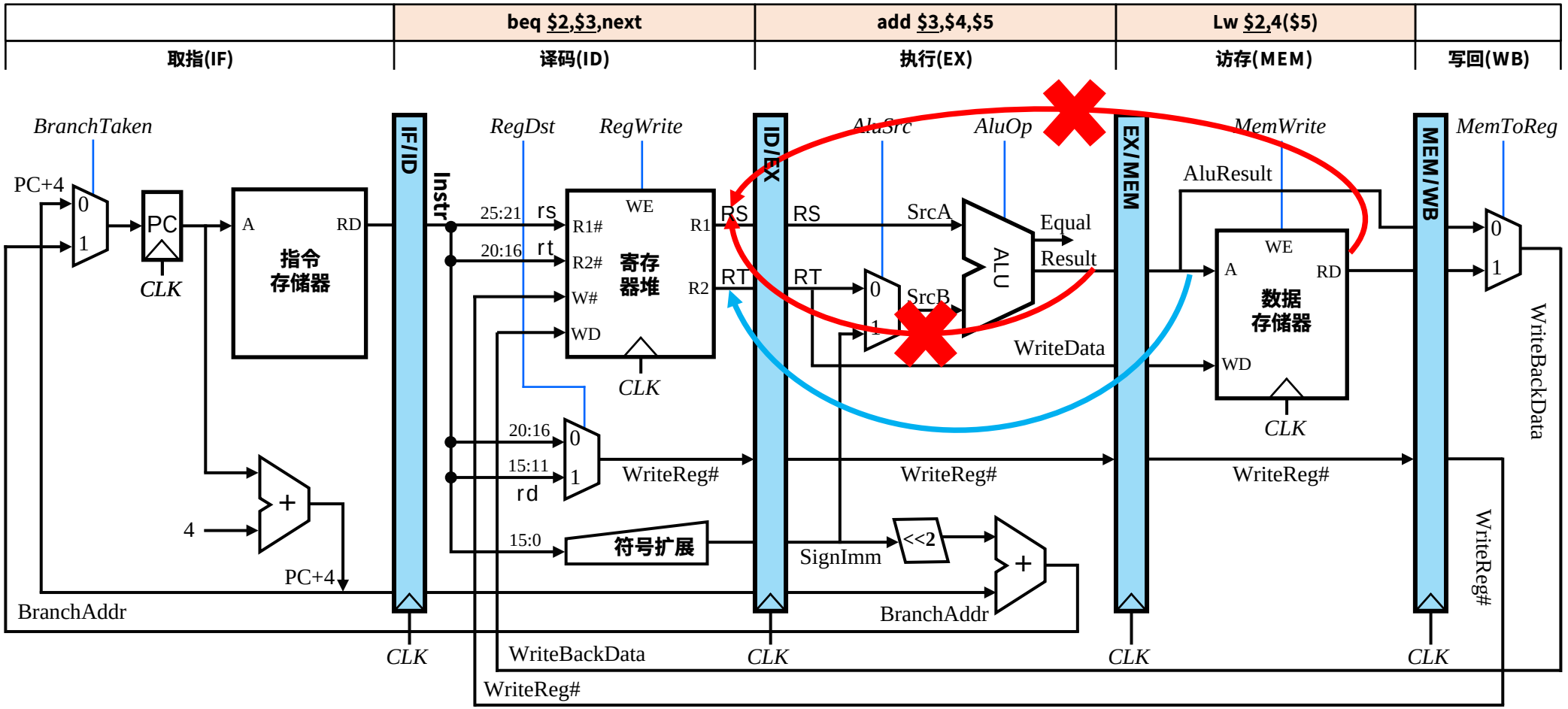



```
LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)
        + RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)
```

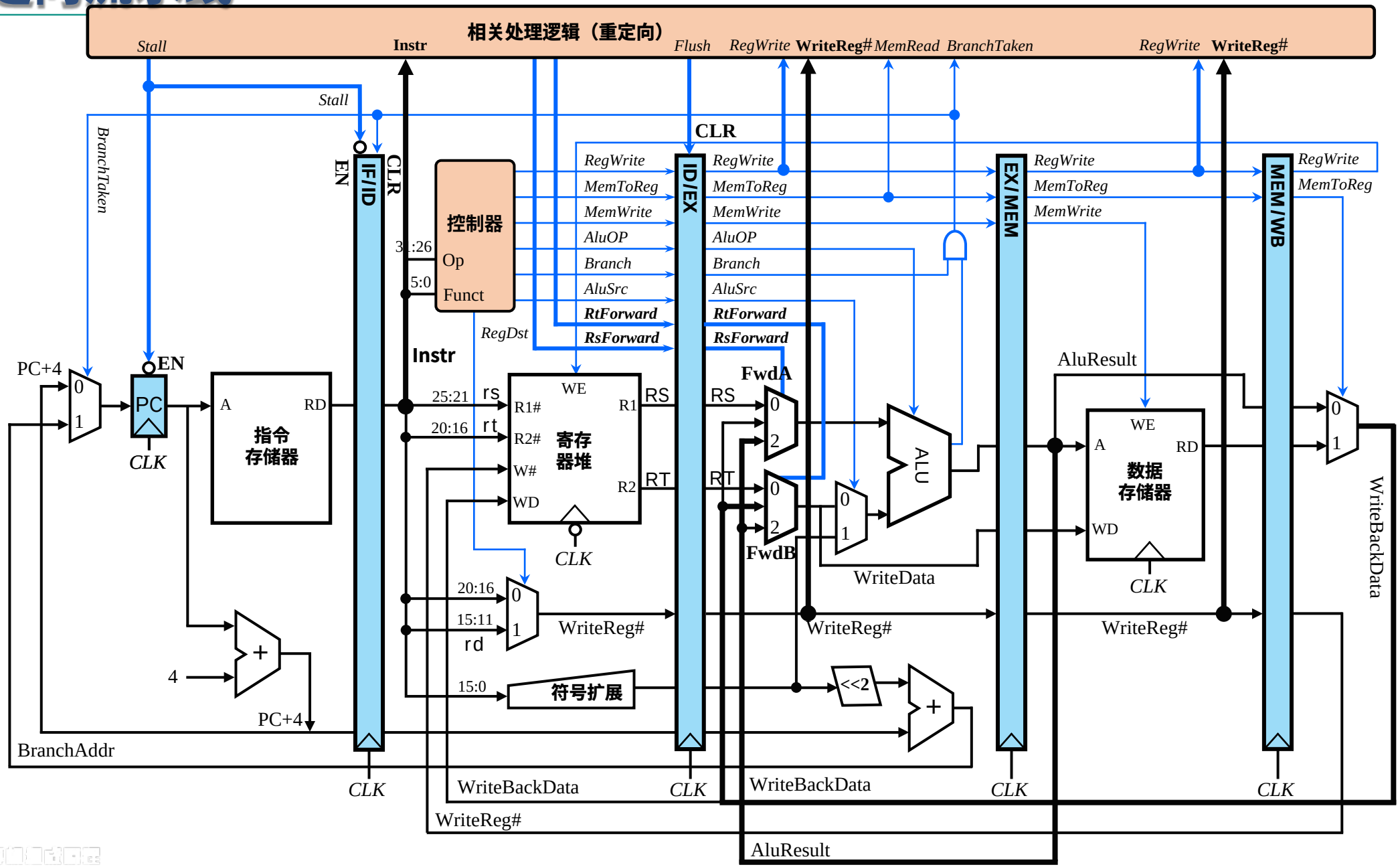
插入气泡消除Load-Use相关



ID段执行分支的重定向问题



重定向流水线


相关处理逻辑（重定向）
Stall
Instr
Flush
RegWrite
WriteReg#
MemRead
BranchTaken
RegWrite
WriteReg#
Stall
BranchTaken
EN
IF/ID
CLR
CLR
控制器
Op
Funct
31:26
5:0
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
ID/EX
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
EX/MEM
RegWrite
MemToReg
MemWrite
MEM/WB
RegWrite
MemToReg
RegDst
Instr
PC+4
EN
PC
CLK
A
RD
指令
存储器
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
R1
R2
寄存
器堆
RS
RT
CLK
FwdA
FwdB
RS
RT
ALU
AluResult
WriteData
WE
A
RD
数据
存储器
WD
CLK
WriteBackData
20:16
15:11
rd
WriteReg#
WriteReg#
WriteReg#
4
PC+4
15:0
符号扩展
<<2
BranchAddr
CLK
WriteBackData
CLK
WriteBackData
CLK
CLK
WriteReg#
AluResult

数据相关执行动态(重定向)

```
1  lw  $5, 4($1)          # $5 为目的寄存器
2  add $6, $5, $7         # $5 依赖第 1 条指令的访存结果
3  sub $1, $2, $3         # 无数据相关
4  or  $7, $6, $7         # $6 依赖第 2 条指令的运算结果
5  and $9, $7, $6         # $7 依赖第 4 条指令的运算结果
```

clks	IF	ID	EX	MEM	WB
3	sub $1, $2, $3	add $6, **$5**, $7	lw **$5**, 4($1)		
4	sub $1, $2, $3	add $6, **$5**, $7		lw **$5**, 4($1)	
5	or $7, $6, $7	sub $1, $2, $3	add $6, **$5**, $7		lw **$5**, 4($1)
6	and $9, $7, $6	or $7, **$6**, $7	sub $1, $2, $3	add **$6**, **$5**, $7	
7	Next Instr	and $9, **$7**, $6	or **$7**, $6, $7	sub $1, $2, $3	add $6, **$5**, $7
8	…	Next Instr	and $9, $7, $6	or $7, $6, $7	sub $1, $2, $3

动态分支预测流水线

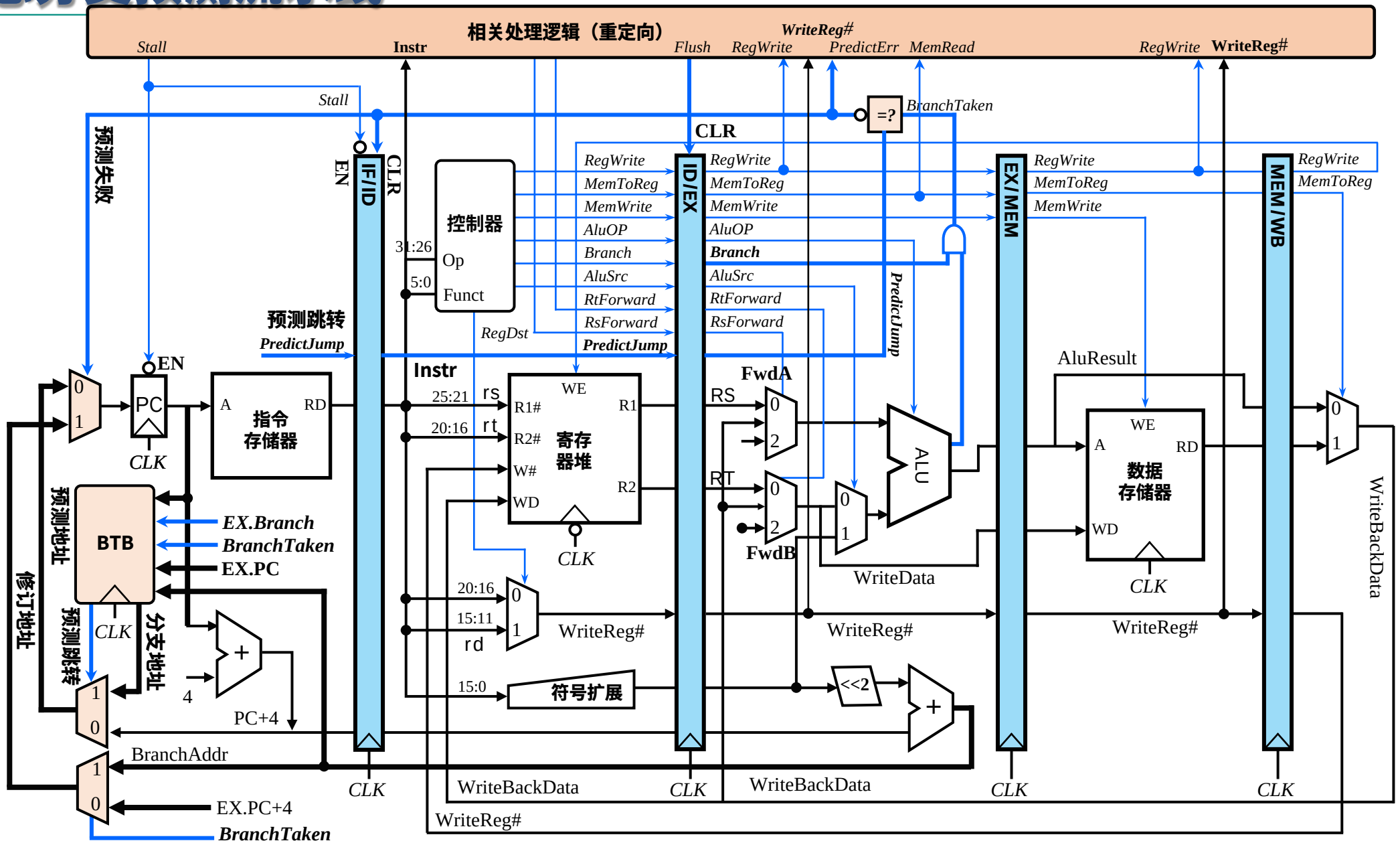
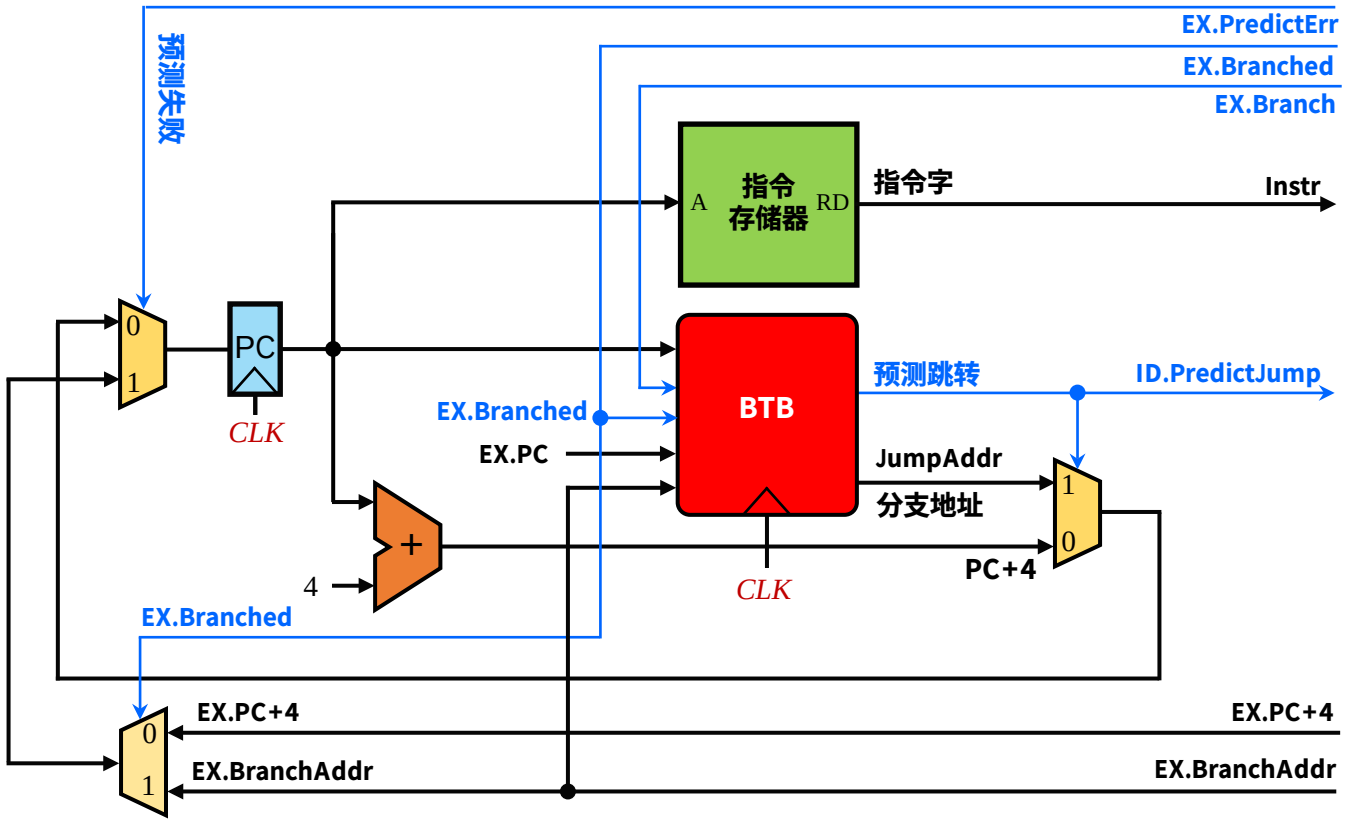
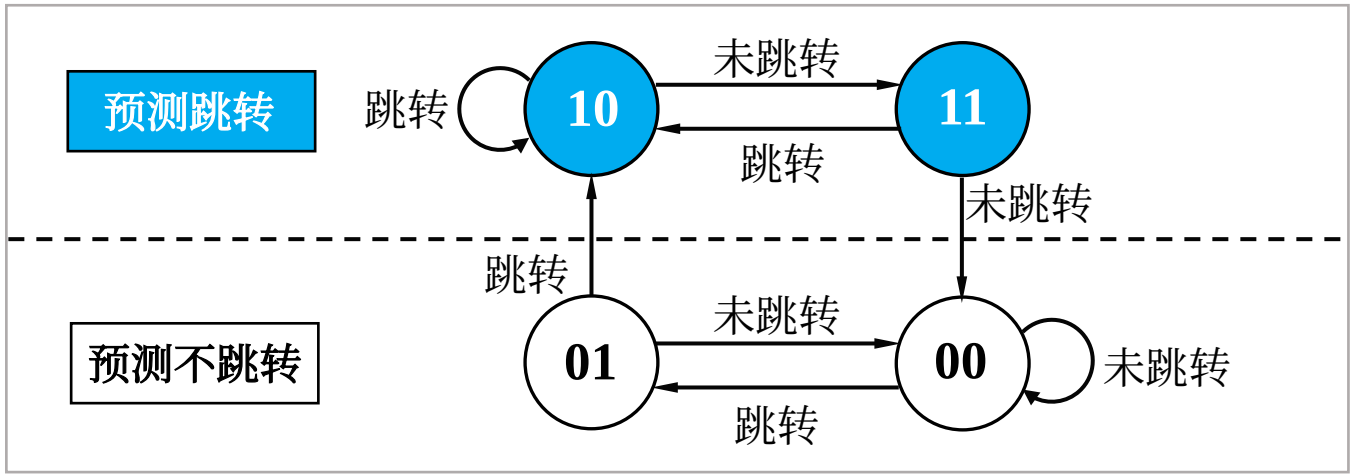

相关处理逻辑（重定向）
Stall
Instr
Flush
RegWrite
WriteReg#
PredictErr
MemRead
RegWrite
WriteReg#
Stall
=?
BranchTaken
预测失败
CLR
EN
IF/ID
CLR
控制器
31:26
Op
5:0
Funct
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
PredictJump
ID/EX
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
PredictJump
EX/MEM
RegWrite
MemToReg
MemWrite
MEM/WB
RegWrite
MemToReg
预测跳转
PredictJump
RegDst
Instr
FwdA
AluResult
EN
PC
CLK
A
RD
指令
存储器
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
R1
R2
寄存
器堆
CLK
RS
RT
FwdB
ALU
WriteData
WE
A
RD
数据
存储器
WD
CLK
WriteBackData
预测地址
BTB
EX.Branch
BranchTaken
EX.PC
修订地址
预测跳转
CLK
分支地址
+
4
PC+4
20:16
15:11
rd
WriteReg#
WriteReg#
WriteReg#
15:0
符号扩展
<<2
+
BranchAddr
EX.PC+4
BranchTaken
CLK
WriteBackData
CLK
WriteBackData
CLK
CLK
WriteReg#

图2.31


预测失效
EX.PredictErr
EX.Branched
EX.Branch
指令
存储器
A
RD
指令字
Instr
0
1
PC
CLK
EX.Branched
EX.PC
BTB
预测跳转
ID.PredictJump
JumpAddr
分支地址
1
0
+
4
CLK
PC+4
EX.Branched
0
1
EX.PC+4
EX.BranchAddr
EX.PC+4
EX.BranchAddr


预测跳转
跳转
10
未跳转
11
跳转
未跳转
跳转
预测不跳转
01
未跳转
00
未跳转
跳转

本章主要内容

- 7.1 流水线概述
- 7.2 流水线数据通路
- 7.3 流水线冲突与处理
- **7.4 流水线的异常与中断**
- 7.5 指令集并行技术

7.4 流水线的异常与中断

- 中断类别
  - 同步中断（指令异常）、异步中断（外设中断）

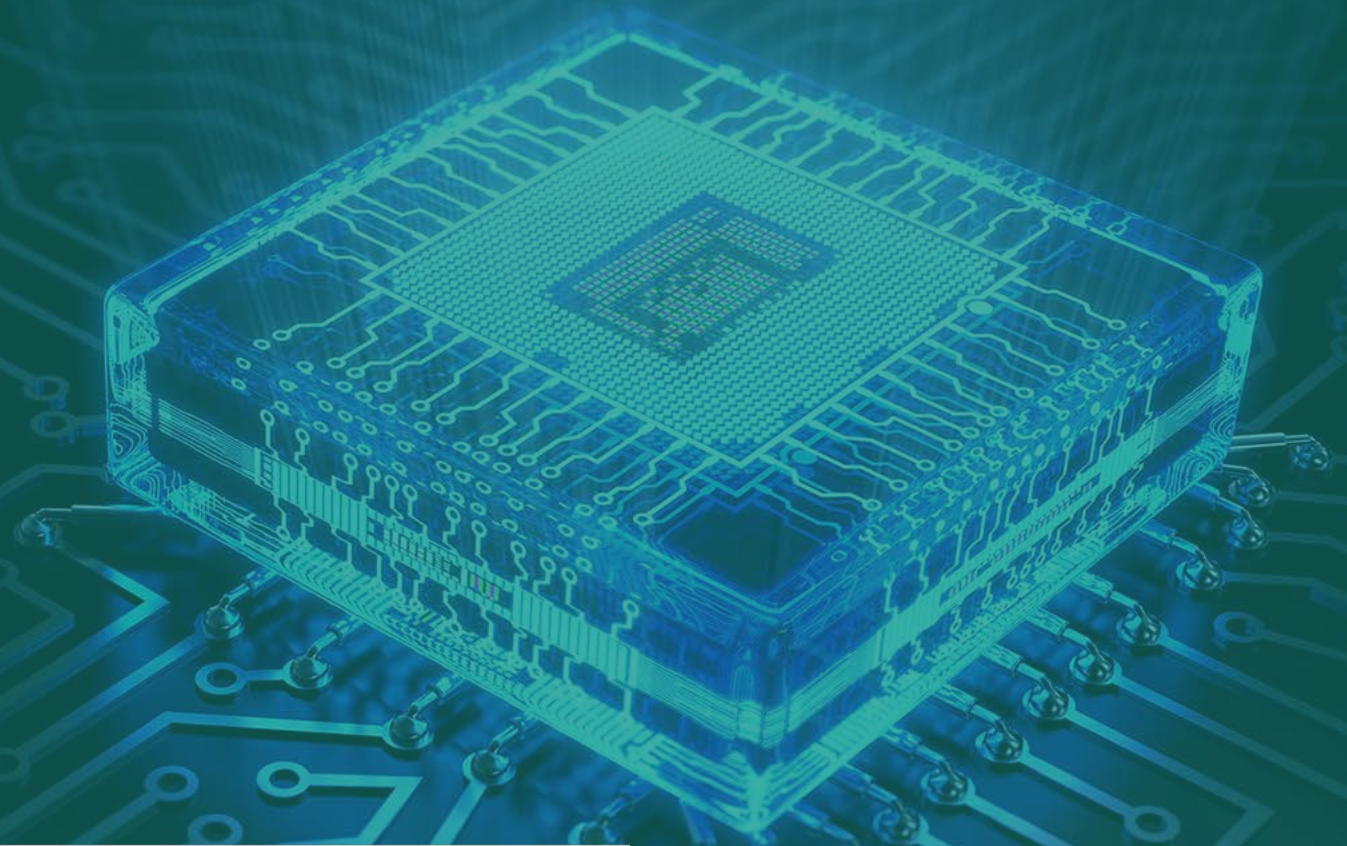
#	IF	ID	EX	MEM	WB
1	缺页或TLB异常	未定义指令	算术运算溢出	缺页或TLB异常	无异常
2	未对齐指令地址	除数为零	自陷异常	未对齐数据地址	
3	存储保护违例			存储保护违例	

- 异步中断处理
  - 保存断点（下一条指令地址） à 设置中断原因 à 关中断 à 指令清空
- 同步中断处理
  - 保存断点（当前异常指令地址） à 设置中断原因 à 关中断 à 指令清空

7.5 指令集并行技术

- 超流水线(*superpipelined*)
  - 技术主要通过增加流水线功能段数目，尽可能细化减少各段关键延迟，从而提高流水线主频的方式提升流水线性能，例如Pentium pro的流水线就多达14段
- 多发射 (*multiple issue*)
  - 复制计算机内部功能部件的数量，使各流水功能段能同时处理多条指令，处理器一次可以发射多条指令进入流水线进行处理。引起更多的相关性，冲突冒险问题更难处理
- 静态多发射，
  - 冲突冒险全部交给编译器静态解决
- 动态多发射
  - 由硬件动态处理多发射流水线运行过程中出现的各种冲突冒险
  - 也称为超标量 (*superscalar*)技术

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST



THANKS

计算机组成原理