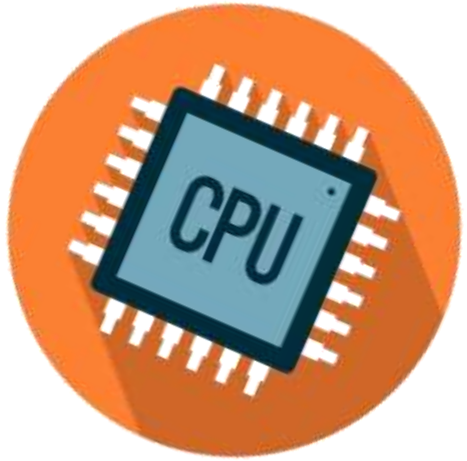
计算机组成原理


# 七、指令流水线

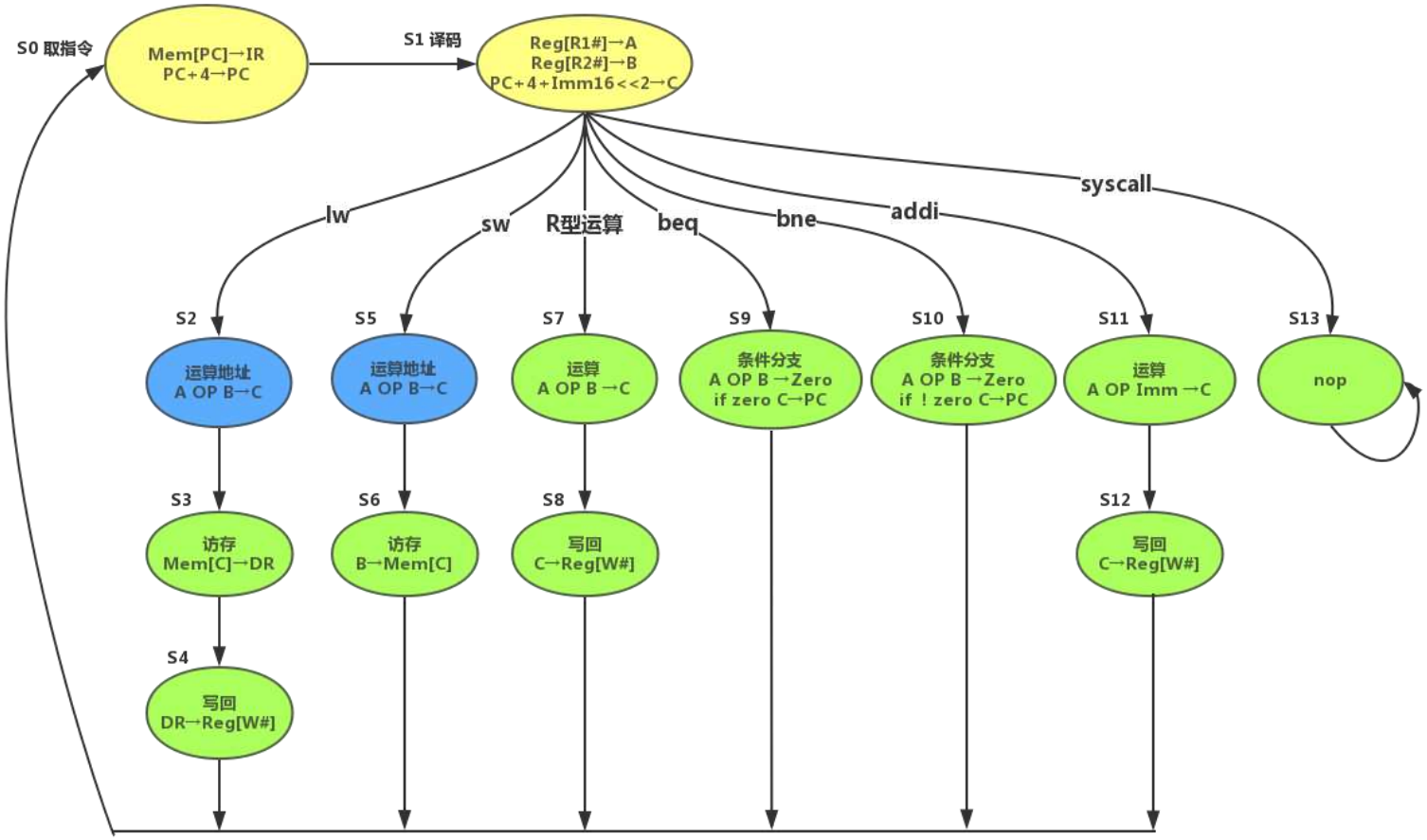


# 本章主要内容

- **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

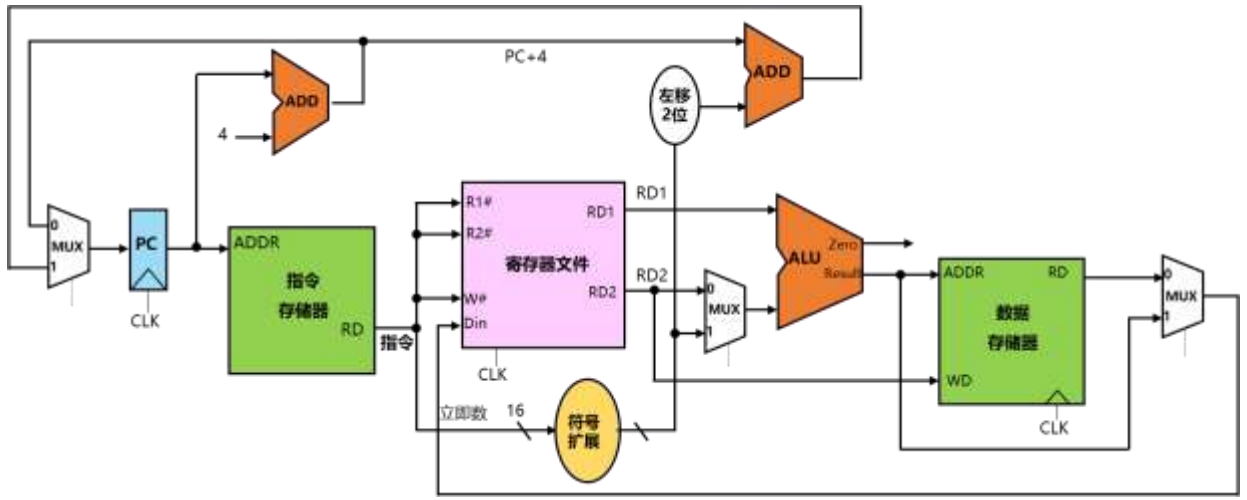


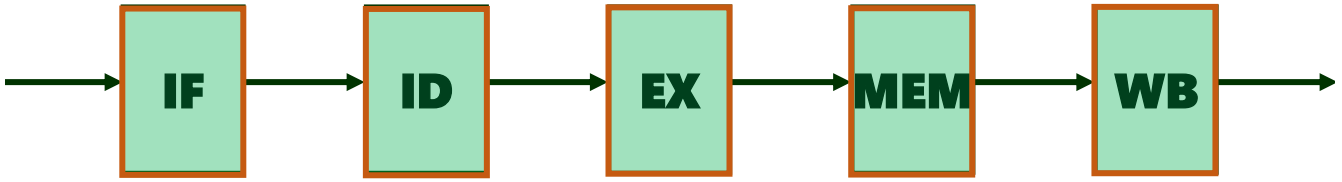
# 流水线

- **流水线技术**：计算机中的流水线技术是把一个复杂的任务分解为若干个阶段，每个阶段与其他阶段并行处理
- **指令流水线**：将流水线技术应用于指令的解释执行过程

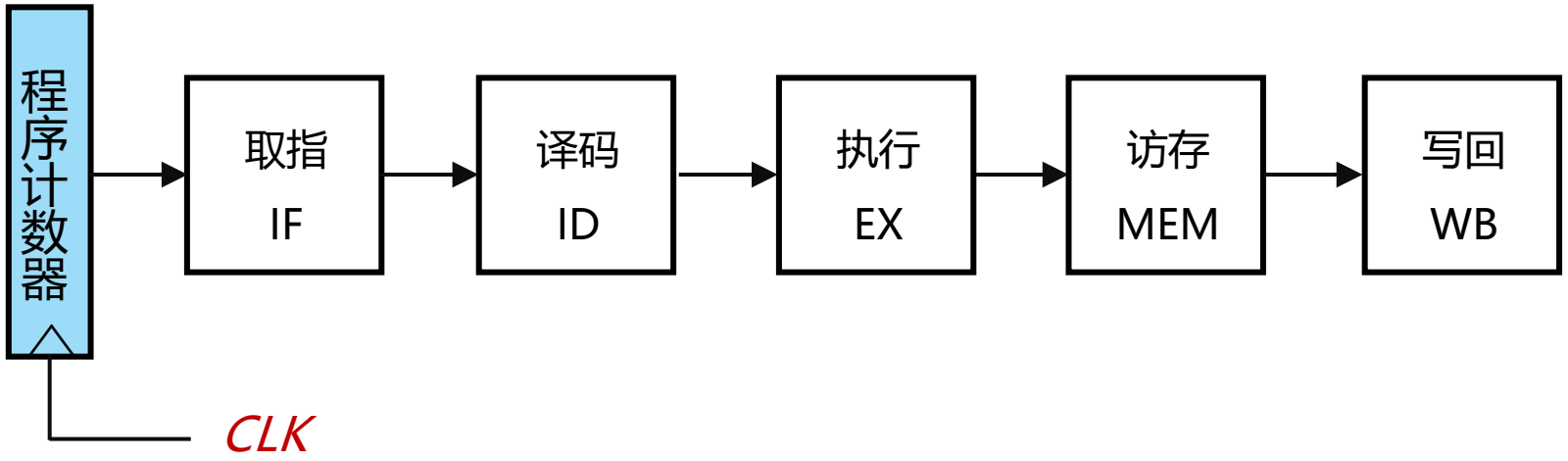
# 单周期指令运行动态

- **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)

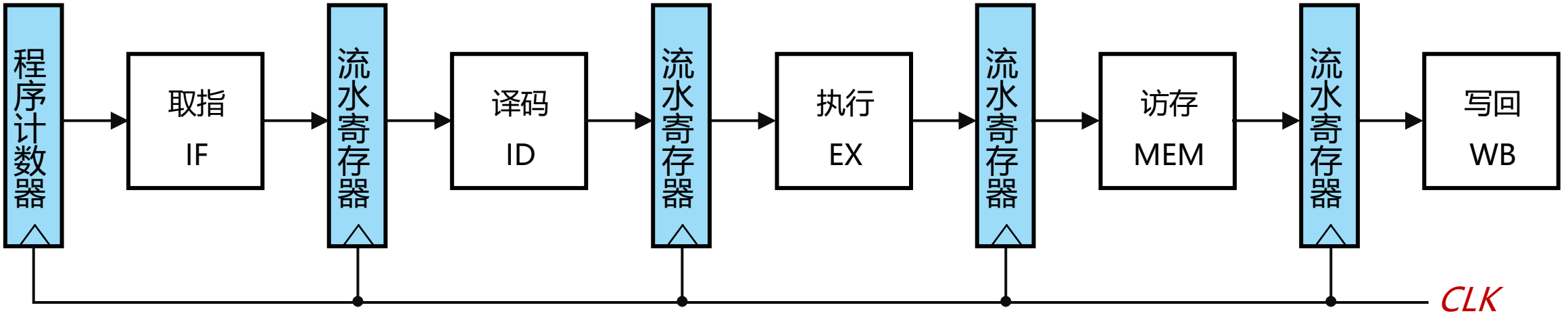


# MIPS单周期逻辑架构

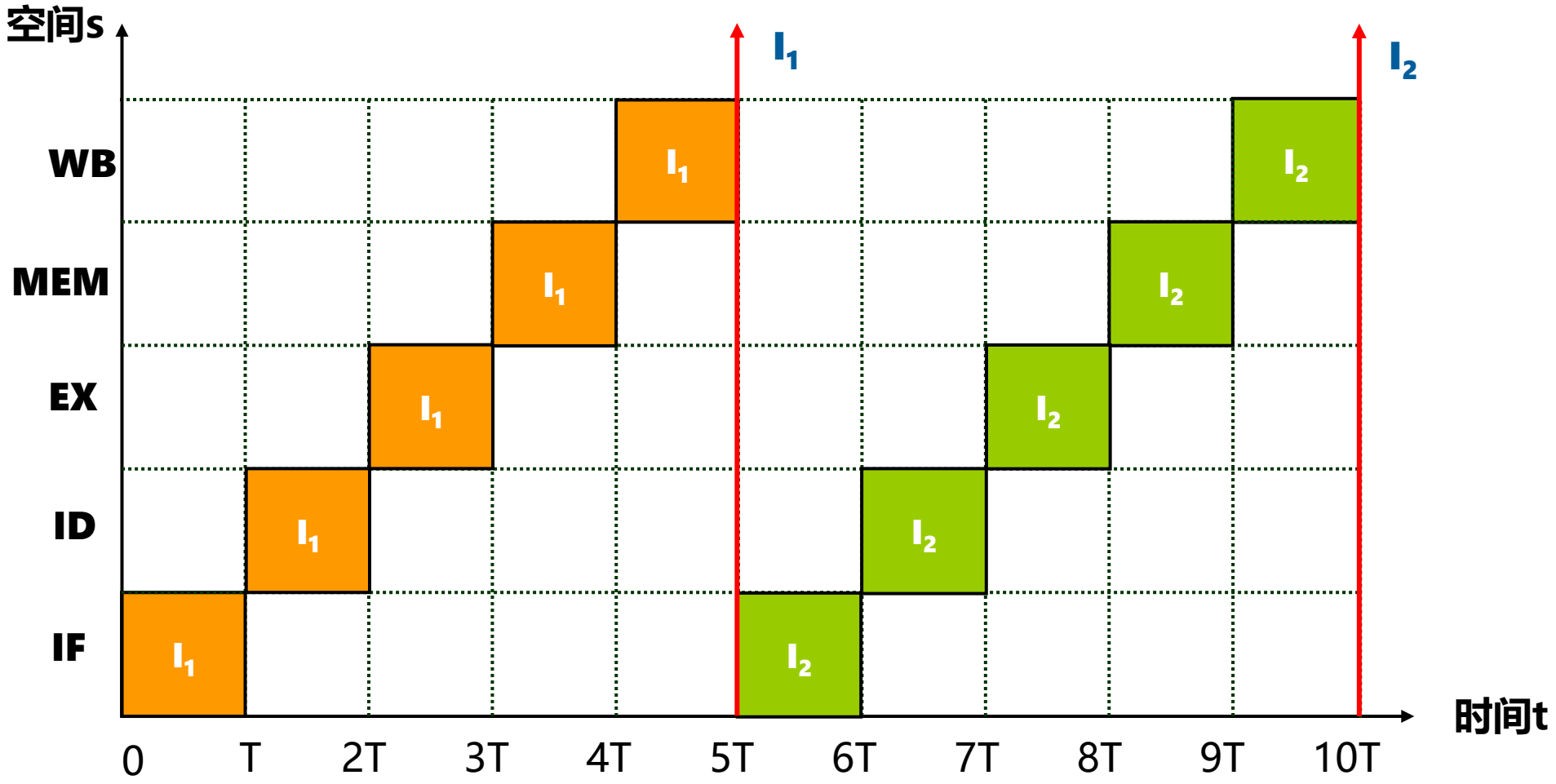


# MIPS指令流水线逻辑结构




- 每个执行阶段的后面都需要增加一个**流水寄存器**，用于锁存本段处理完成的所有数据或结果，以保证本段的执行结果能在下一个时钟周期给下一个阶段使用
- 程序计数器、流水寄存器均采用统一公共时钟进行同步，每来一个时钟，各段组合逻辑功能部件处理完成的数据都将锁存到段尾的流水寄存器中，作为后段的输入

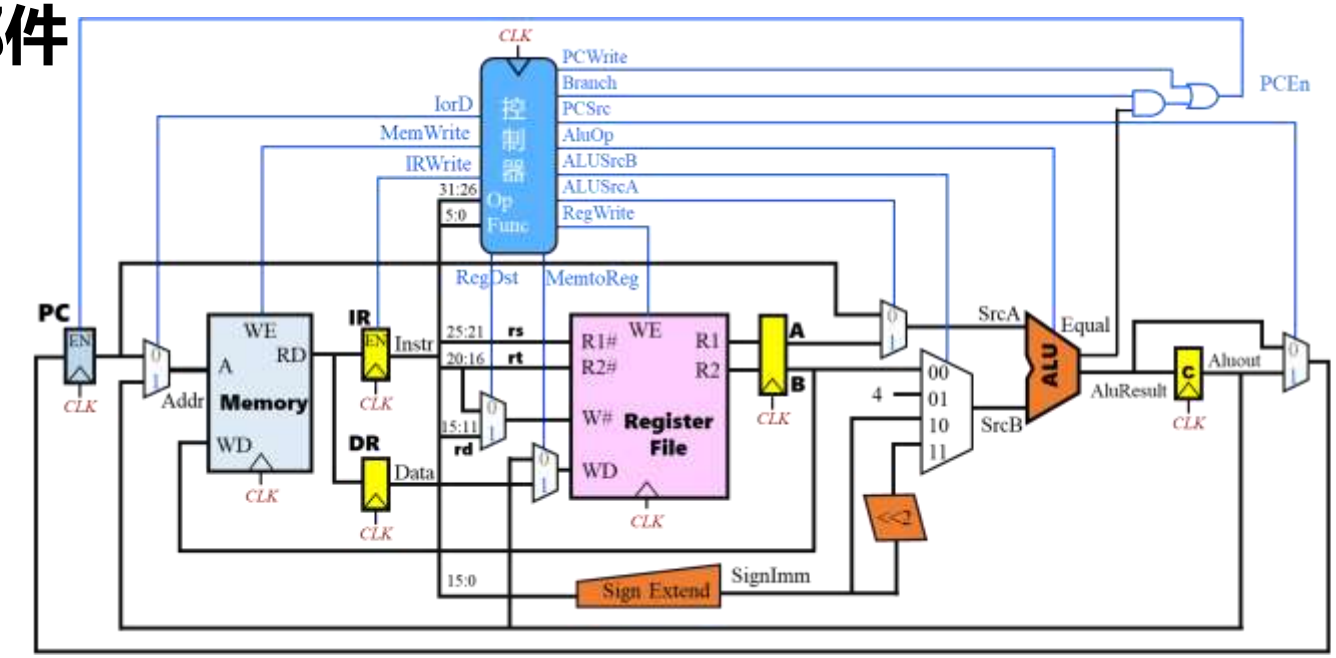
# 单周期时空图



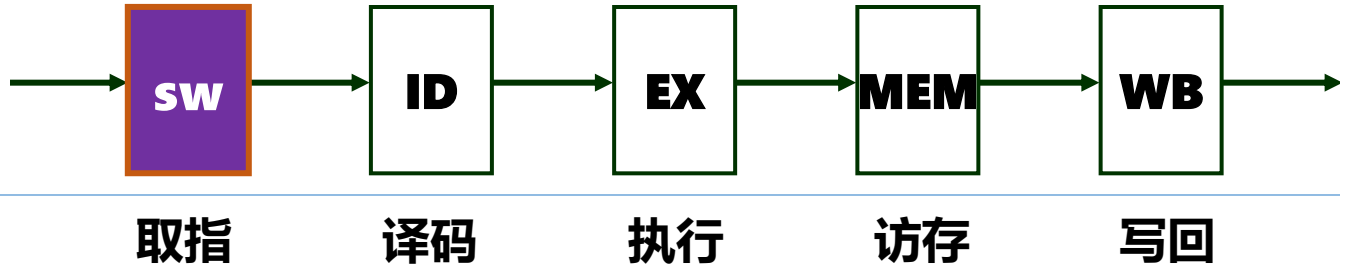

完成n条指令需要 5nT

# 多周期指令运行动态

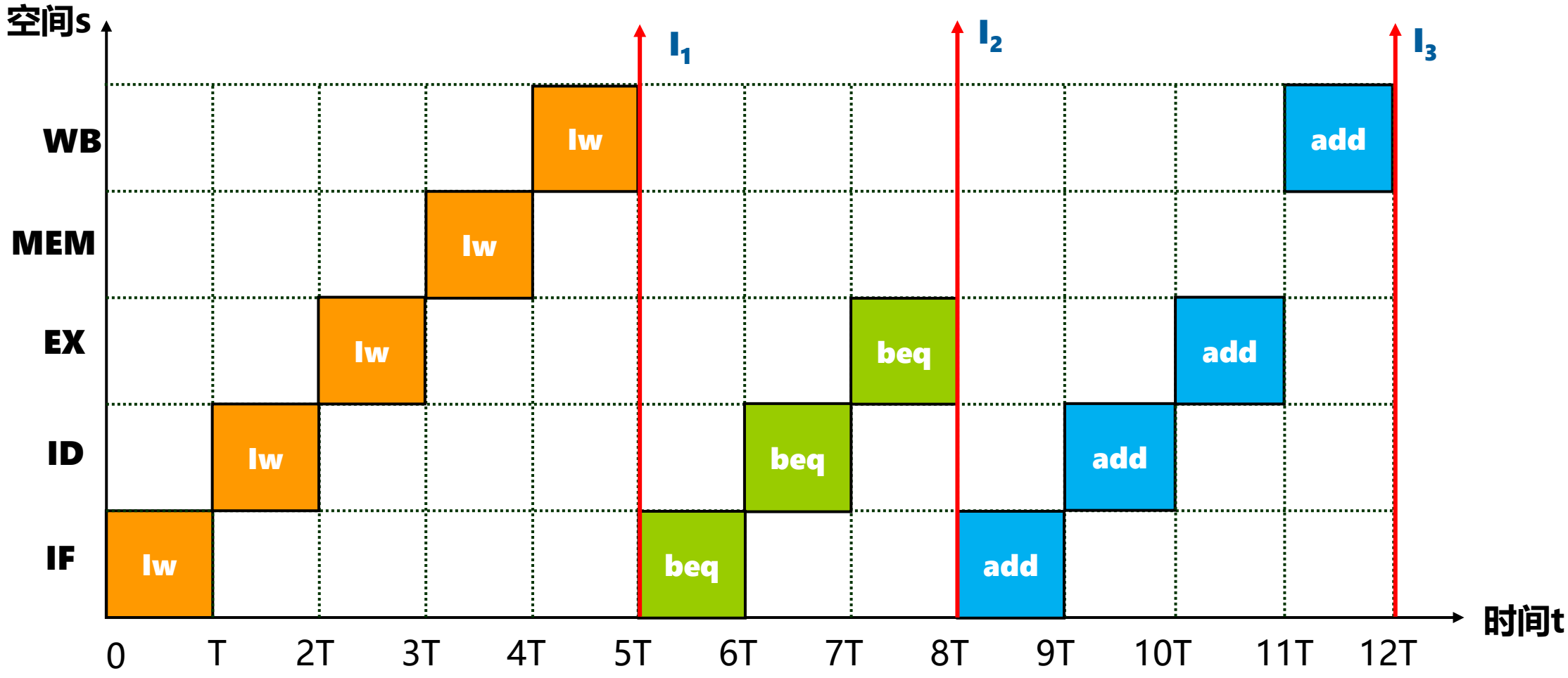
- **数据通路细分为5段，可复用功能部件**




- LW指令　　5个时钟周期
- BEQ指令　　3个时钟周期
- ADD指令　　4个时钟周期
- J指令　　3个时钟周期



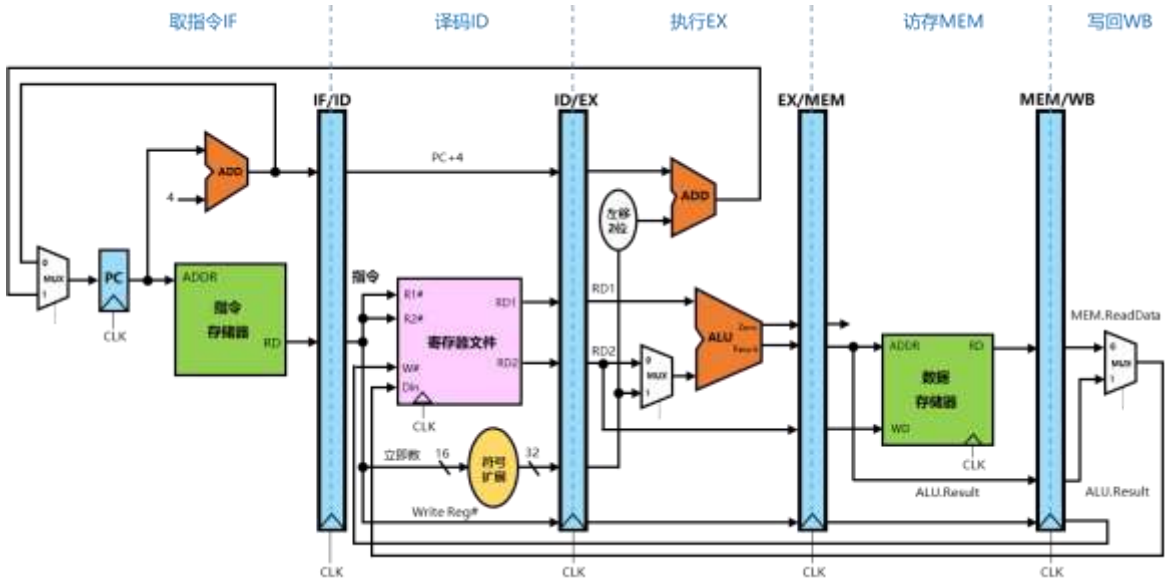
# 多周期时空图

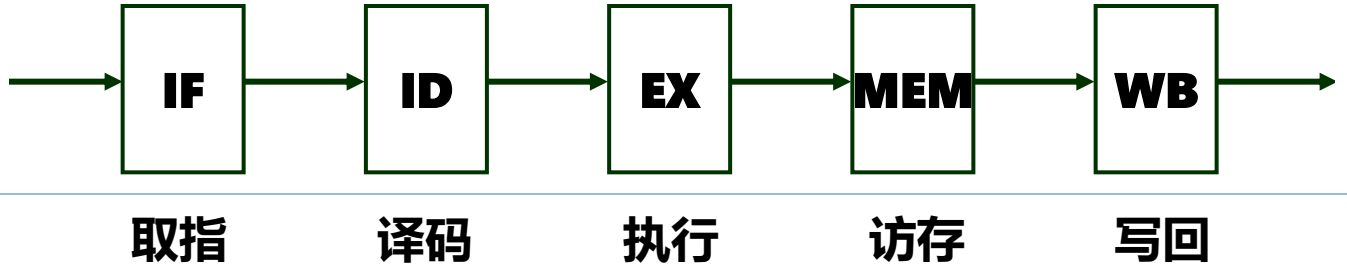



完成n条指令时间与指令有关

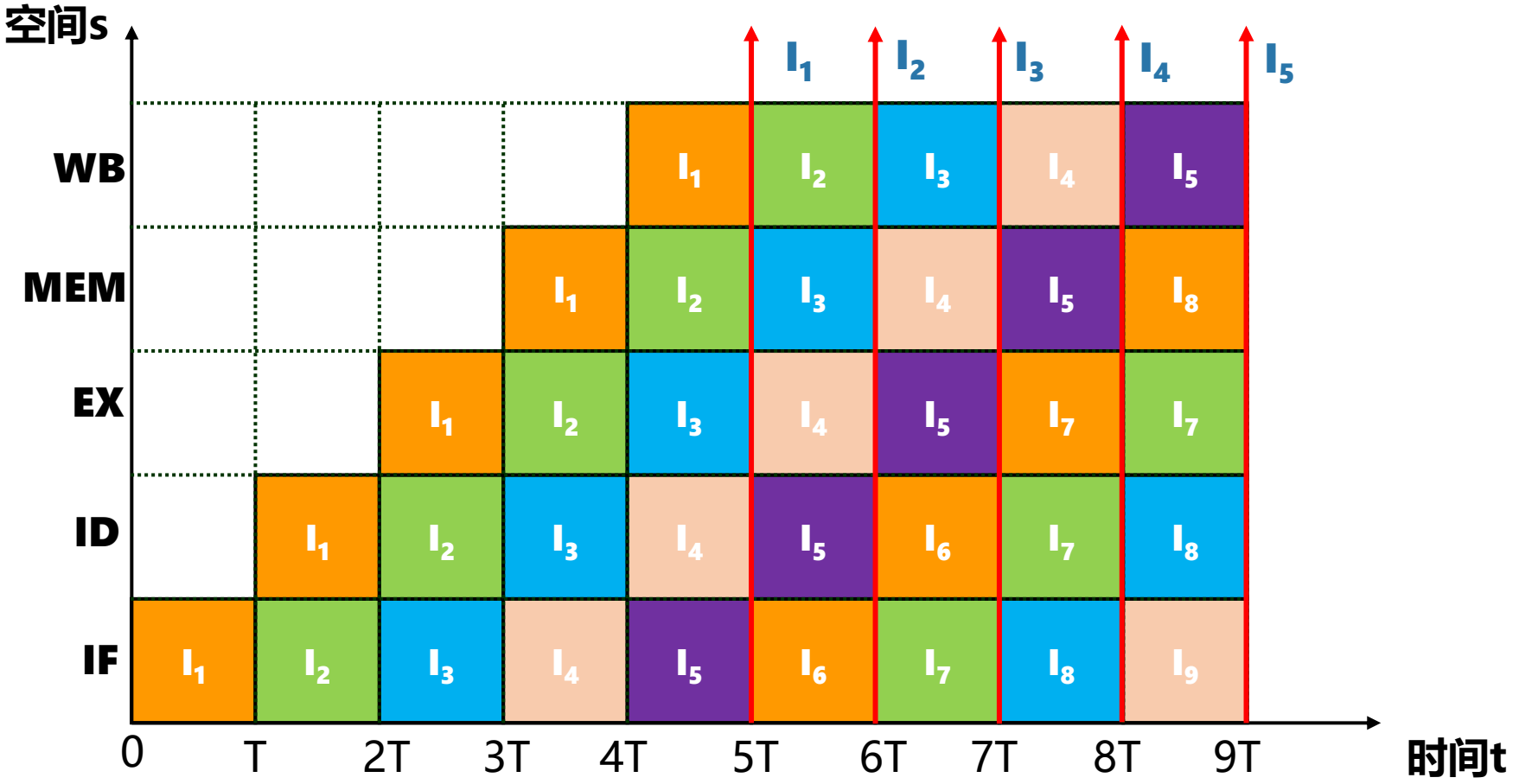
# 流水线指令运行动态

- **数据通路细分为5段，<u>各段完全并发</u>**
  - 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**

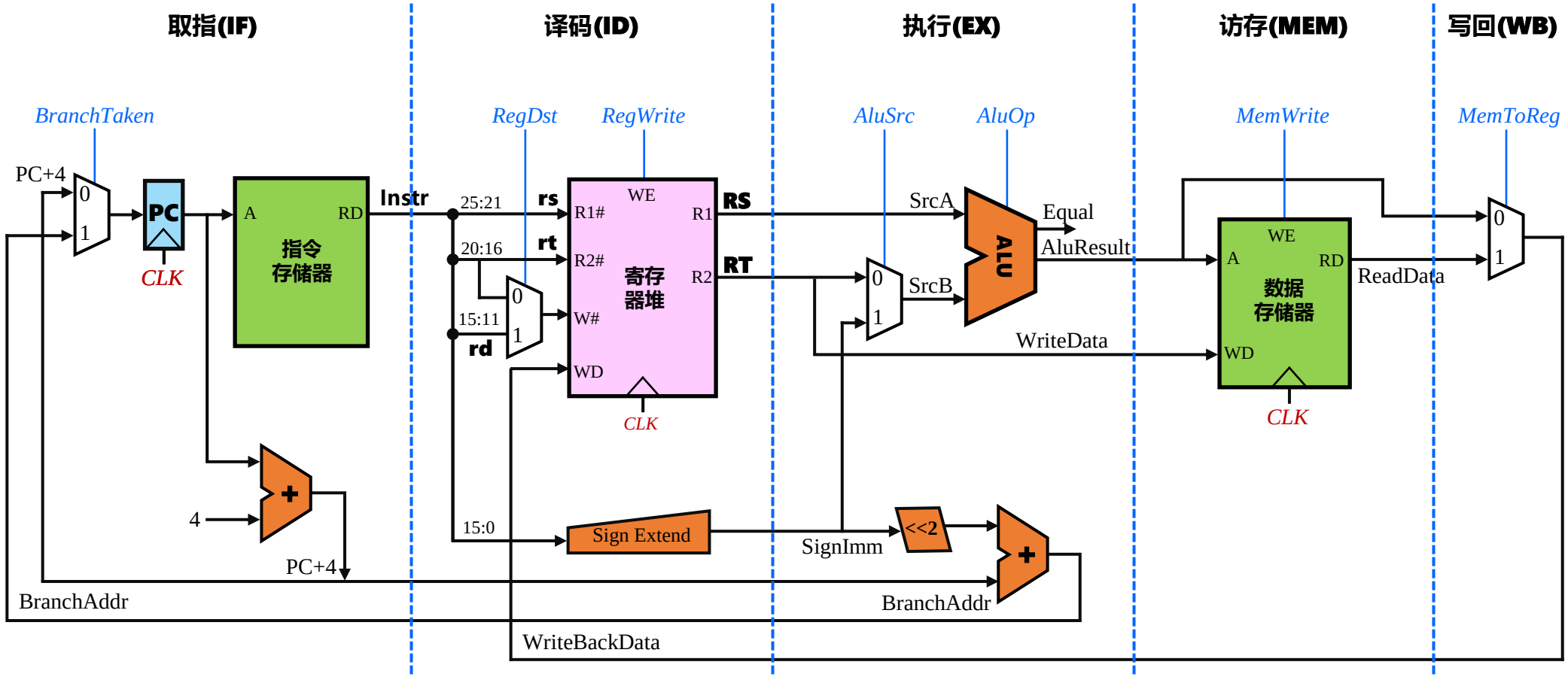
# 理想流水线

- **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

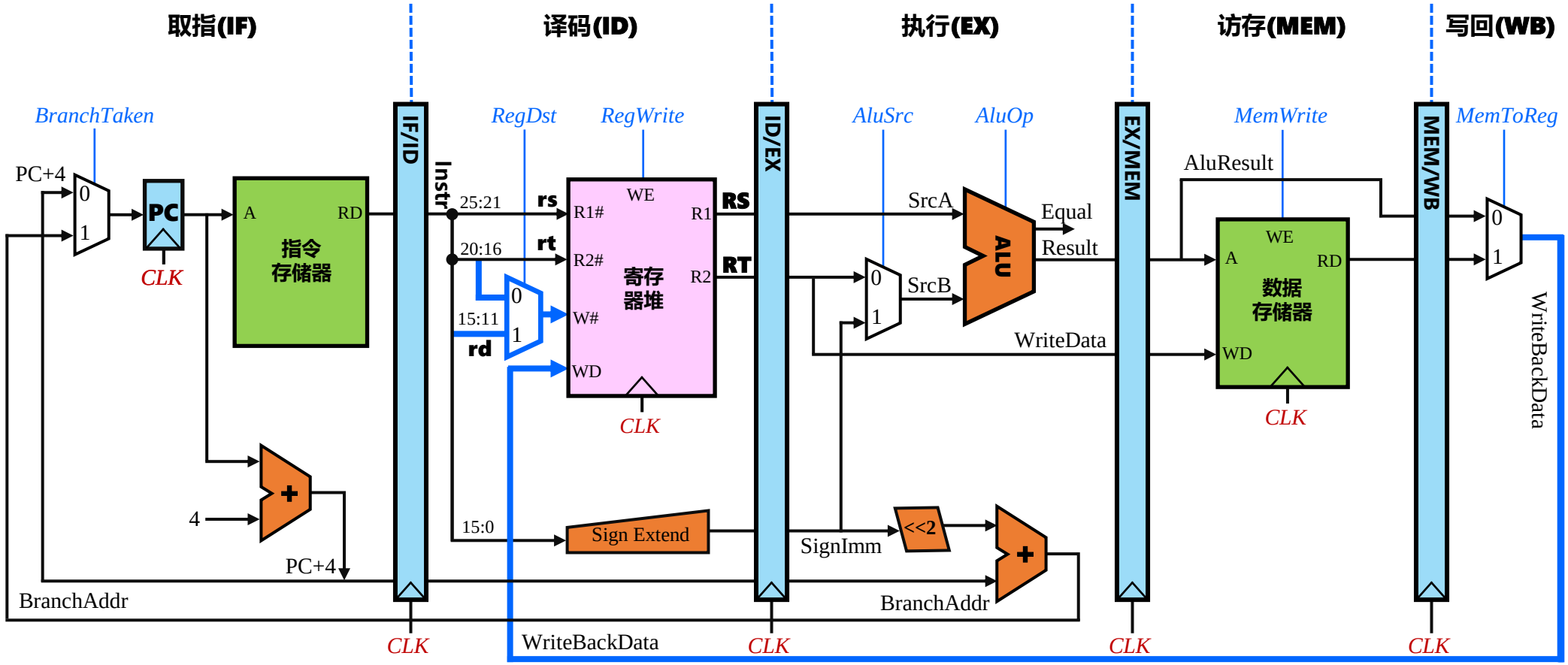
- 7.1 流水线概述
- 7.2 流水线数据通路
- 7.3 流水线冲突与处理
- 7.4 流水线的异常与中断
- 7.5 指令集并行技术

# 单周期MIPS处理器数据通路



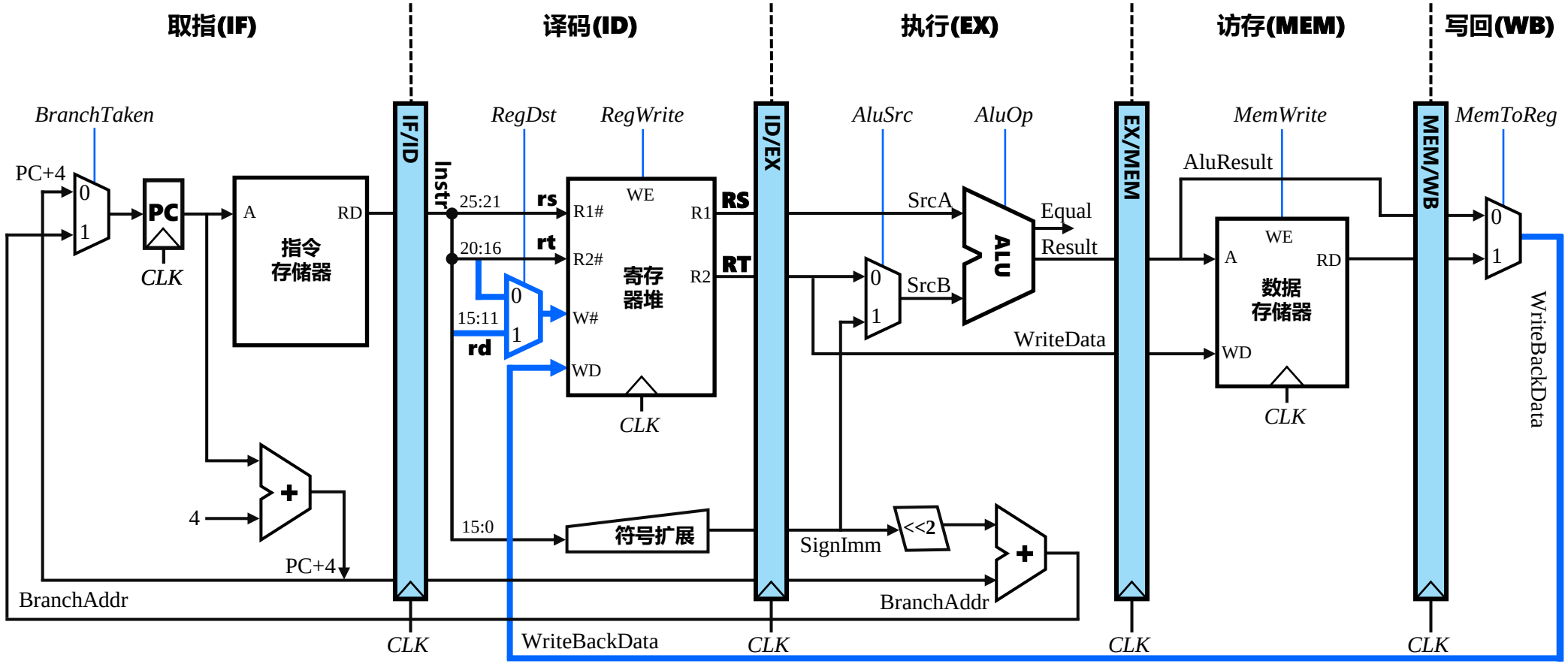

- IF段：程序计数器 PC、指令存储器以及计算下条指令的地址逻辑
- ID段：操作控制器、取操作数逻辑、立即数符号扩展模块
- EX段：算术逻辑运算单元ALU、分支地址计算模块
- MEM段：数据存储器读写模块
- WB段：寄存器写入控制模块。

# 单周期MIPS数据通路流水线改造

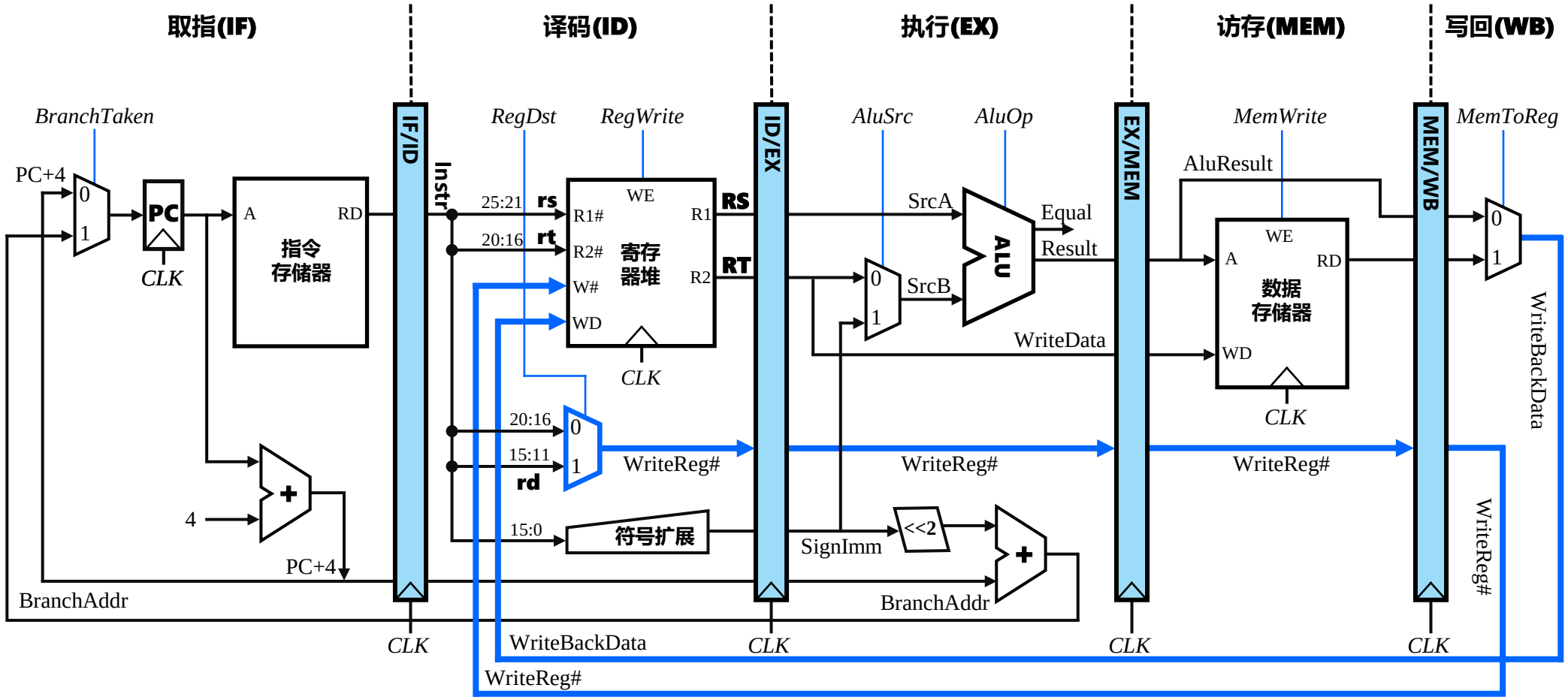



- 在图中虚线位置加入长条形的流水寄存器部件，总共**增加了4个流水寄存器**，根据其所连接的功能段的名称将它们分别命名为**IF/ID**、**ID/EX**、**EX/MEM**、**MEM/WB**
- 流水寄存器用于**锁存前段加工处理完毕的数据和控制信号**，通常这些**数据和控制信号**都会横穿流水寄存器传递到下一段
- 所有流水寄存器、程序计数器PC、寄存器堆、数据存储器均采用统一时钟CLK进行同步，每来一个时钟，就会有一条新的指令进入流水线取指令IF段;同时流水寄存器会锁存前段加工处理完成的数据和控制信号，为下一段的功能部件提供数据输入

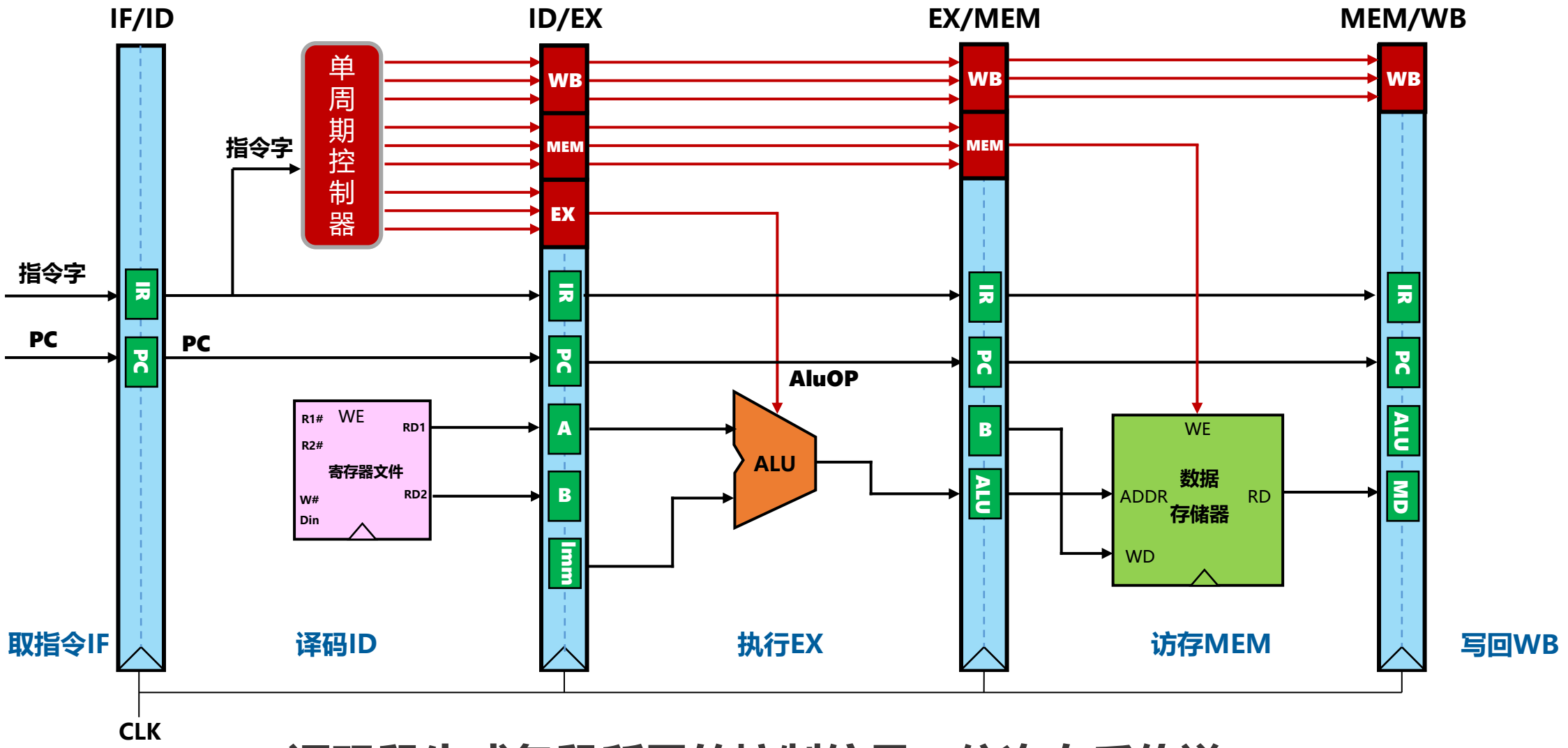
# 单周期MIPS数据通路流水线改造



# 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

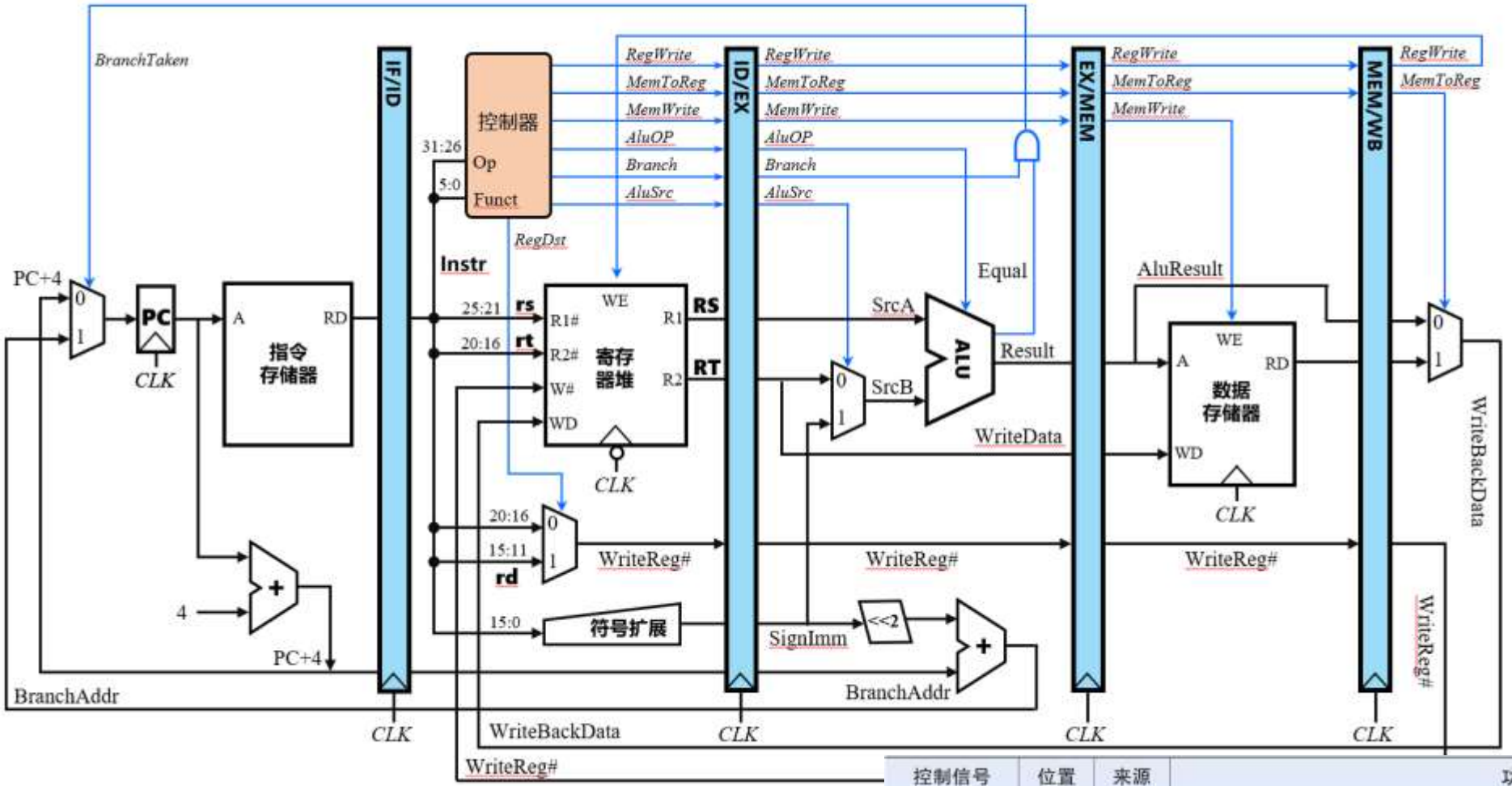



- 译码段生成各段所需的控制信号，依次向后传递

# 控制信号分类

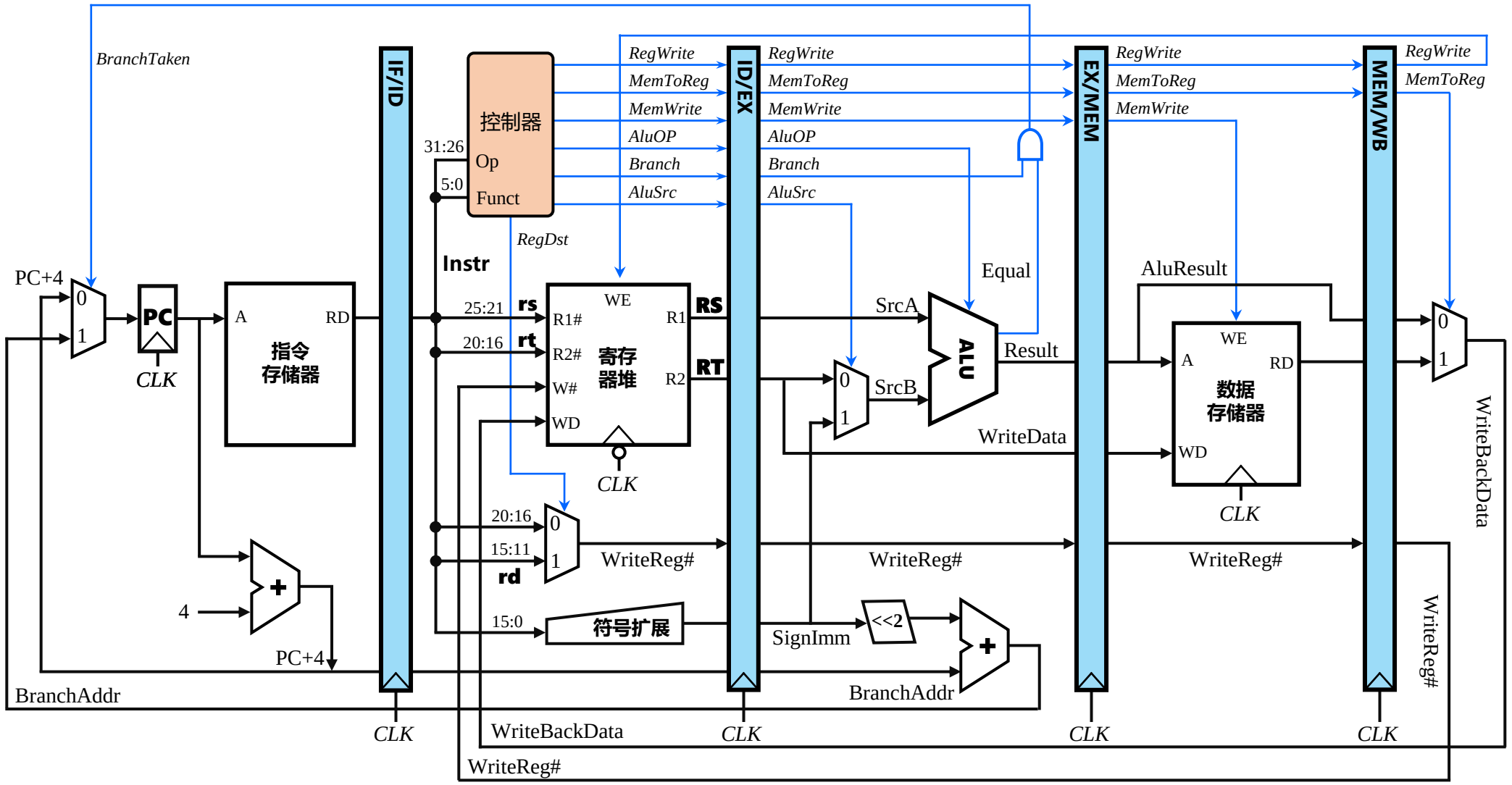
| 控制信号 | 位置 | 来源 | 功能说明 |
| --- | --- | --- | --- |
| **BranchTaken** | **IF** | **EX** | 分支跳转信号，为 1 表示跳转，由 EX 段的 Branch 信号与 equal 标志进行逻辑与生成 |
| **RegDst** | ID | ID | 写入目的寄存器选择，为 1 时目的寄存器为 rd 寄存器，为 0 时为 rt 寄存器 |
| **RegWrite** | **ID** | **WB** | 控制寄存器堆写操作，为 1 时数据需要写回寄存器堆中的指定寄存器 |
| **AluSrc** | EX | 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

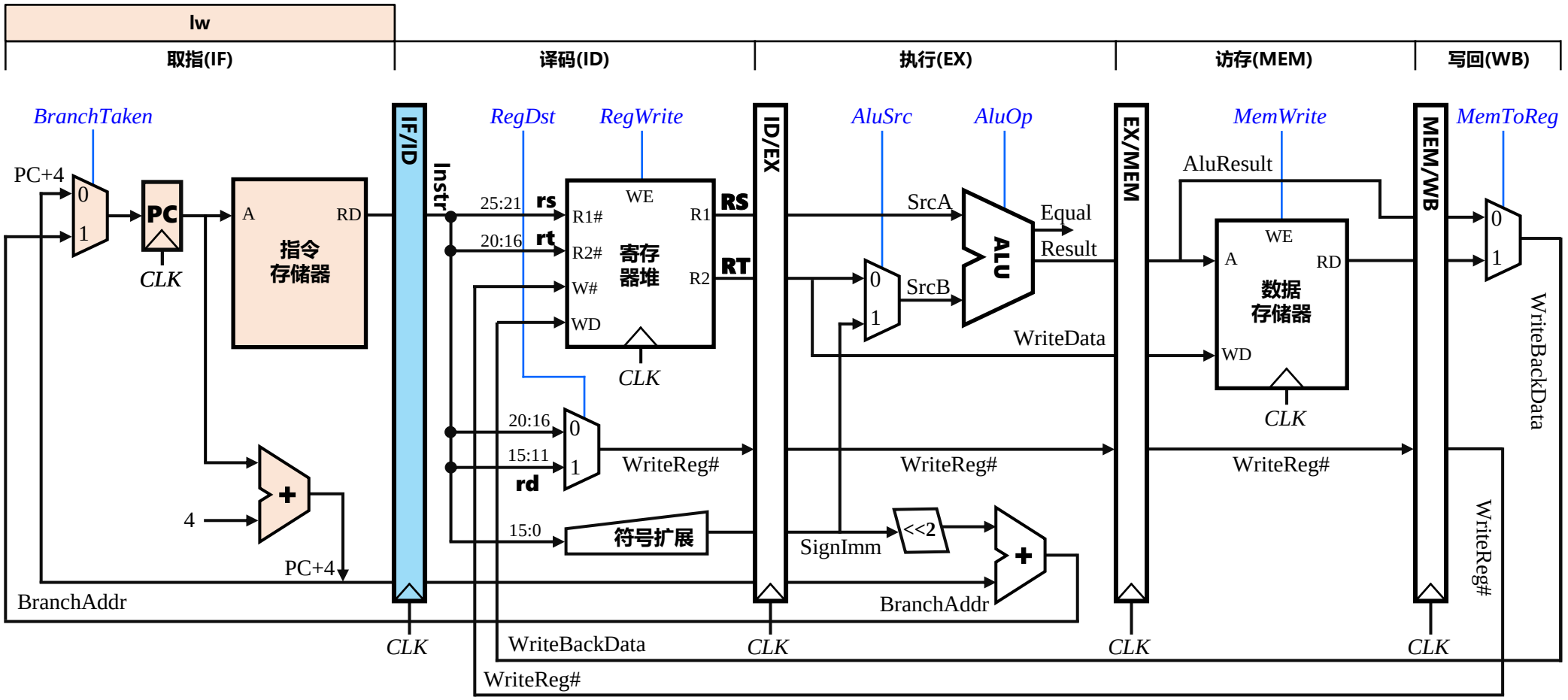



| 控制信号 | 位置 | 来源 | 功能说明 |
|---|---|---|---|
| BranchTaken | IF | EX | 分支跳转信号，为 1 表示跳转，由 EX 段的 Branch 信号与 equal 标志进行逻辑与生成 |
| RegDst | ID | ID | 写入目的寄存器选择，为 1 时目的寄存器为 rd 寄存器，为 0 时为 rt 寄存器 |
| RegWrite | ID | WB | 控制寄存器堆写操作，为 1 时数据需要写回寄存器堆中的指定寄存器 |
| AluSrc | EX | EX | ALU 的第二输入选择控制，为 0 时输入寄存器 rt，为 1 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 |

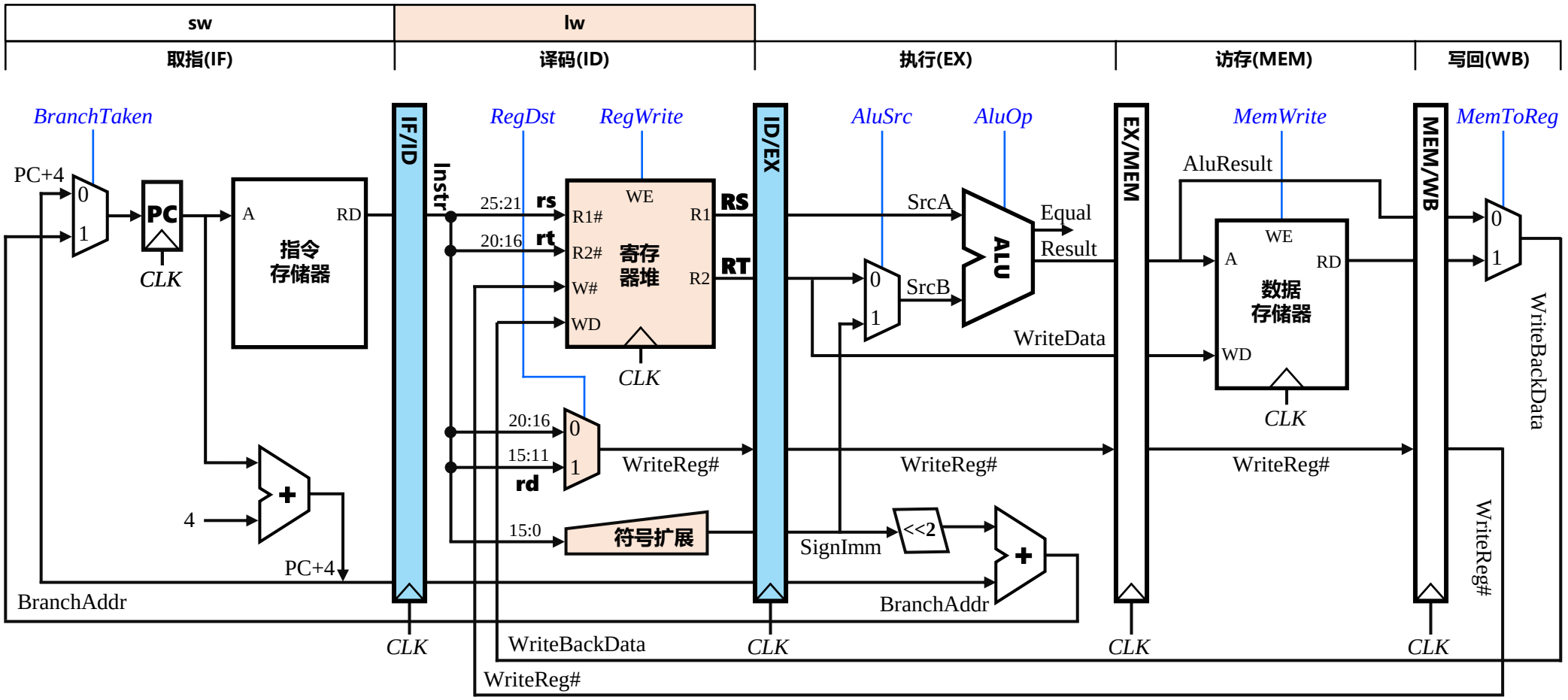
# 5段流水线控制信号与传递



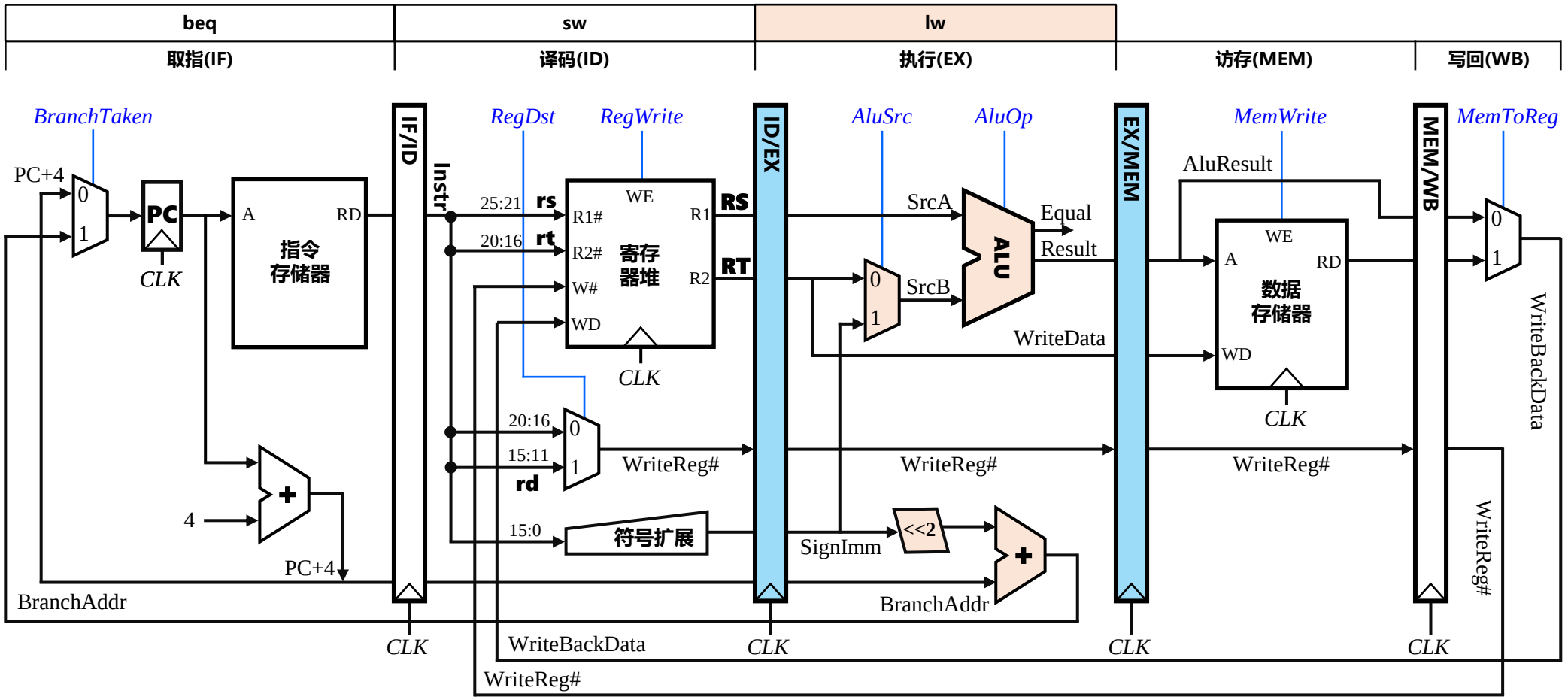
# 7.2.3 指令在流水线中的执行过程



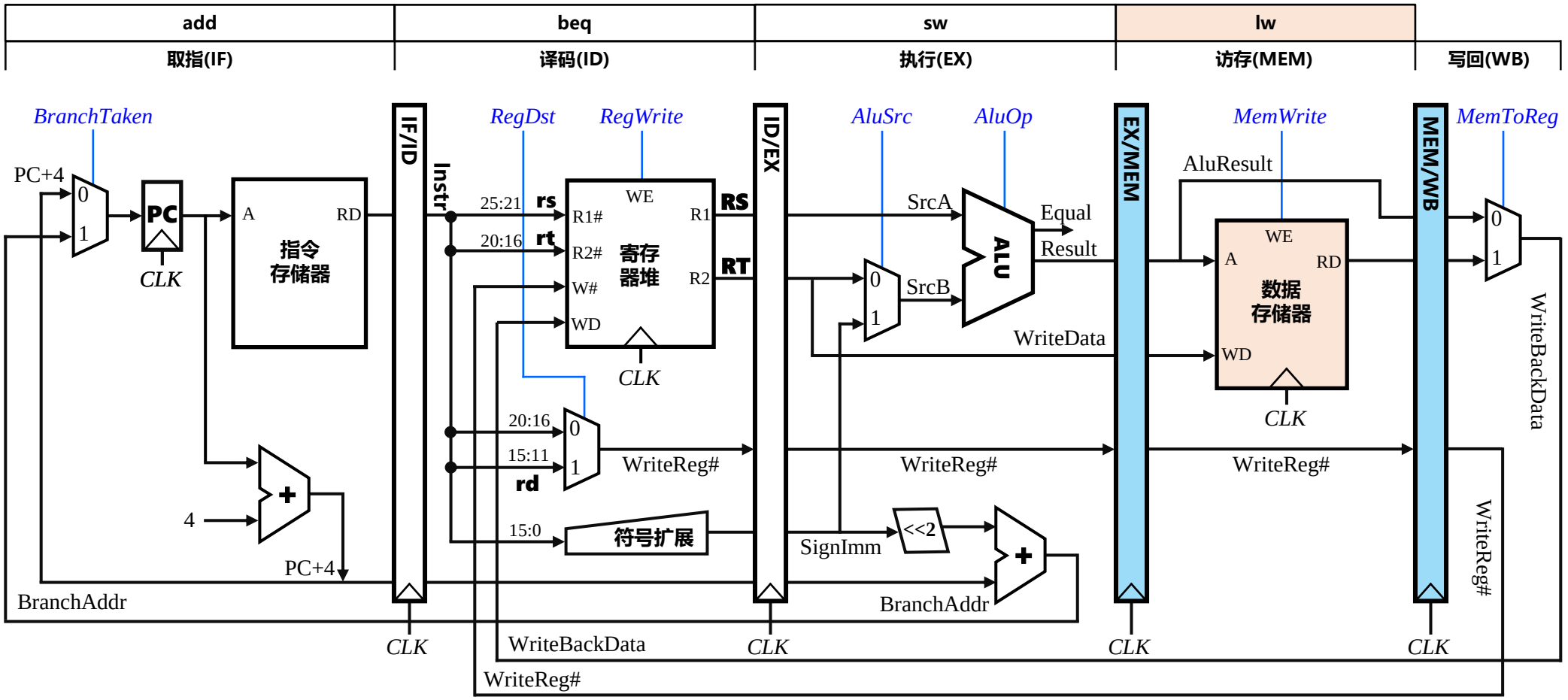
# 7.2.3 指令在流水线中的执行过程



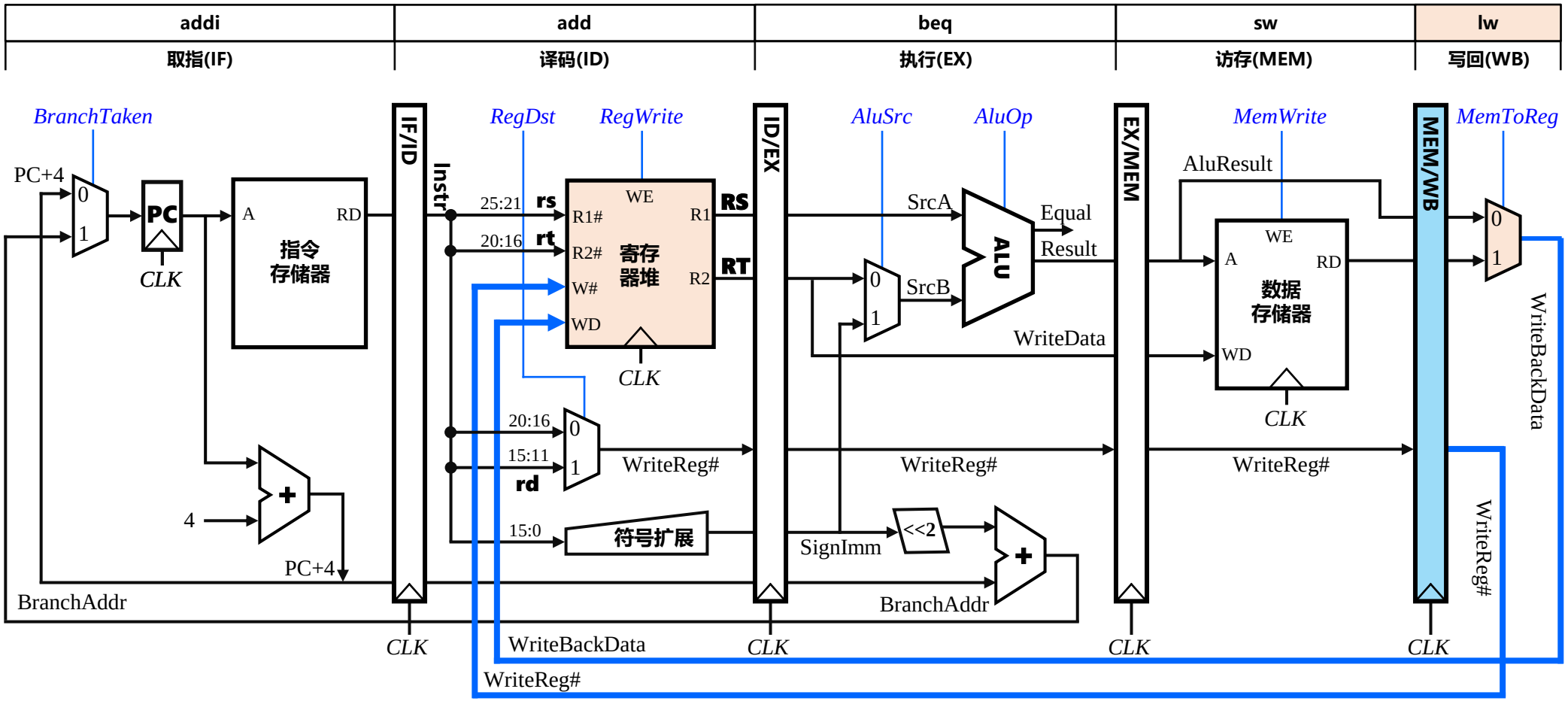
# 7.2.3 指令在流水线中的执行过程



# 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



习题：若某计算机最复杂指令的执行需要完成5个子功能，分别由功能部件A~E实现，各功能部件所需时间分别为 80ps、50ps、50ps、70ps和50ps，采用流水线方式执行指令，流水段寄存器延迟时间为20ps，则CPU 时钟周期至少为多少?

习题：若某计算机最复杂指令的执行需要完成5个子功能，分别由功能部件A~E实现，各功能部件所需时间分别为 80ps、50ps、50ps、70ps和50ps，采用流水线方式执行指令，流水段寄存器延迟时间为20ps，则CPU 时钟周期至少为多少?

CPU时钟周期=各功能段的最长执行时间 + 流水段寄存器时时长= 100ps

# 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）

- 指令相关：两条指令间存在某种依赖关系
- 指令相关会导致流水线冲突(冒险)(Hazzard)
  - 数据相关
  - 结构相关
  - 控制相关
- 流水线冲突包括
  - 结构冲突
  - 控制冲突
  - 数据冲突

# 指令流水线的冲突、相关、冒险（hazard）

- **结构冲突**
  - 多条指令在同一时钟周期都需使用同一操作部件引起的冲突称为结构冲突
  - **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**
  - 当流水线遇到分支指令或其他会改变PC值的指令时，在分支指令之后载入流水线的相邻指令可能因为分支跳转而不能进入执行阶段
  - 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**
  - 当前指令要用到先前指令的操作结果，而这个结果尚未产生或尚未送达指定的位置，会导致当前指令无法继续执行
  - 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

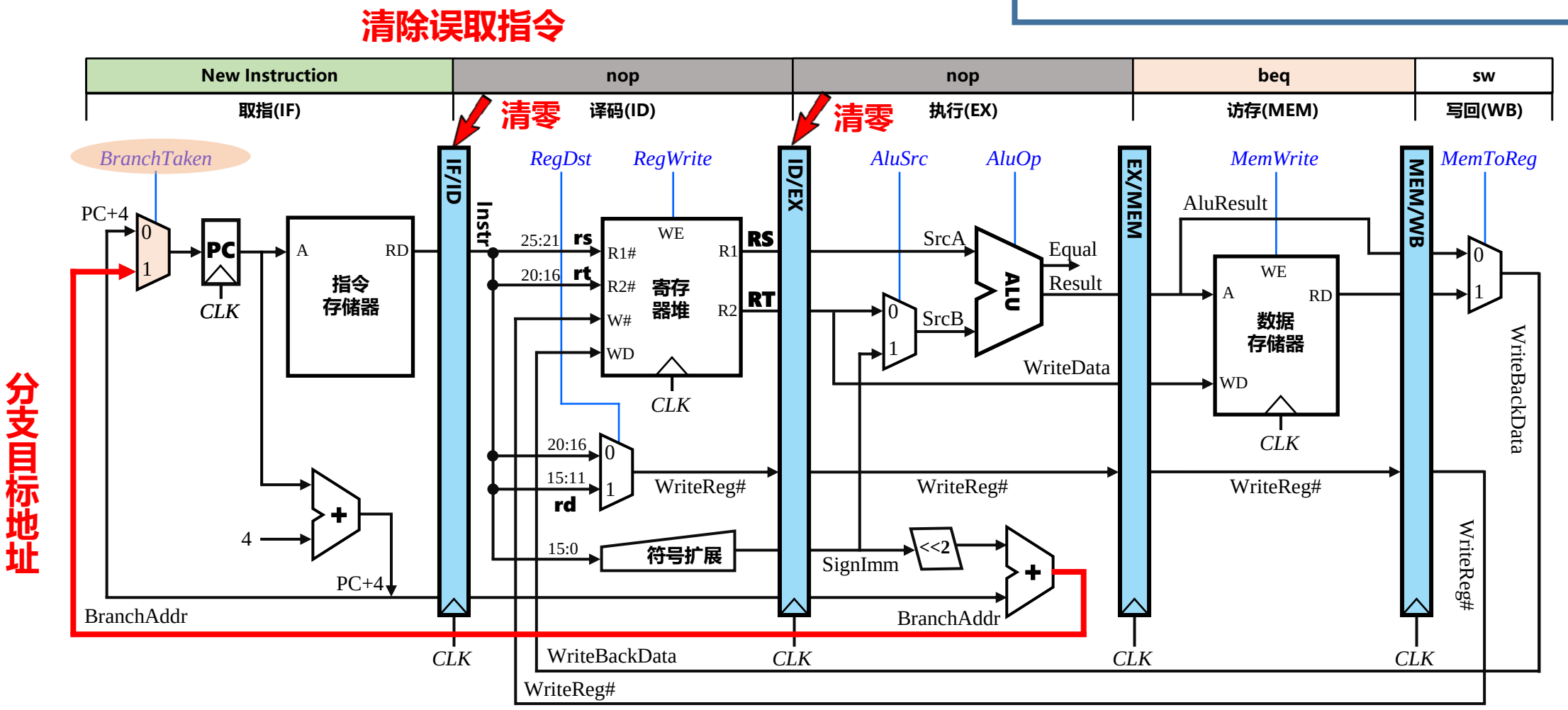
```
ADD $s1, $s2, $s3
ADD $s4, $s1, $s3
```

# 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?**
  - 只需要在实际分支跳转时，将分支指令所在功能段左侧所有即将存放误取指令的流水寄存器同步清零即可
  - 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

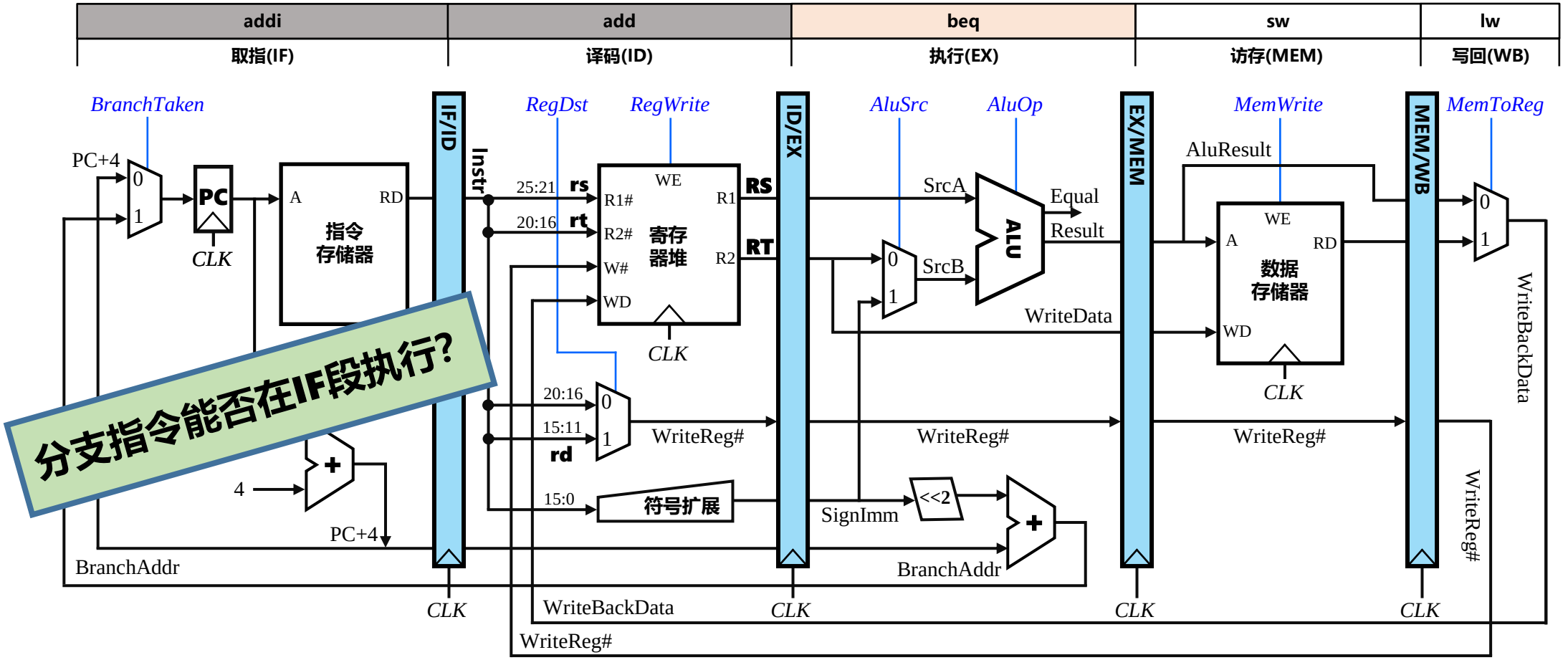
# 分支相关

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**

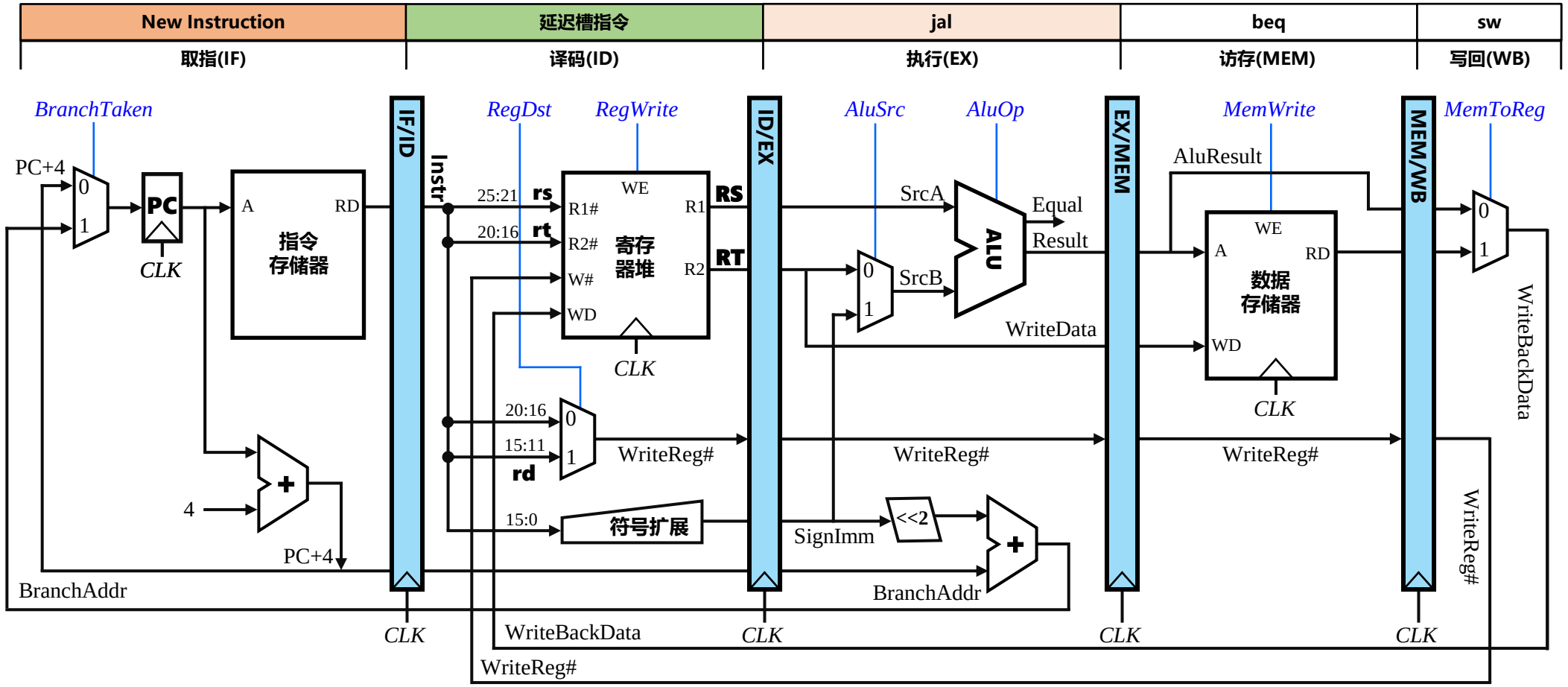
# 分支指令执行时机?




**分支执行越早，性能损失越小！**

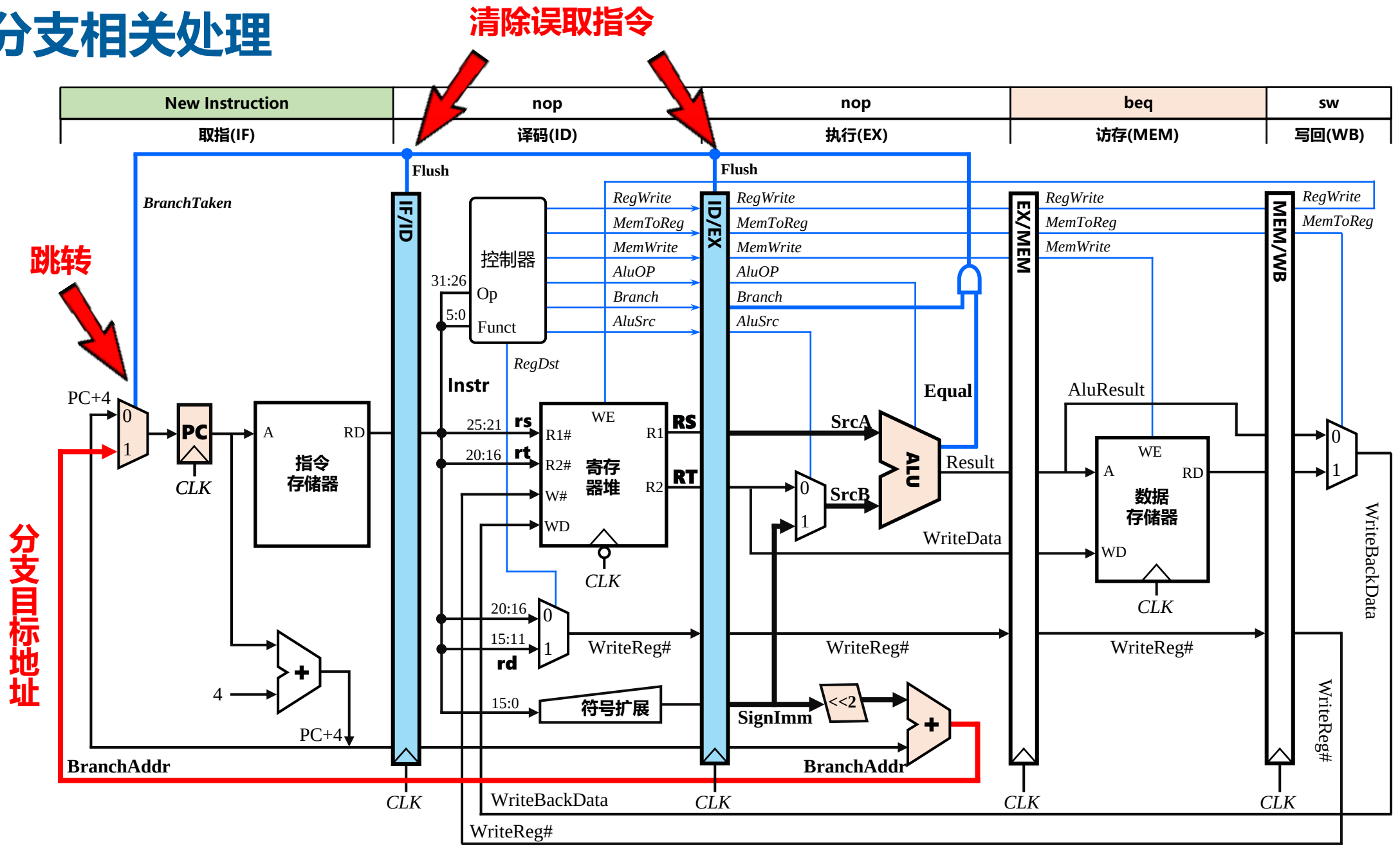
# MIPS延迟槽技术

```
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
Op
Funct
31:26
5:0
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
20:16
15:11
rd
0
1
WriteReg#
15:0
符号扩展
SignImm
<<2
4
PC+4
BranchAddr
WriteBackData
WriteReg#
Equal
SrcA
SrcB
ALU
Result
WriteData
AluResult
数据
存储器
WriteBackData
WriteReg#

# 分支相关执行动态

```
1  beq  $0, $0, 8        # 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, 8        # 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 | | | | | |

# 数据冲突

- **先写后读冲突**
  - 如果指令I2的源操作数是指令I1的目的操作数
  - 由于指令I2要用到指令I1的结果，如果指令I2在指令I1将结果写寄存器之前就在ID段读取了该寄存器的旧值，则会导致读取数据出错
- **先读后写冲突**
  - 如果指令I2的目的操作数是指令I1的源操作数
  - 当指令I2去写该寄存器的时候，指令I1已经读取过该寄存器了
- **写后写冲突**
  - 如果指令I2和指令I1的目的操作数是相同的

# 数据相关处理机制

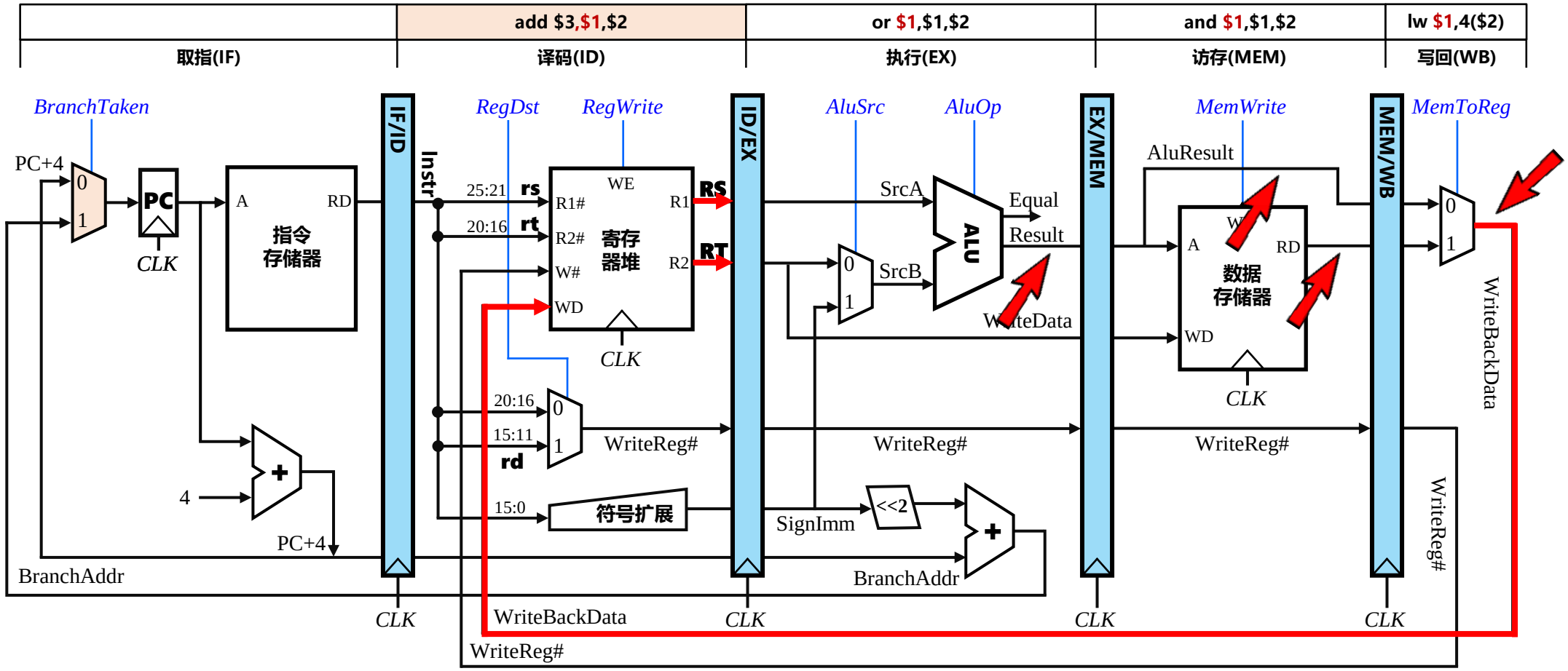
- **软件方法（编译器完成）**
  - 插入空指令
  - 调整程序顺序，使相关性在流水线中消失
- **硬件方法**
  - **插入气泡（空操作）**
    - 向后段插入气泡（接口信号清零）
    - 向前给出阻塞信号（流水线停顿）避免当前指令被新指令取代
  - **数据重定向bypass（数据旁路）**
    - 将后端处理后的数据（还没来得及写回）重定向
    - 数据在哪就从哪送到运算器

习题：在无转发机制的5段基本流水线中，下列指令序列存在数据冲突的指令对是
I1:ADD R1, R2,R3: (R2)+(R3)→R1
I2:ADD R5,R2, R4; (R2)+(R4) →R5
I3:ADDR4, R5, R3; (R5)+(R3)→R4
I4:ADD R5, R2, R6; (R2)+(R6)→R5

习题：在无转发机制的5段基本流水线中，下列指令序列存在数据冲突的指令对是
I1:ADD R1, R2,R3: (R2)+(R3)→R1
I2:ADD R5,R2, R4; (R2)+(R4) →R5
I3:ADDR4, R5, R3; (R5)+(R3)→R4
I4:ADD R5, R2, R6; (R2)+(R6)→R5

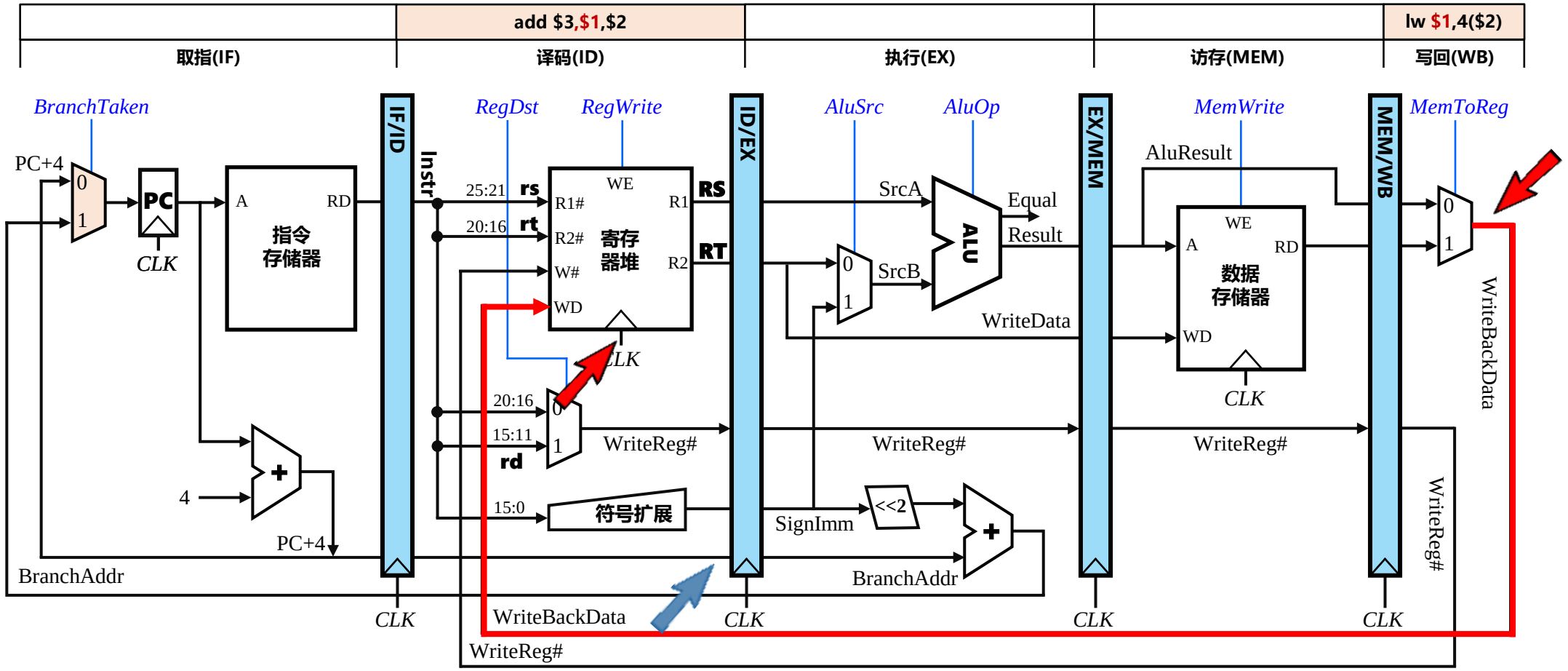
I2和I3.
指令I3 的源寄存器R5需要等待12指令中结果写回后才能取
正确的寄存器值

# 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**

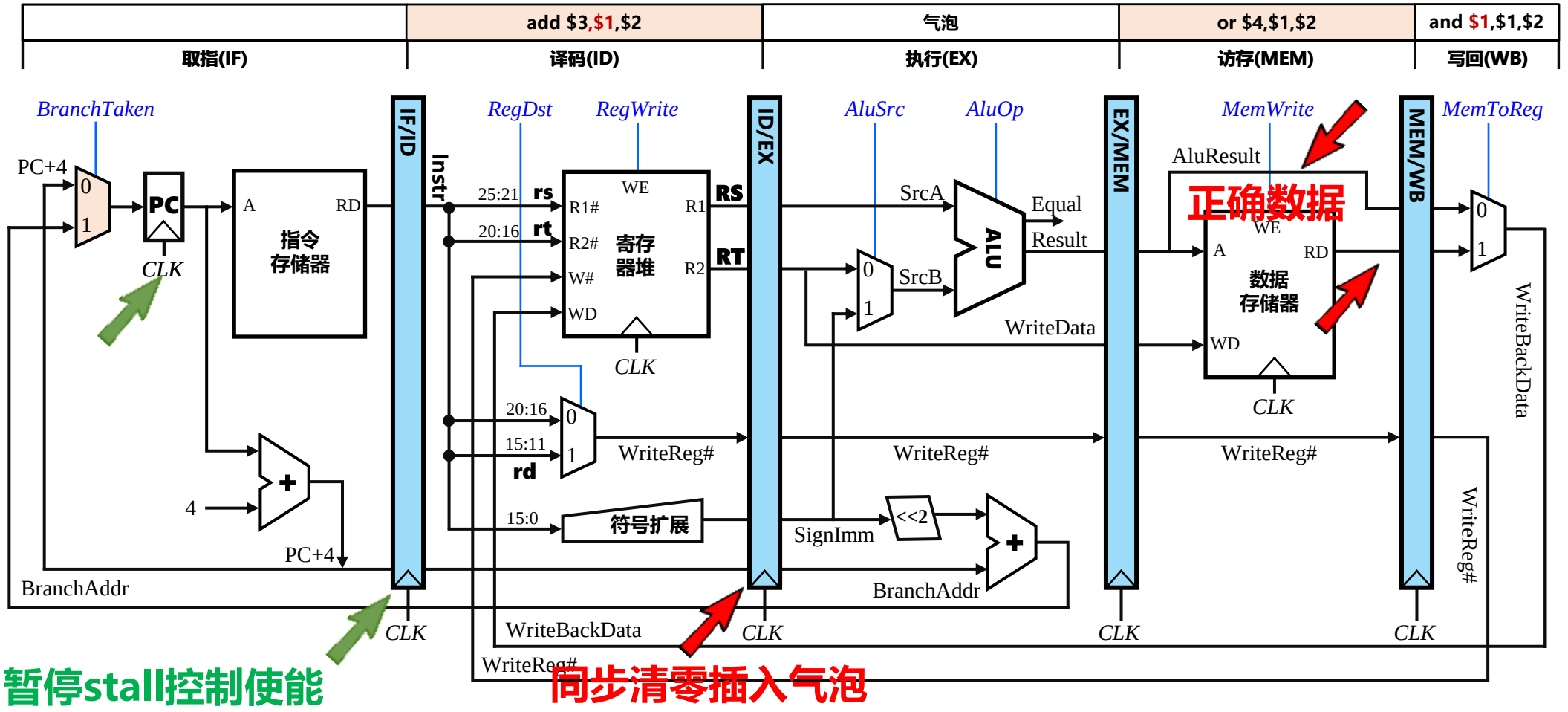
# ID段与WB段数据相关消除




**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

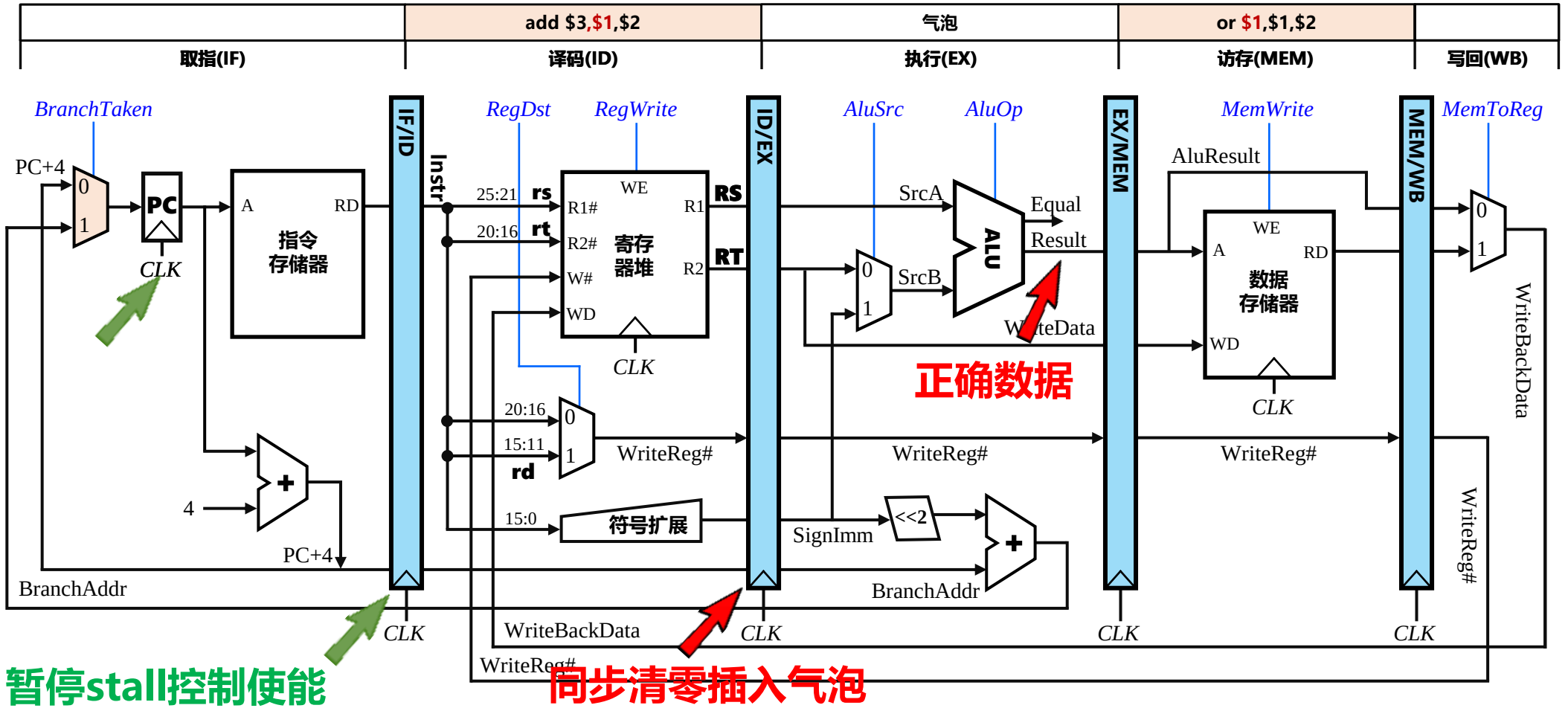
**下一个时钟相关性?**




**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*

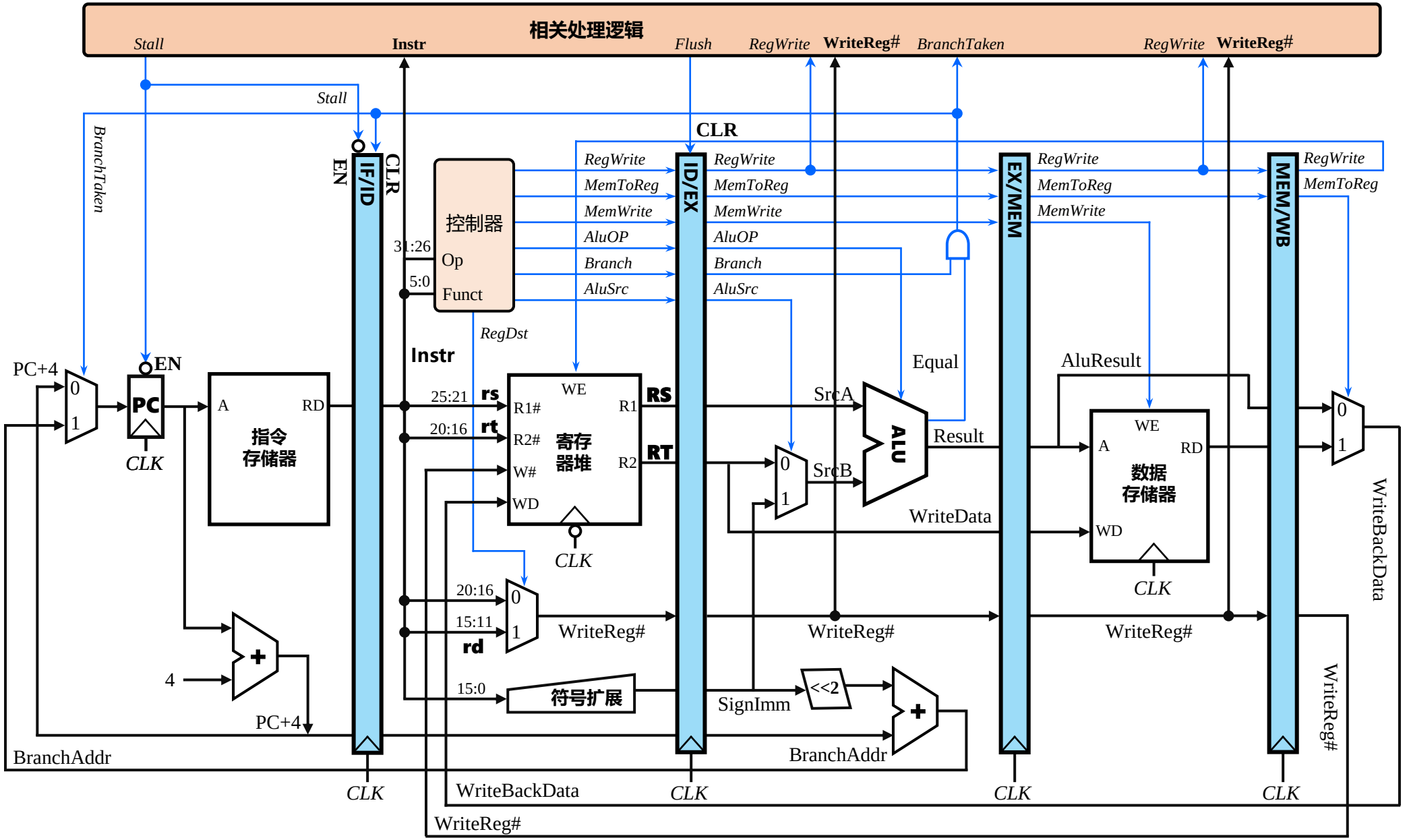
*PC.EN = ~Stall*

*IF/ID.EN = ~Stall*

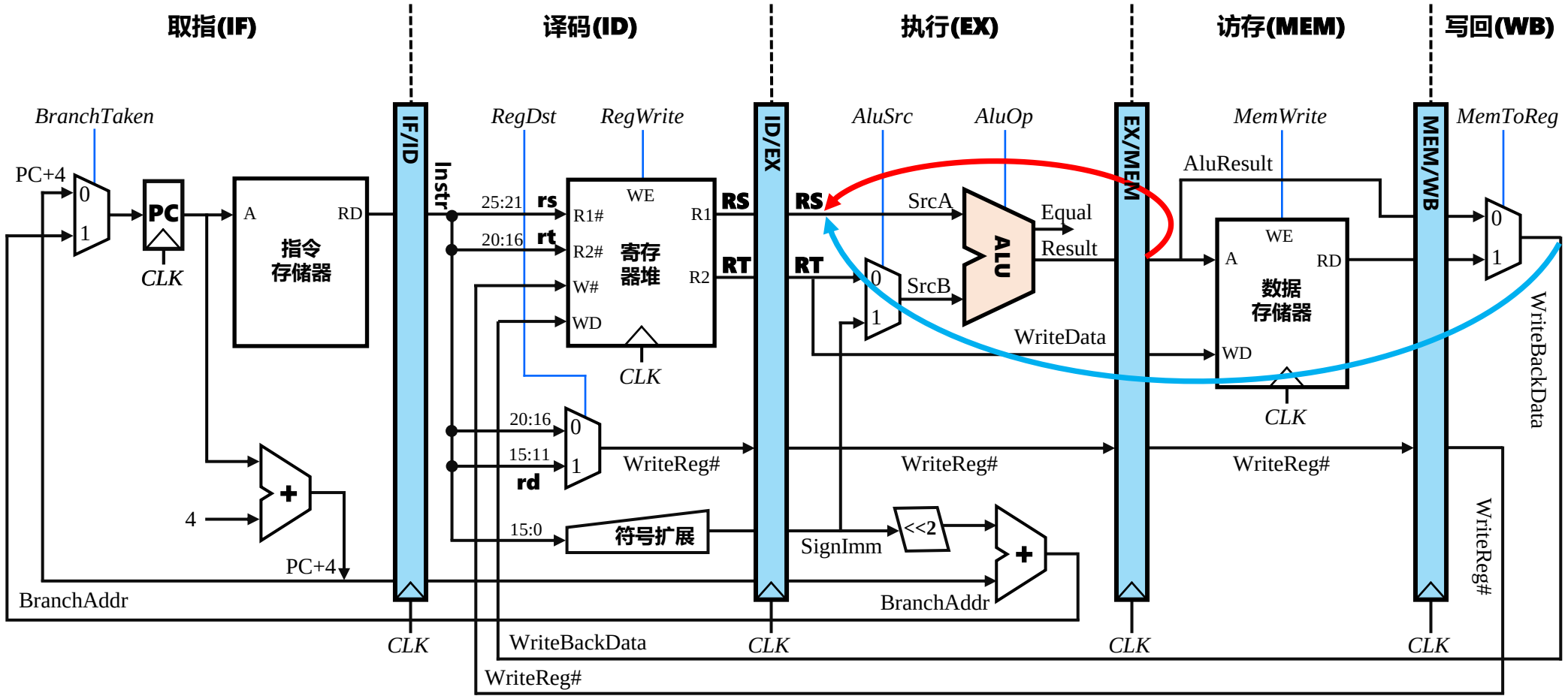
*IF/ID.CLR = BranchTaken*

*ID/EX.CLR = Flush = BranchTaken + DataHazzard*

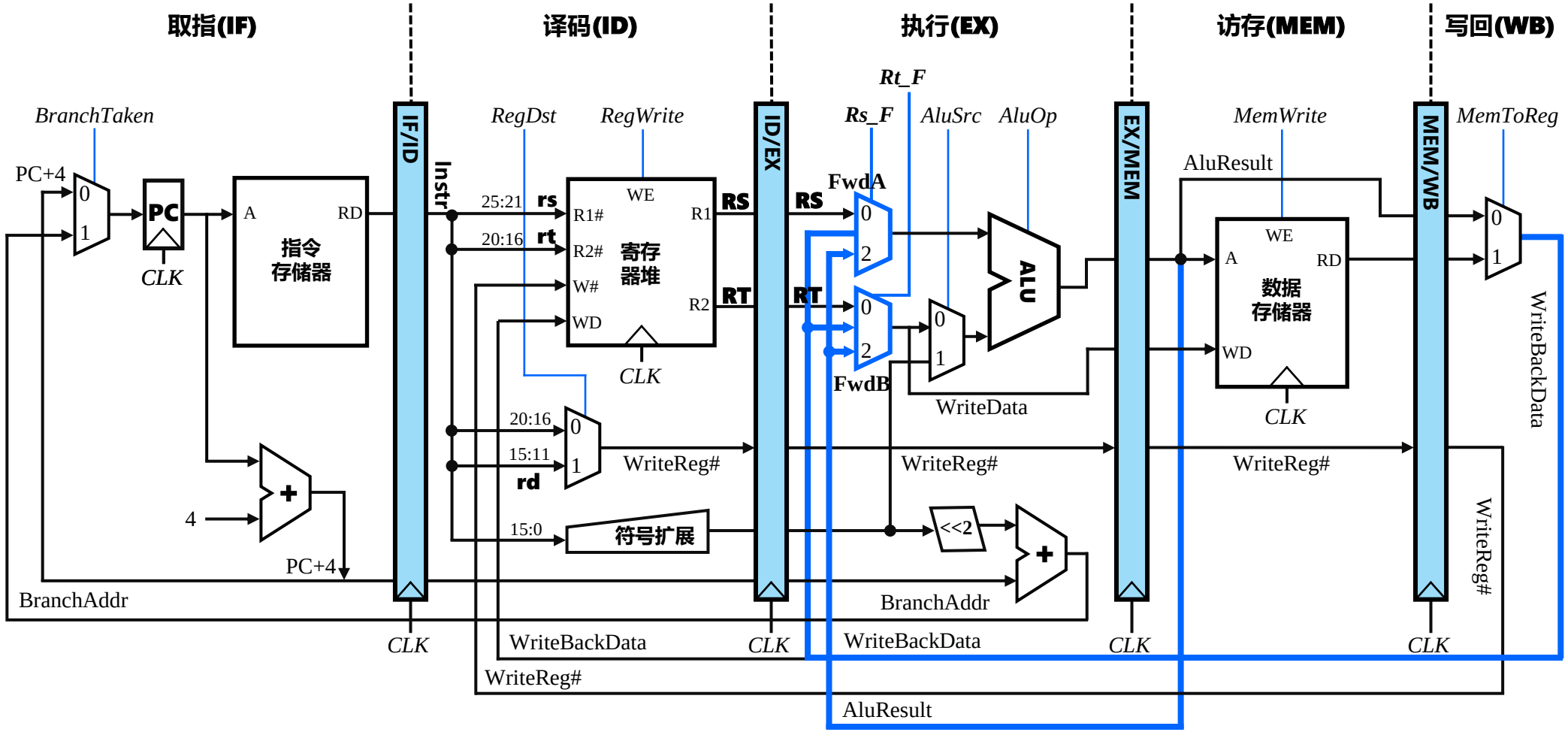
# 气泡流水线顶层视图


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
寄存
器堆
R1
R2
RS
RT
CLK
Equal
SrcA
ALU
Result
0
1
SrcB
WriteData
AluResult
WE
A
RD
数据
存储器
WD
CLK
0
1
WriteBackData
20:16
15:11
rd
0
1
WriteReg#
WriteReg#
WriteReg#
4
+
PC+4
15:0
符号扩展
SignImm
<<2
+
BranchAddr
BranchAddr
CLK
CLK
CLK
CLK
WriteBackData
WriteReg#

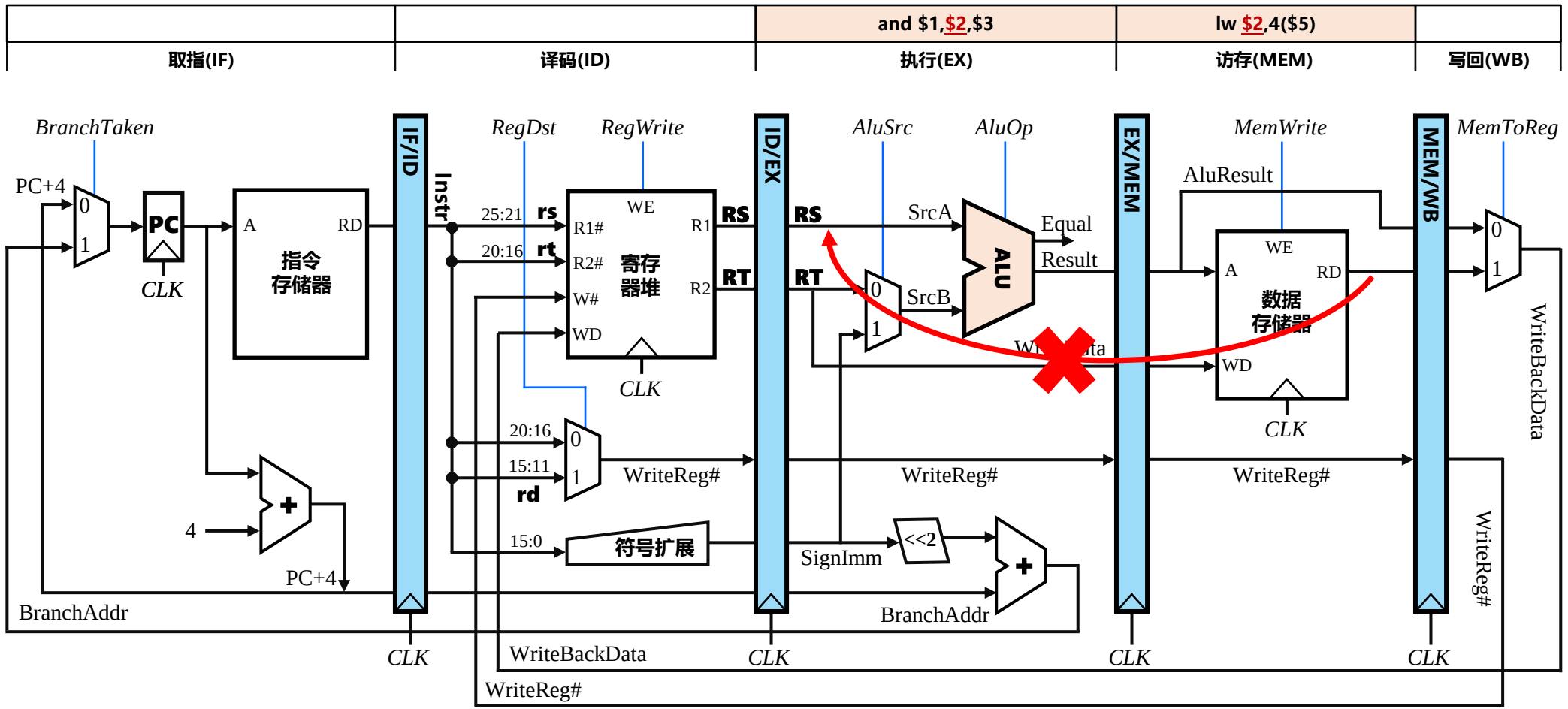
# 数据重定向


取指(IF)
译码(ID)
执行(EX)
访存(MEM)
写回(WB)
BranchTaken
PC+4
0
1
PC
CLK
A
RD
指令
存储器
IF/ID
Instr
RegDst
RegWrite
25:21
rs
20:16
rt
WE
R1#
R2#
W#
WD
寄存
器堆
R1
R2
RS
RT
CLK
ID/EX
AluSrc
AluOp
RS
RT
SrcA
SrcB
ALU
Equal
Result
WriteData
EX/MEM
MemWrite
AluResult
WE
A
RD
数据
存储器
WD
CLK
MEM/WB
MemToReg
WriteBackData
20:16
15:11
rd
WriteReg#
15:0
符号扩展
SignImm
<<2
4
PC+4
BranchAddr
BranchAddr
CLK
WriteBackData
WriteReg#
CLK
CLK
CLK
CLK
WriteReg#

# 数据重定向数据通路

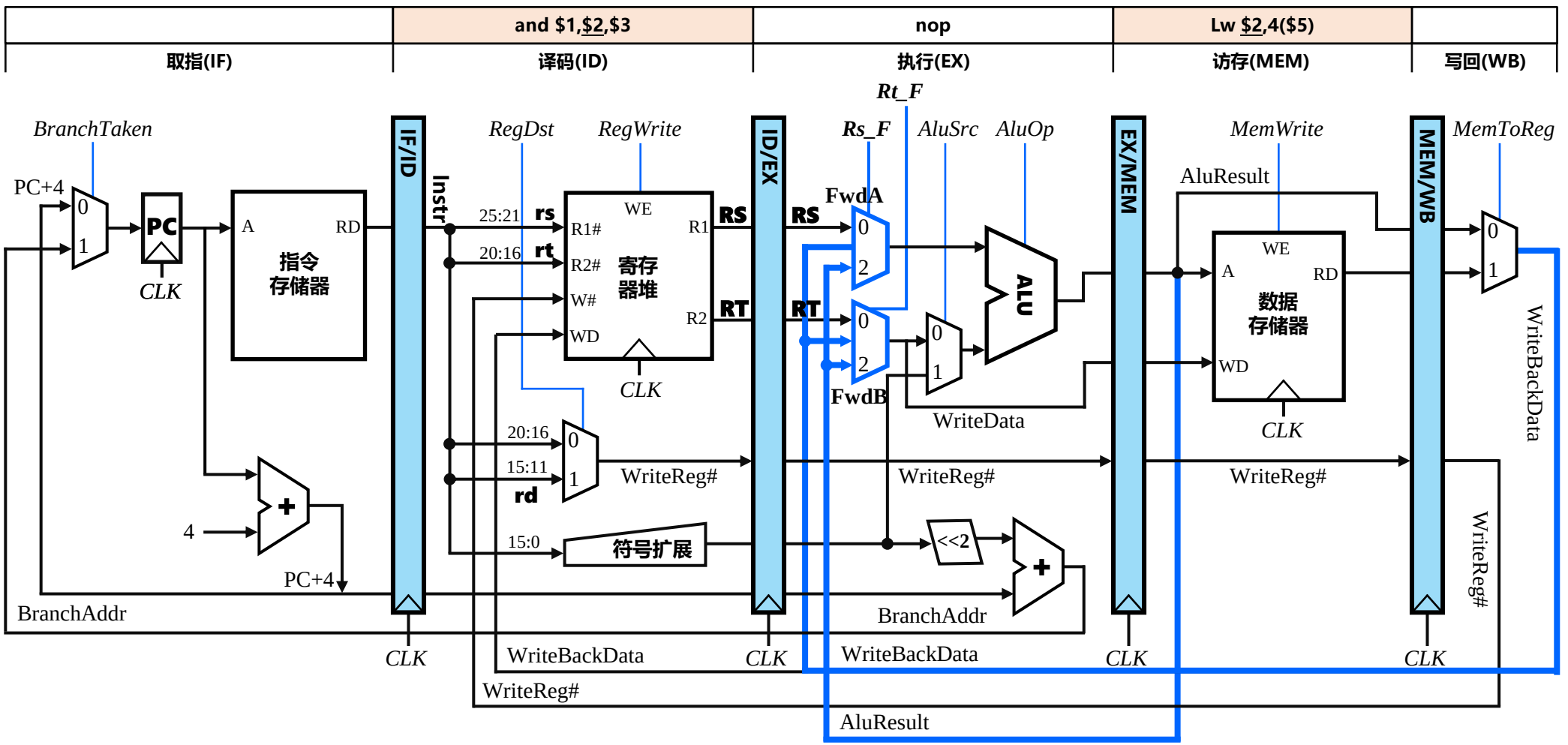

取指(IF)
译码(ID)
执行(EX)
访存(MEM)
写回(WB)
IF/ID
ID/EX
EX/MEM
MEM/WB
BranchTaken
PC+4
PC
CLK
指令
存储器
A
RD
Instr
RegDst
RegWrite
25:21
rs
20:16
rt
寄存
器堆
WE
R1#
R2#
W#
WD
R1
R2
RS
RT
Rt_F
Rs_F
AluSrc
AluOp
FwdA
FwdB
ALU
MemWrite
MemToReg
AluResult
数据
存储器
WriteData
20:16
15:11
rd
WriteReg#
15:0
符号扩展
<<2
4
BranchAddr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

# Load-Use相关

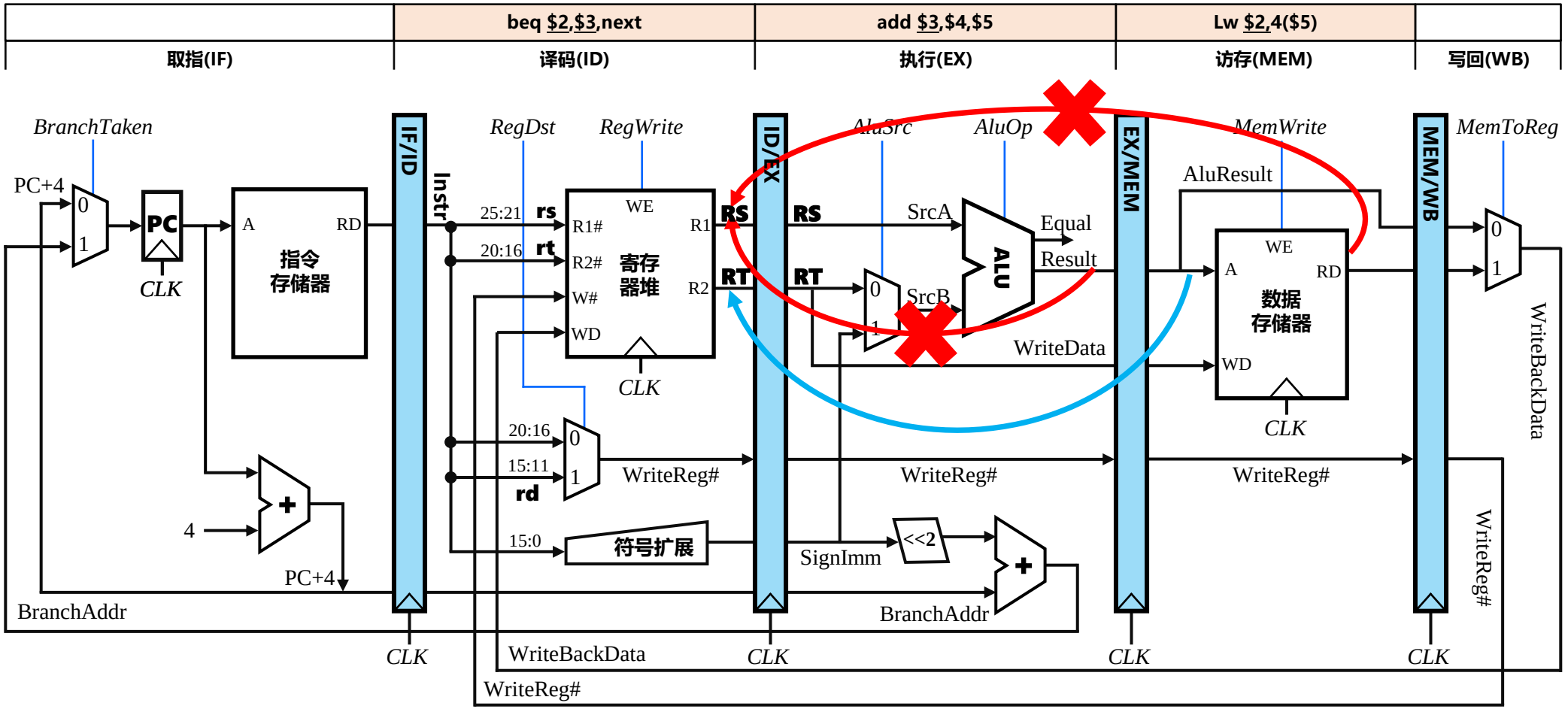



```
LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)
        + RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)
```

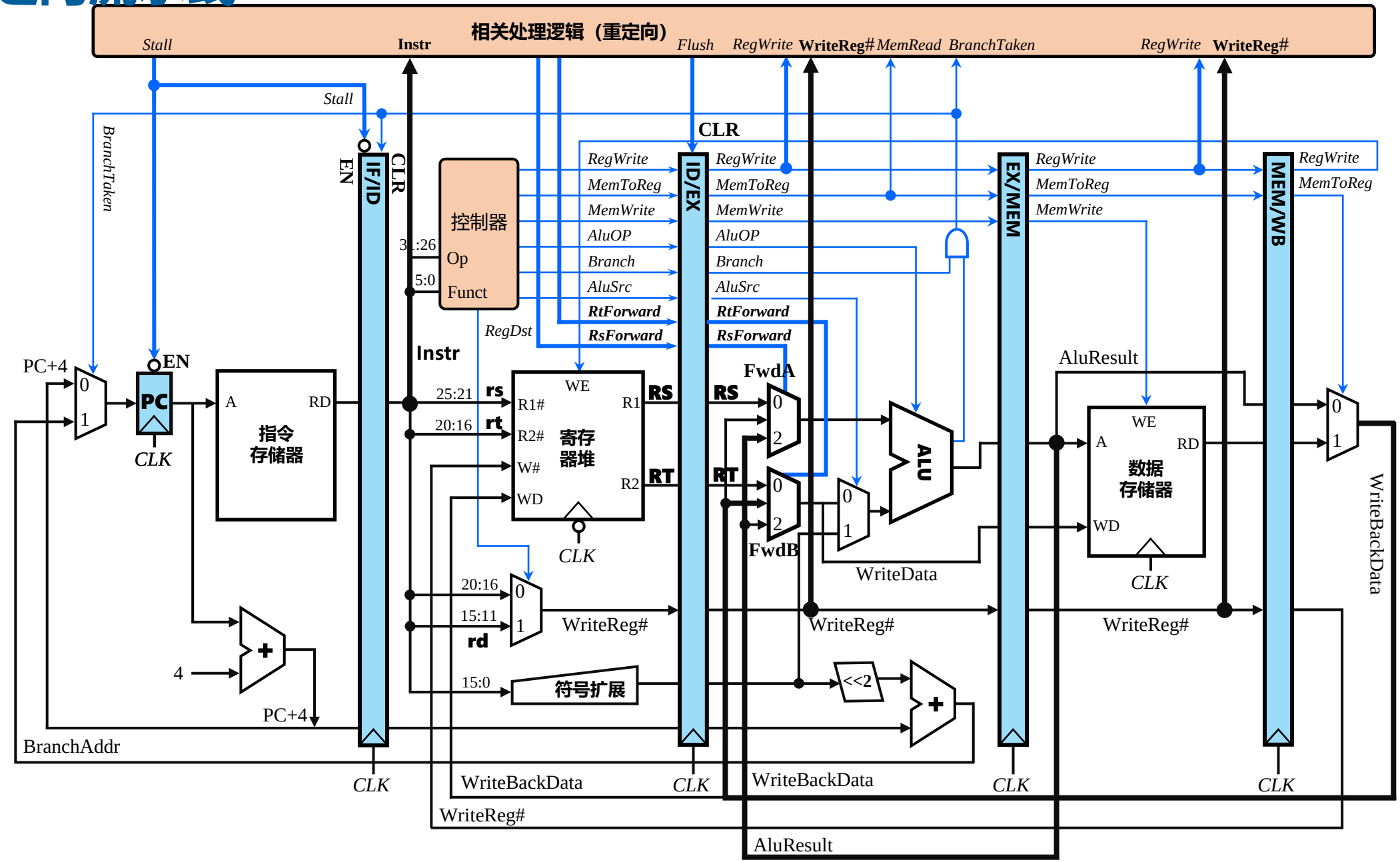
# 插入气泡消除Load-Use相关



# 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
RegDst
ID/EX
EX/MEM
MEM/WB
Instr
PC+4
EN
PC
CLK
指令
存储器
A
RD
25:21
20:16
rs
rt
WE
R1#
R2#
W#
WD
R1
R2
寄存
器堆
RS
RT
CLK
FwdA
FwdB
ALU
AluResult
WriteData
20:16
15:11
rd
WriteReg#
4
PC+4
15:0
符号扩展
<<2
BranchAddr
WriteBackData
WriteReg#
AluResult
数据
存储器
WE
A
RD
WD
CLK
WriteBackData

# 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

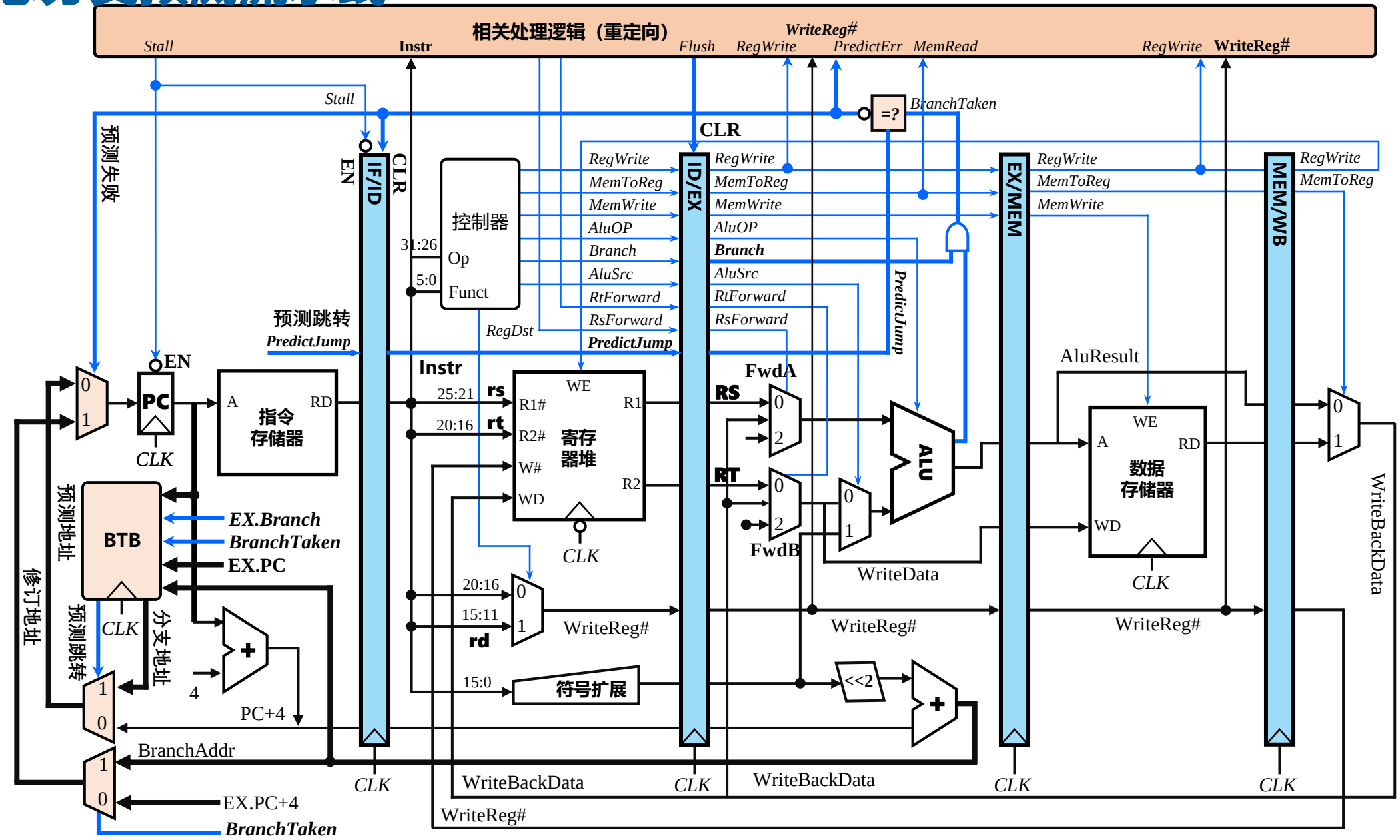
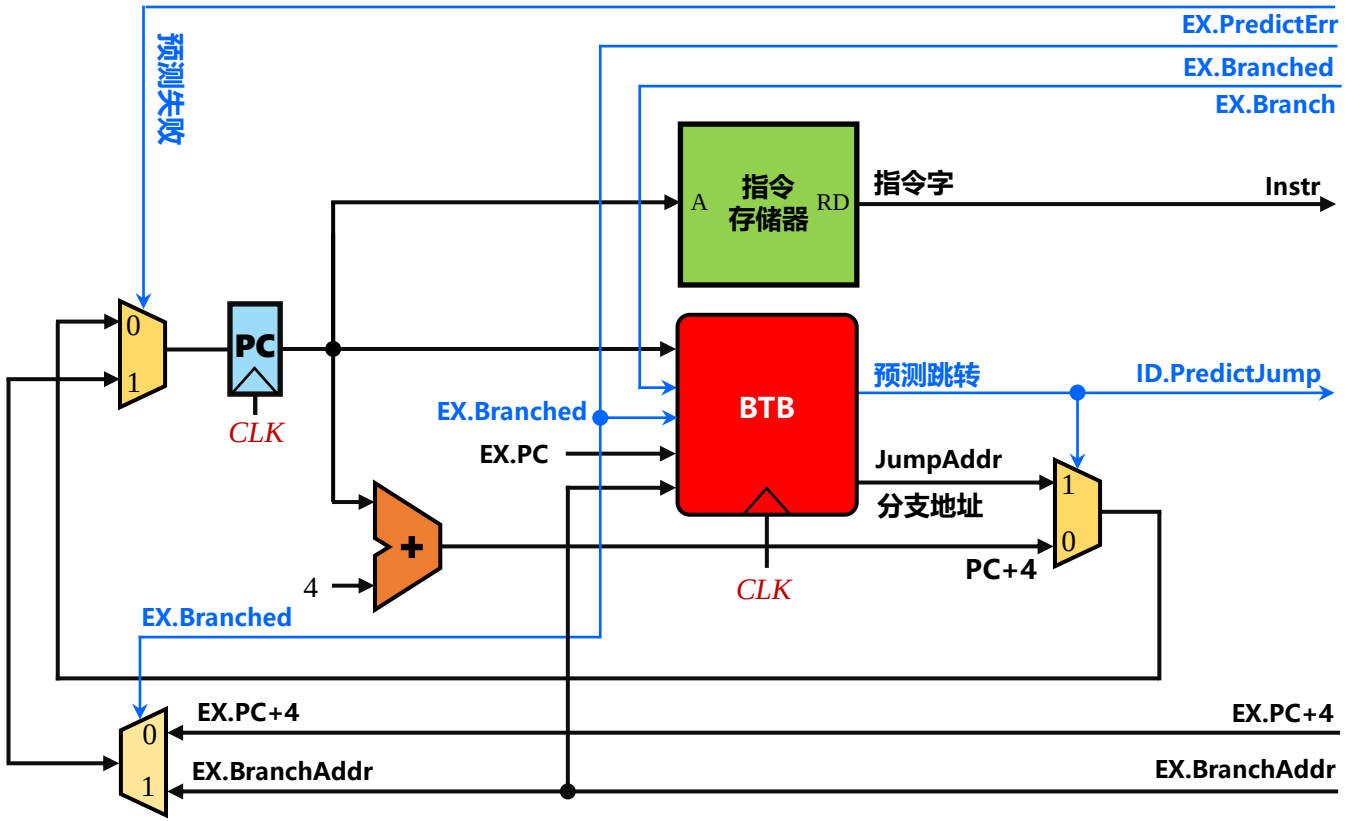
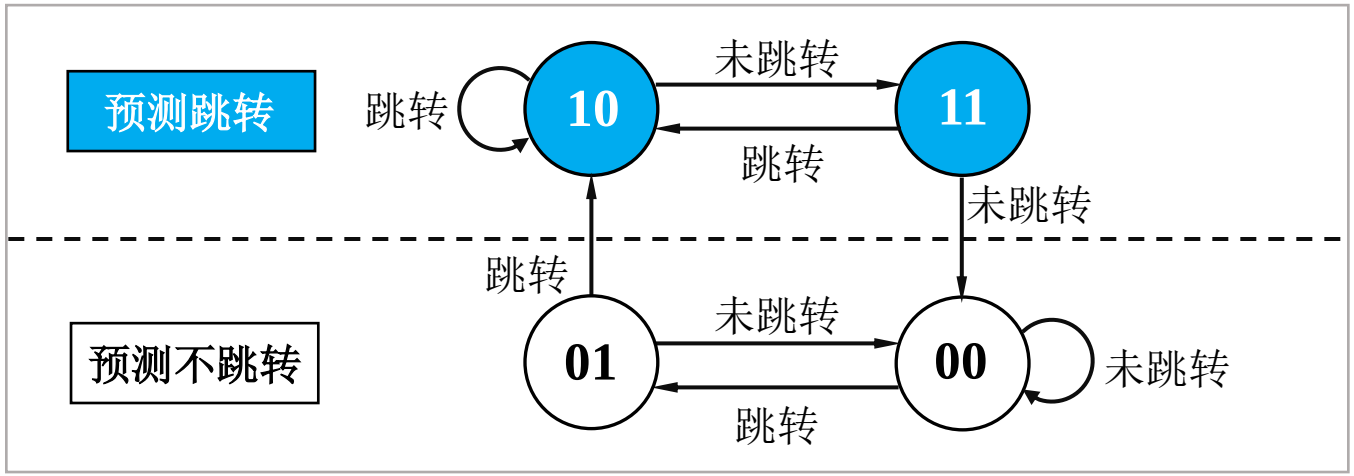

相关处理逻辑（重定向）
Stall
Instr
Flush
RegWrite
WriteReg#
PredictErr
MemRead
RegWrite
WriteReg#
Stall
=?
BranchTaken
CLR
预测失败
EN
IF/ID
CLR
控制器
31:26
Op
5:0
Funct
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
PredictJump
ID/EX
RegWrite
MemToReg
MemWrite
AluOP
Branch
AluSrc
RtForward
RsForward
PredictJump
EX/MEM
RegWrite
MemToReg
MemWrite
MEM/WB
RegWrite
MemToReg
预测跳转
PredictJump
RegDst
Instr
FwdA
RS
RT
FwdB
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
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


EX.PredictErr
EX.Branched
EX.Branch
预测失效
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

# 性能分析

- 理想情况下指令流水线CPI应该是1，但由于流水阻塞或清空会损失一些周期，因此指令流水线的实际CPI比理想情况下的CPI略高
- 流水线的最小时钟周期取决于各功能段中最慢功能段的关键延迟

$$T_{min\_clk} = \max(T_{if\_max}, T_{id\_max}, T_{ex\_max}, T_{mem\_max}, T_{wb\_max})$$

| 序号 | 功能段 | 标识 | 功能段延迟 | 65nm CMOS 工艺实际值 |
|---|---|---|---|---|
| 1 | IF | $T_{if\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | 300ps = 30 + 250 + 20 |
| 2 | ID | $T_{id\_max}$ | $2(T_{clk\_to\_q} + T_{RF\_read} + T_{setup})$ | 400ps = 2 × (30 + 150 + 20) |
| 3 | EX | $T_{ex\_max}$ | $T_{clk\_to\_q} + 2T_{mux} + T_{alu} + T_{setup}$ | 300ps = 30 + 2 × 25 + 200 + 20 |
| 4 | MEM | $T_{mem\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | 300ps = 30 + 250 + 20 |
| 5 | WB | $T_{wb\_max}$ | $T_{clk\_to\_q} + T_{mux} + T_{setup}$ | 75ps = 30 + 25 + 20 |

思考：如果可以优化流水线一个功能部件的关键延迟以提升处理器整体性能，应该选择哪个部件进行优化?如果这种优化与成本是线性关系，如何优化才能使处理器性能达到最优，且成本最低?

| 序号 | 功能段 | 标识 | 功能段延迟 | 65nm CMOS 工艺实际值 |
|---|---|---|---|---|
| 1 | IF | $T_{if\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | $300ps = 30 + 250 + 20$ |
| 2 | ID | $T_{id\_max}$ | $2(T_{clk\_to\_q} + T_{RF\_read} + T_{setup})$ | $400ps = 2 \times (30 + 150 + 20)$ |
| 3 | EX | $T_{ex\_max}$ | $T_{clk\_to\_q} + 2T_{mux} + T_{alu} + T_{setup}$ | $300ps = 30 + 2 \times 25 + 200 + 20$ |
| 4 | MEM | $T_{mem\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | $300ps = 30 + 250 + 20$ |
| 5 | WB | $T_{wb\_max}$ | $T_{clk\_to\_q} + T_{mux} + T_{setup}$ | $75ps = 30 + 25 + 20$ |

思考：如果可以优化流水线一个功能部件的关键延迟以提升处理器整体性能，应该选择哪个部件进行优化?如果这种优化与成本是线性关系，如何优化才能使处理器性能达到最优，且成本最低?

| 序号 | 功能段 | 标识 | 功能段延迟 | 65nm CMOS 工艺实际值 |
|---|---|---|---|---|
| 1 | IF | $T_{if\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | $300ps = 30 + 250 + 20$ |
| 2 | ID | $T_{id\_max}$ | $2(T_{clk\_to\_q} + T_{RF\_read} + T_{setup})$ | $400ps = 2 \times (30 + 150 + 20)$ |
| 3 | EX | $T_{ex\_max}$ | $T_{clk\_to\_q} + 2T_{mux} + T_{alu} + T_{setup}$ | $300ps = 30 + 2 \times 25 + 200 + 20$ |
| 4 | MEM | $T_{mem\_max}$ | $T_{clk\_to\_q} + T_{mem} + T_{setup}$ | $300ps = 30 + 250 + 20$ |
| 5 | WB | $T_{wb\_max}$ | $T_{clk\_to\_q} + T_{mux} + T_{setup}$ | $75ps = 30 + 25 + 20$ |

$$T_{min\_clk} = \max(T_{if\_max}, T_{id\_max}, T_{ex\_max}, T_{mem\_max}, T_{wb\_max})$$

- 优化最慢的功能段
- 这里ID段最慢，应该优化其中的寄存器堆读延迟 TRF_read=150ps，当这个时间延迟优化到100时。ID段时延与IF、EX、MEM 段的相同，再进一步优化没有意义，只会增加成本。

# 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）→ 设置中断原因 → 关中断 → 指令清空

## 同步中断处理

- 保存断点（当前异常指令地址）→ 设置中断原因 → 关中断 → 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