



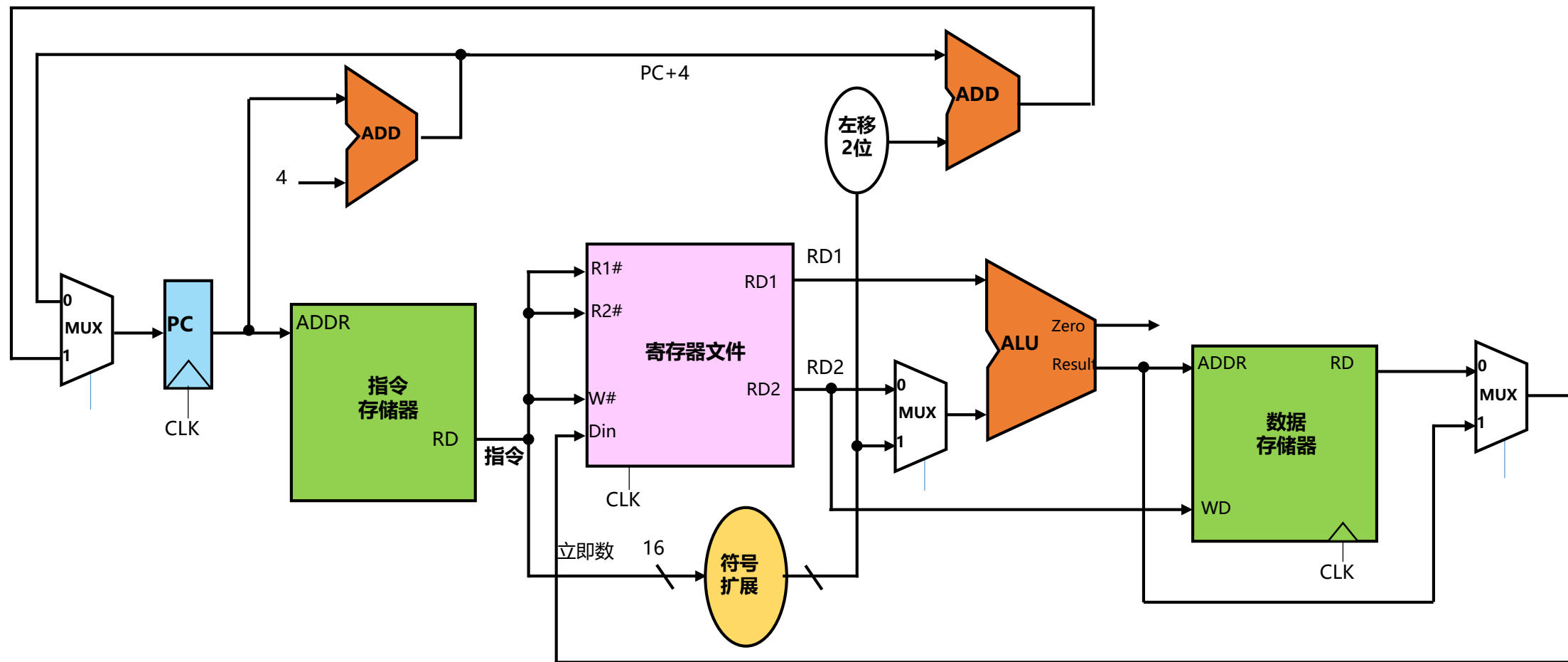
流水线介绍

集美大学 叶从容
2021/8/29

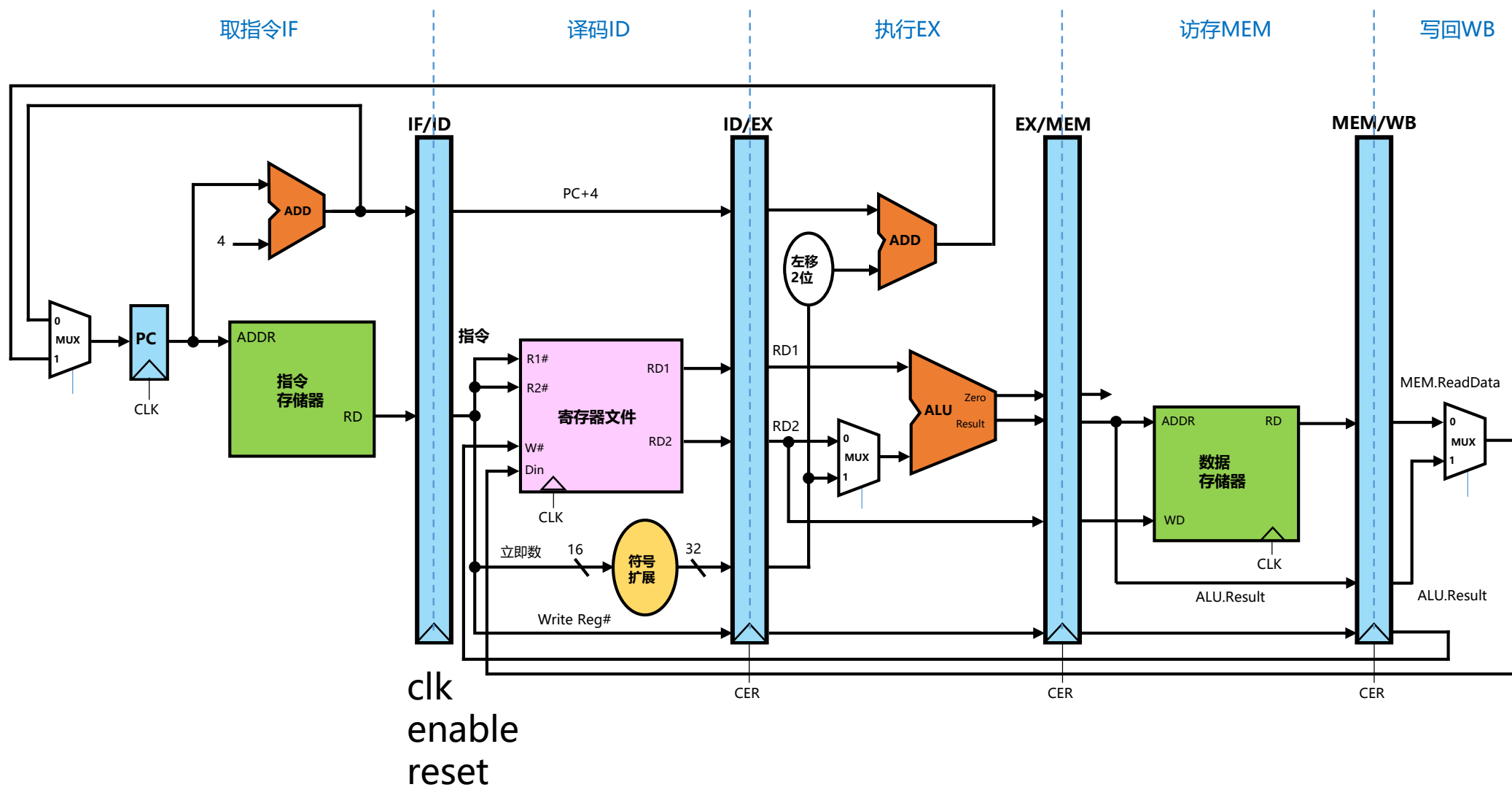
大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

1.1.1 单周期处理器



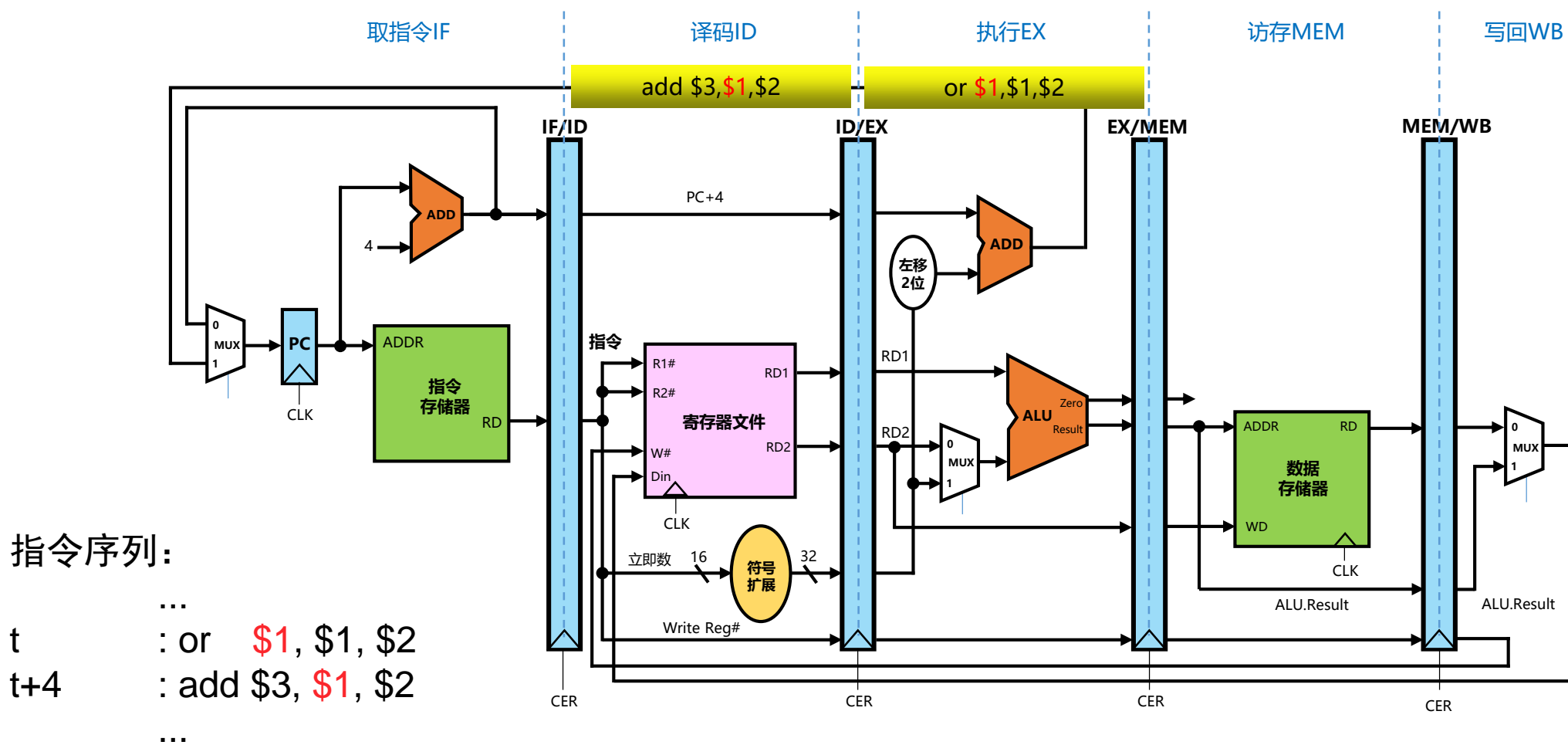
1.1.2 经典五级流水线



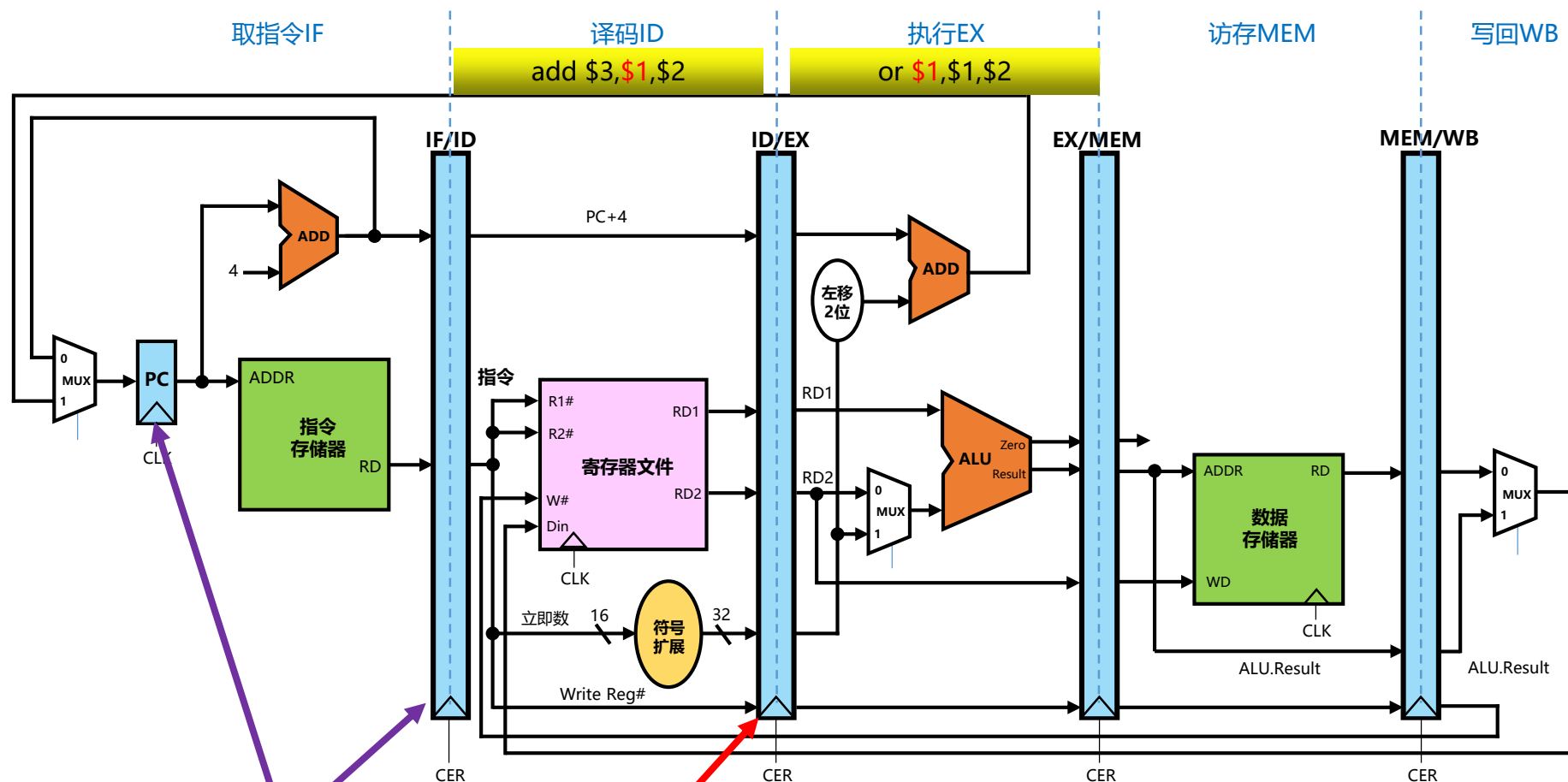
大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

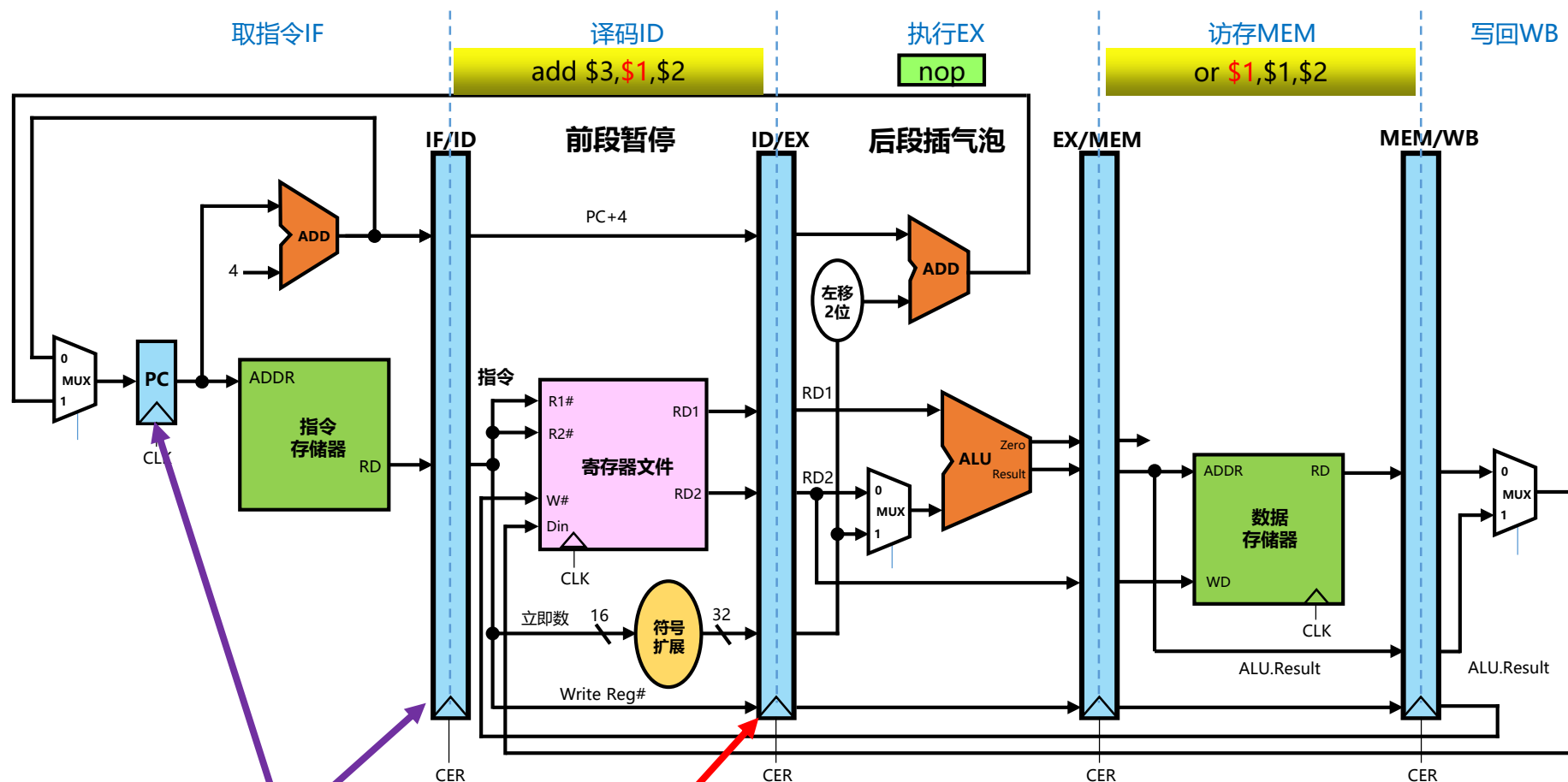
1.2.1 数据冒险



1.2.2 气泡方案0



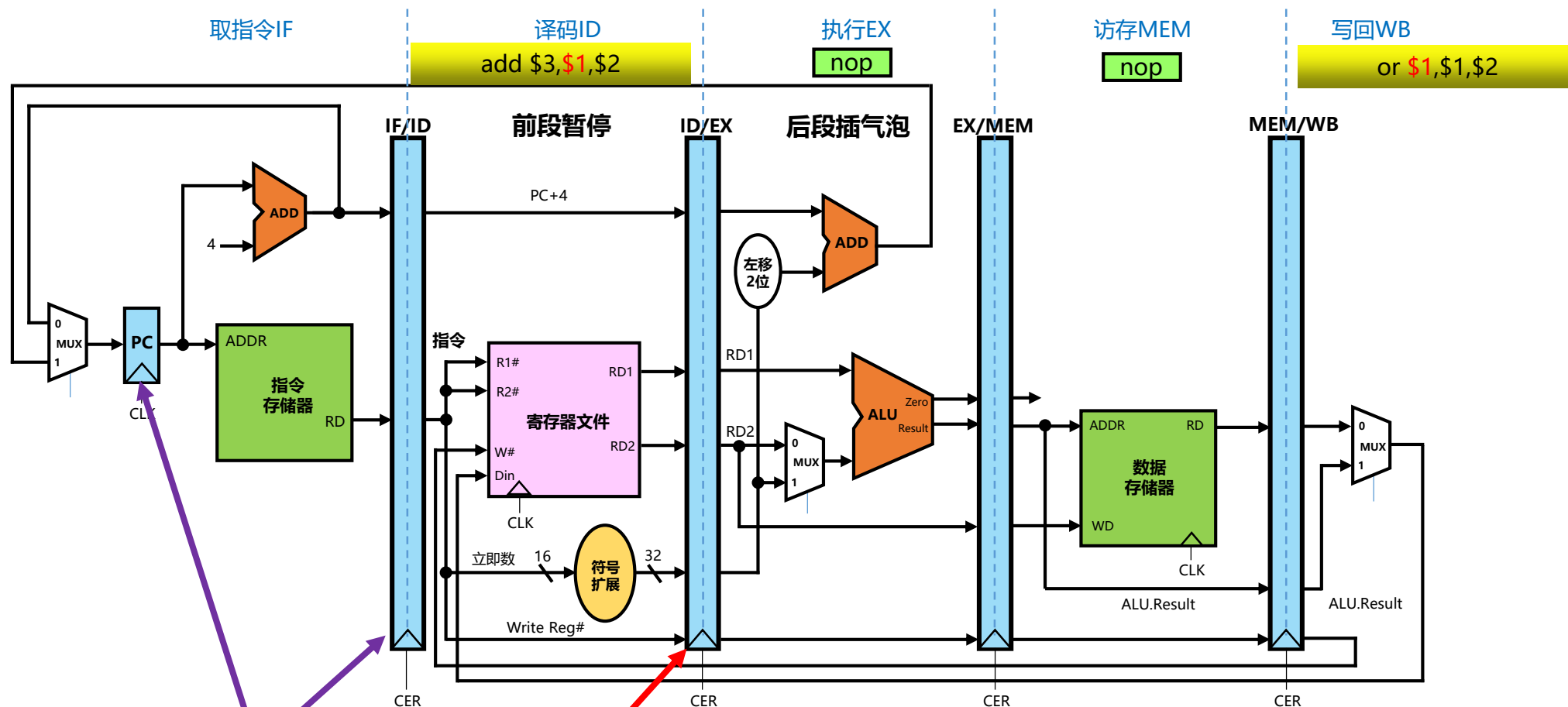
1.2.2 气泡方案1



暂停stall

插入气泡

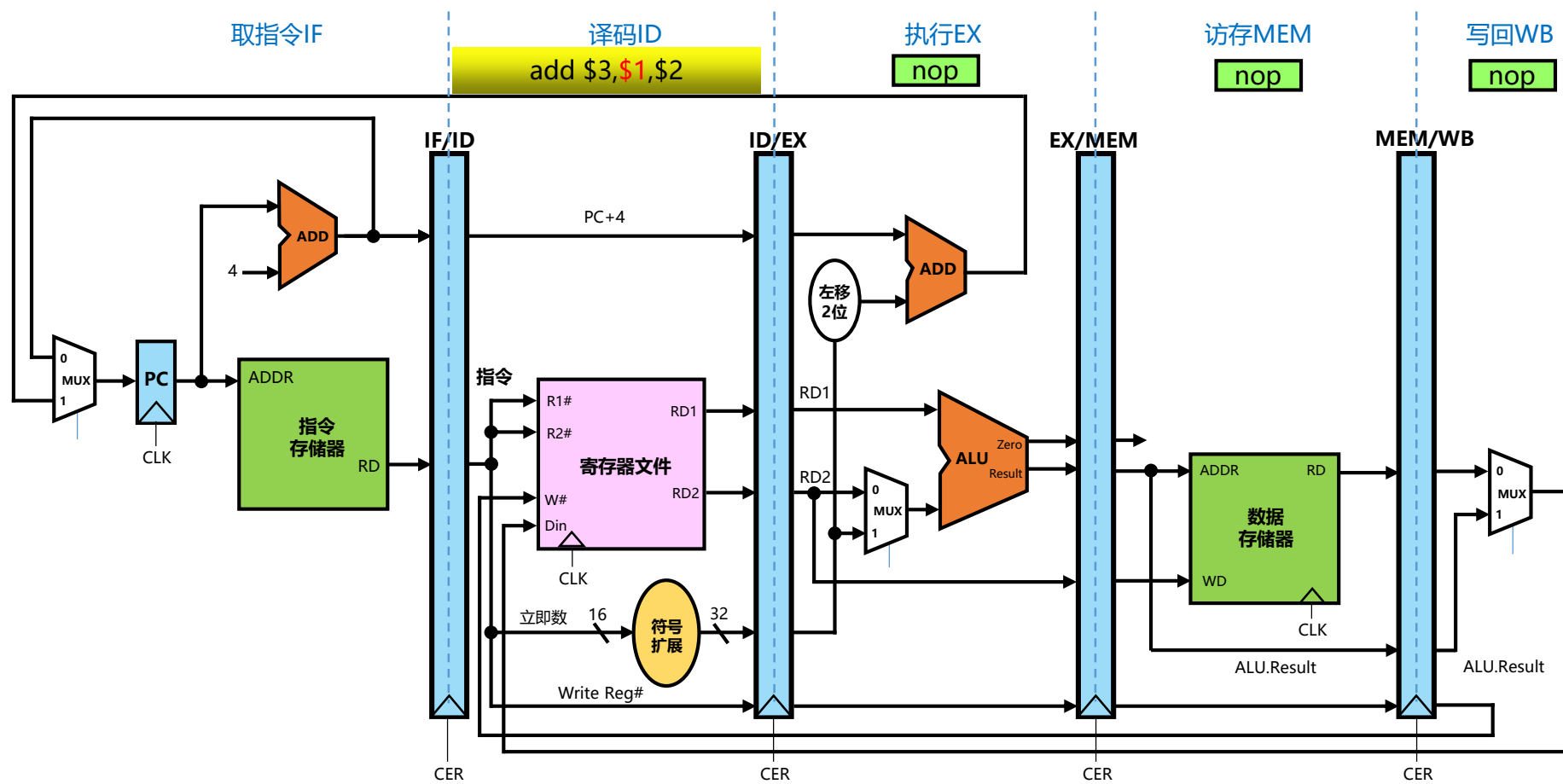
1.2.2 气泡方案2



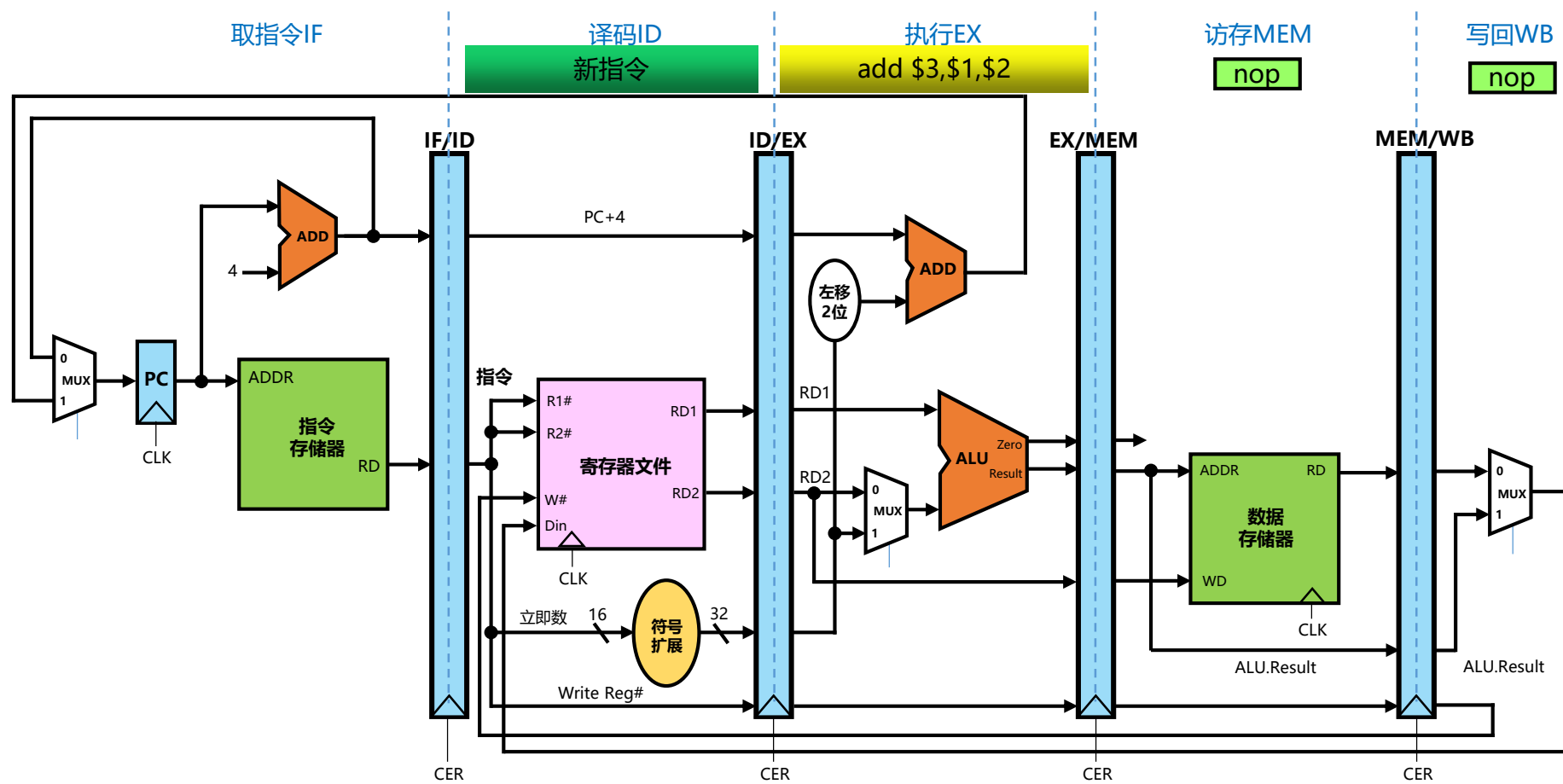
暂停stall

插入气泡

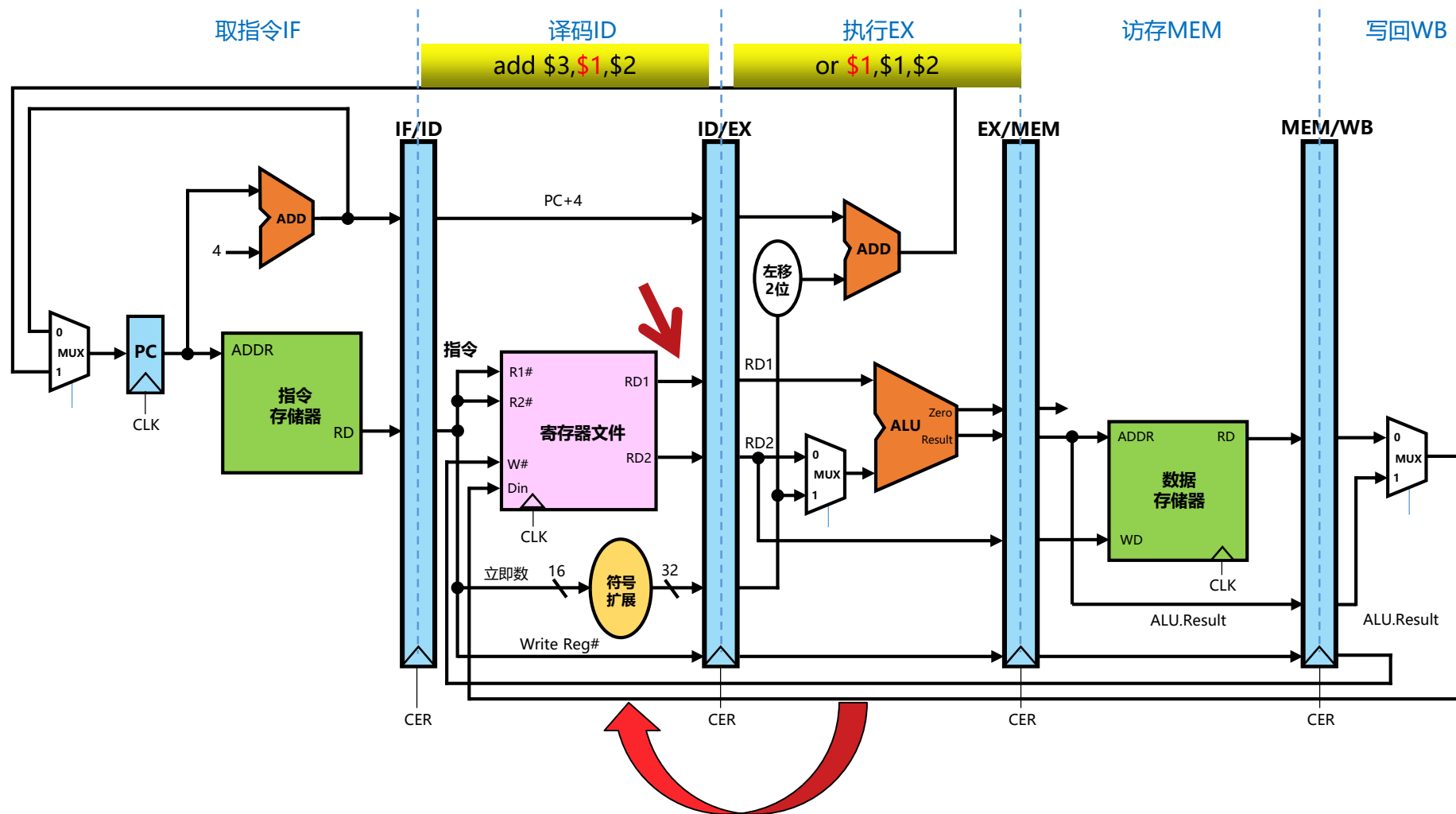
1.2.2 气泡方案3



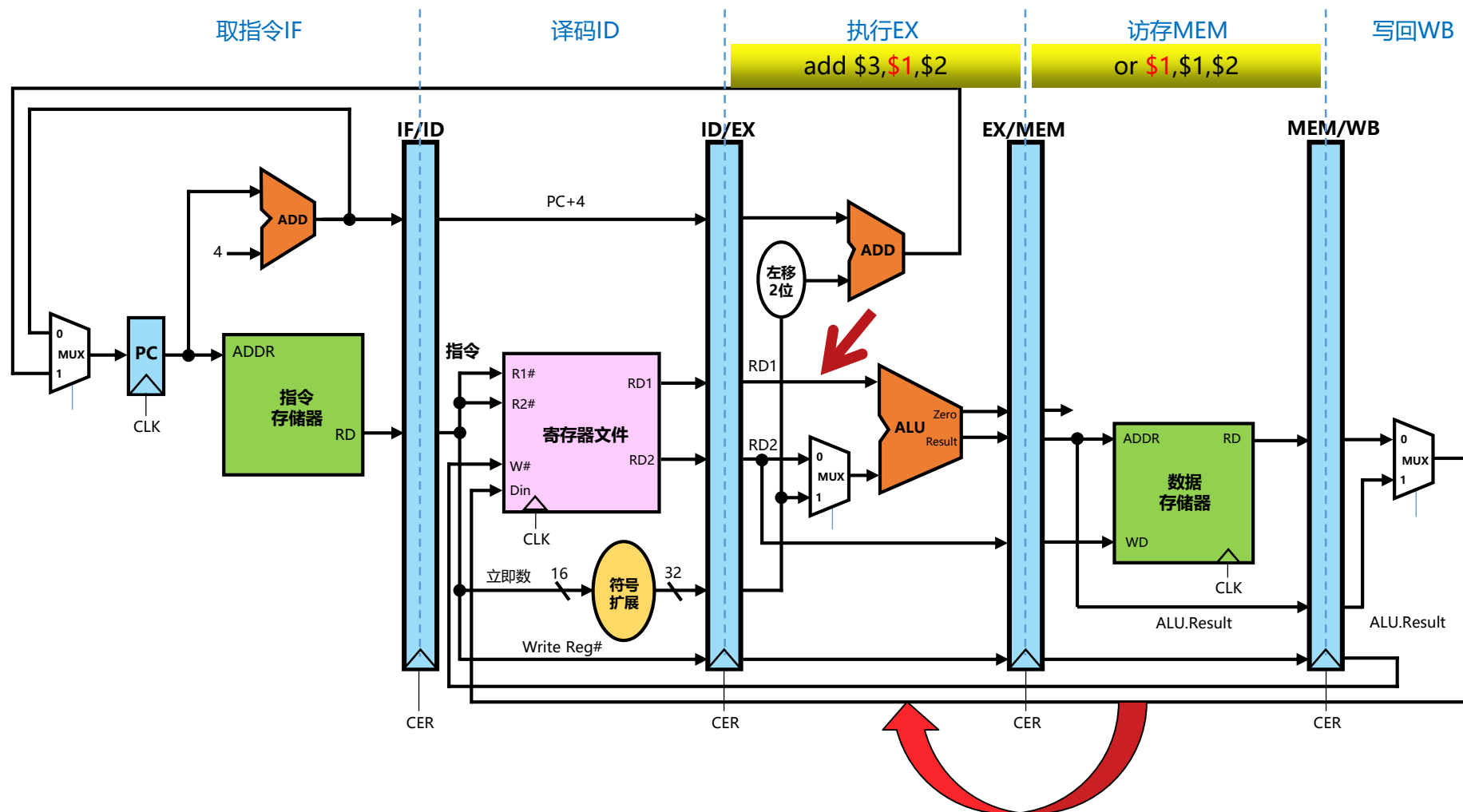
1.2.2 气泡方案4



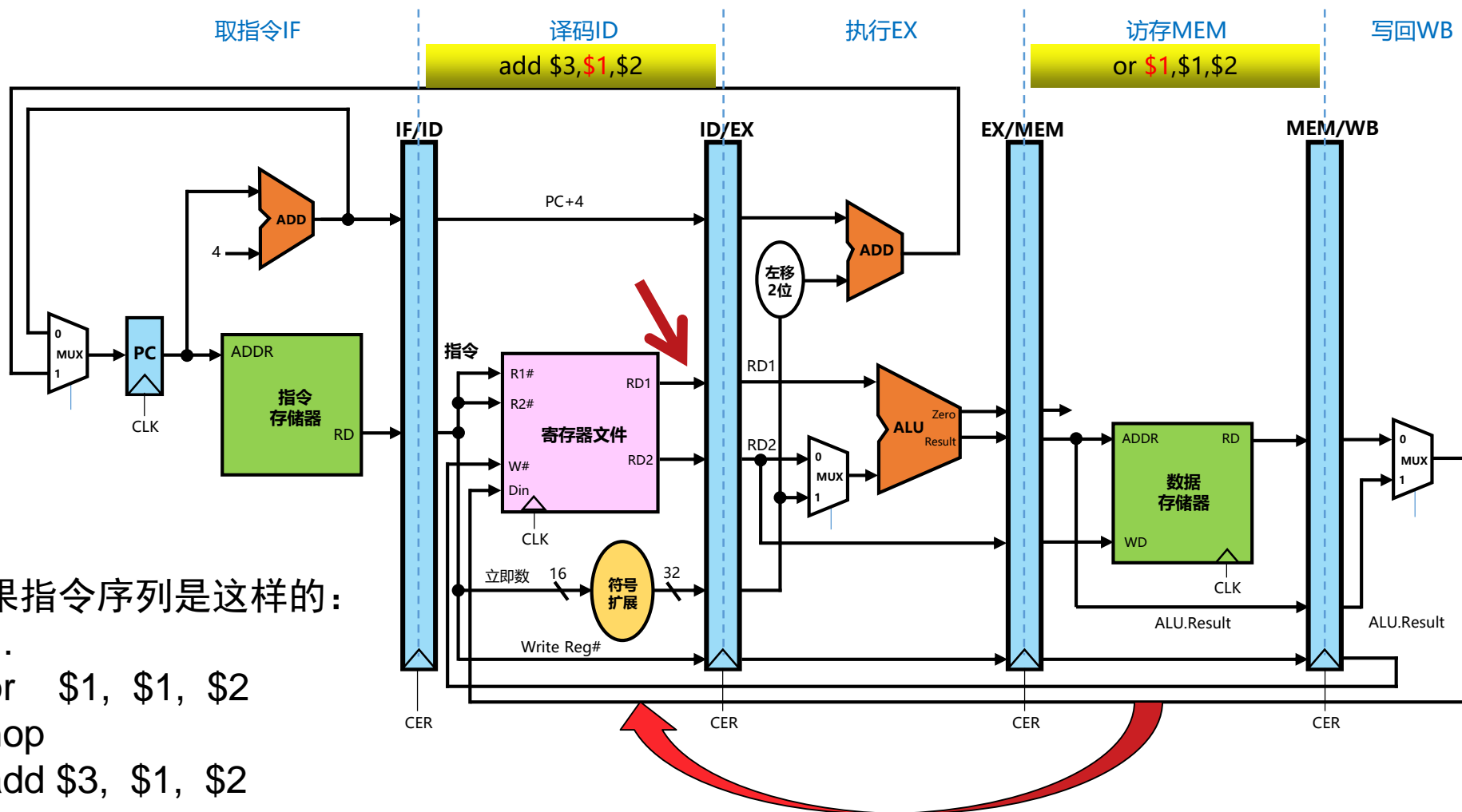
1.2.3 数据重定向方案0 (forwarding/bypass)



1.2.3 数据重定向方案1



1.2.3 数据重定向方案2



▲ 思考：如果指令序列是这样的：

...

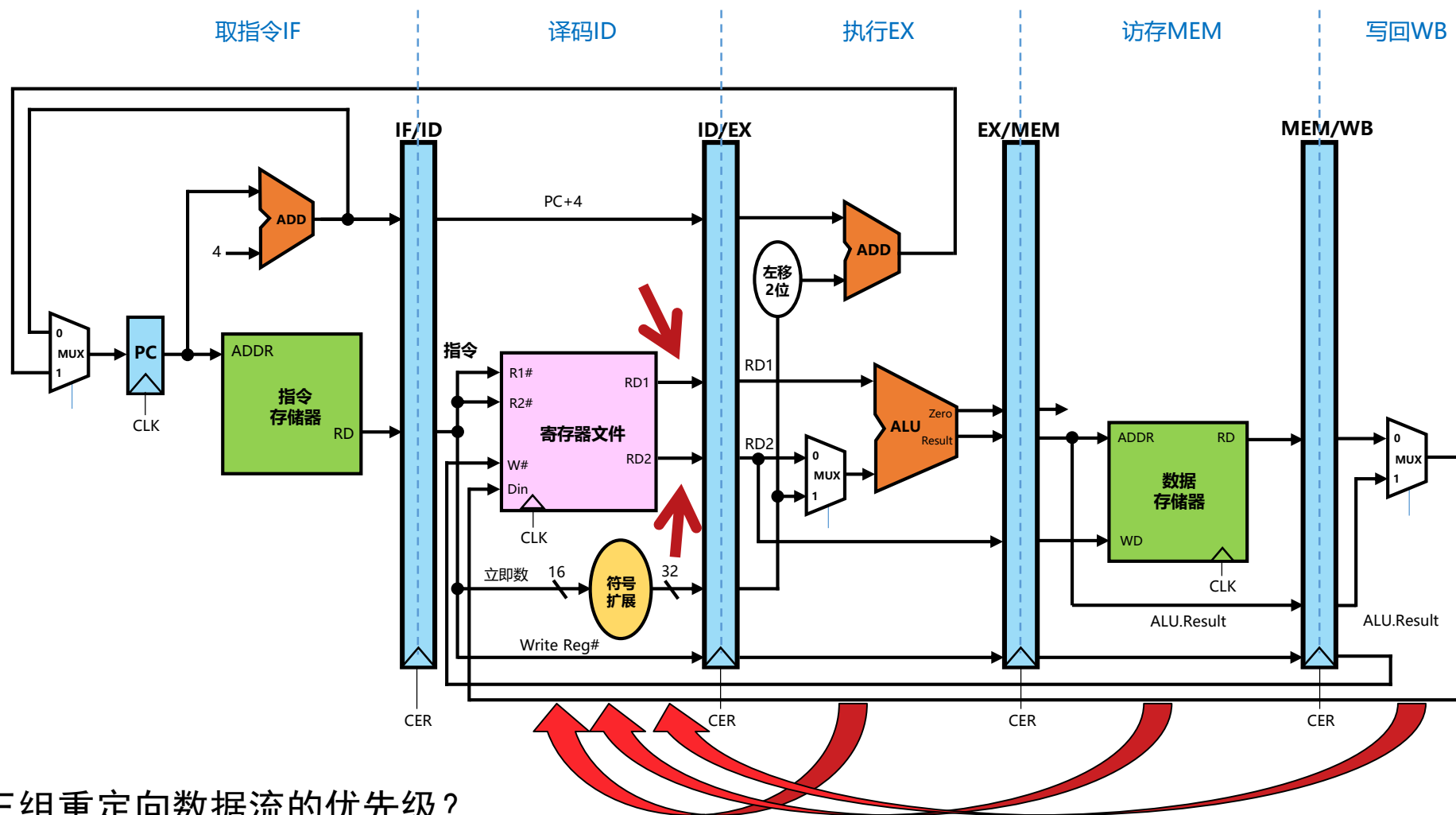
t: or \$1, \$1, \$2

t+4: nop

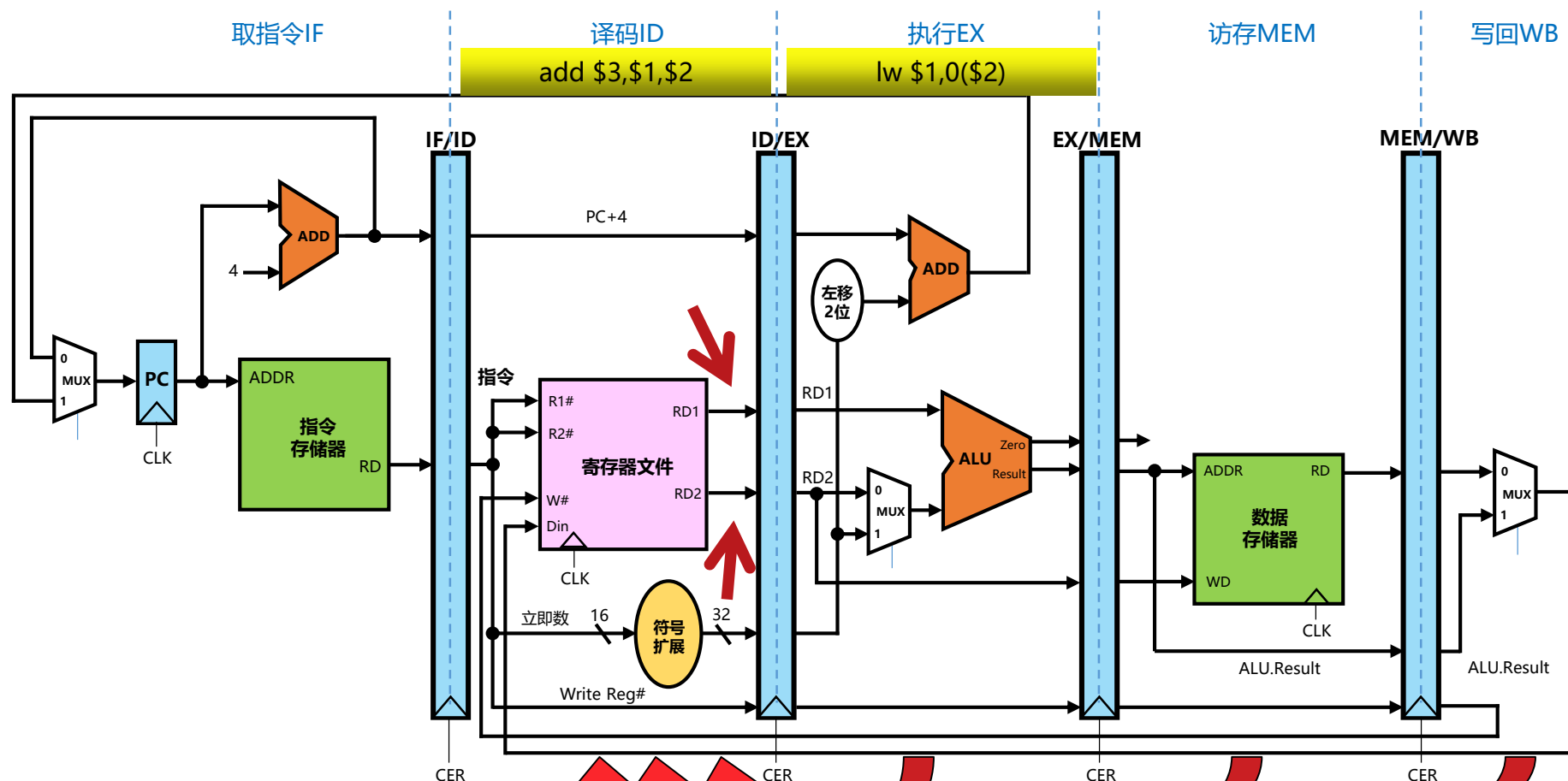
t+8: add \$3, \$1, \$2

...

1.2.3 数据重定向方案3

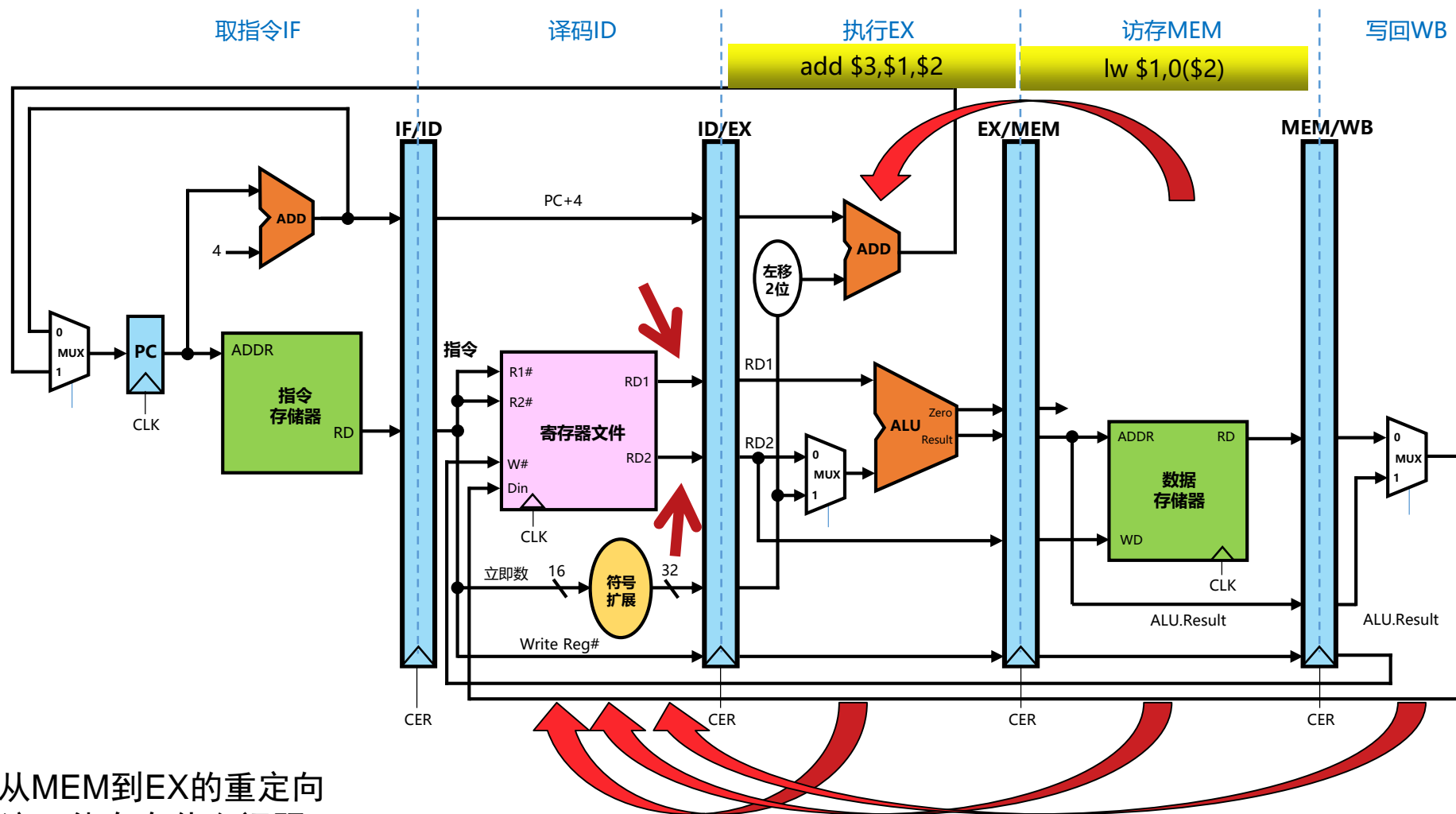


1.2.3 数据重定向方案4

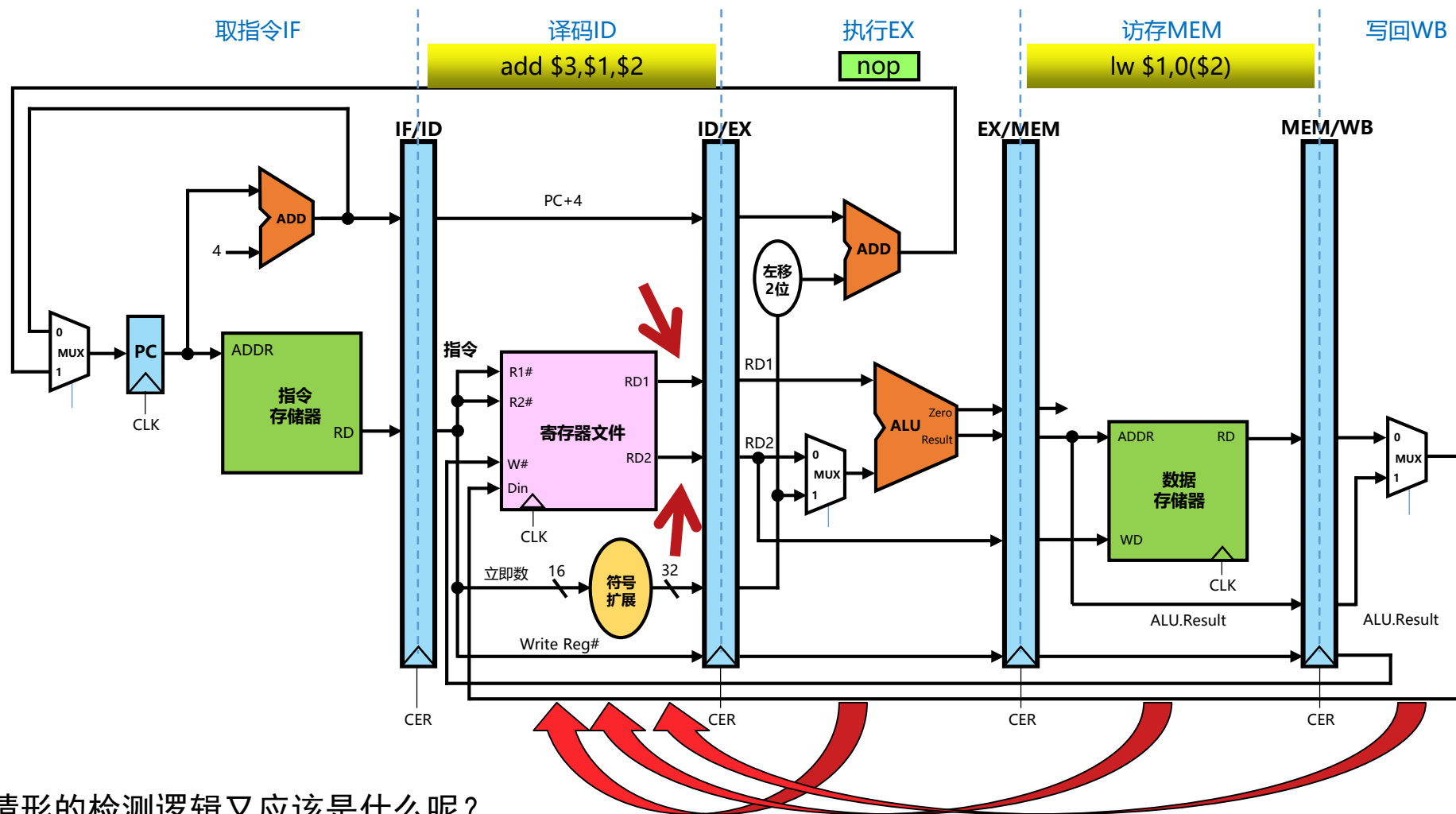


▲ 思考：EX阶段的lw指令，尚未取到\$1寄存器的值

1.2.3 数据重定向方案4



1.2.3 数据重定向方案4

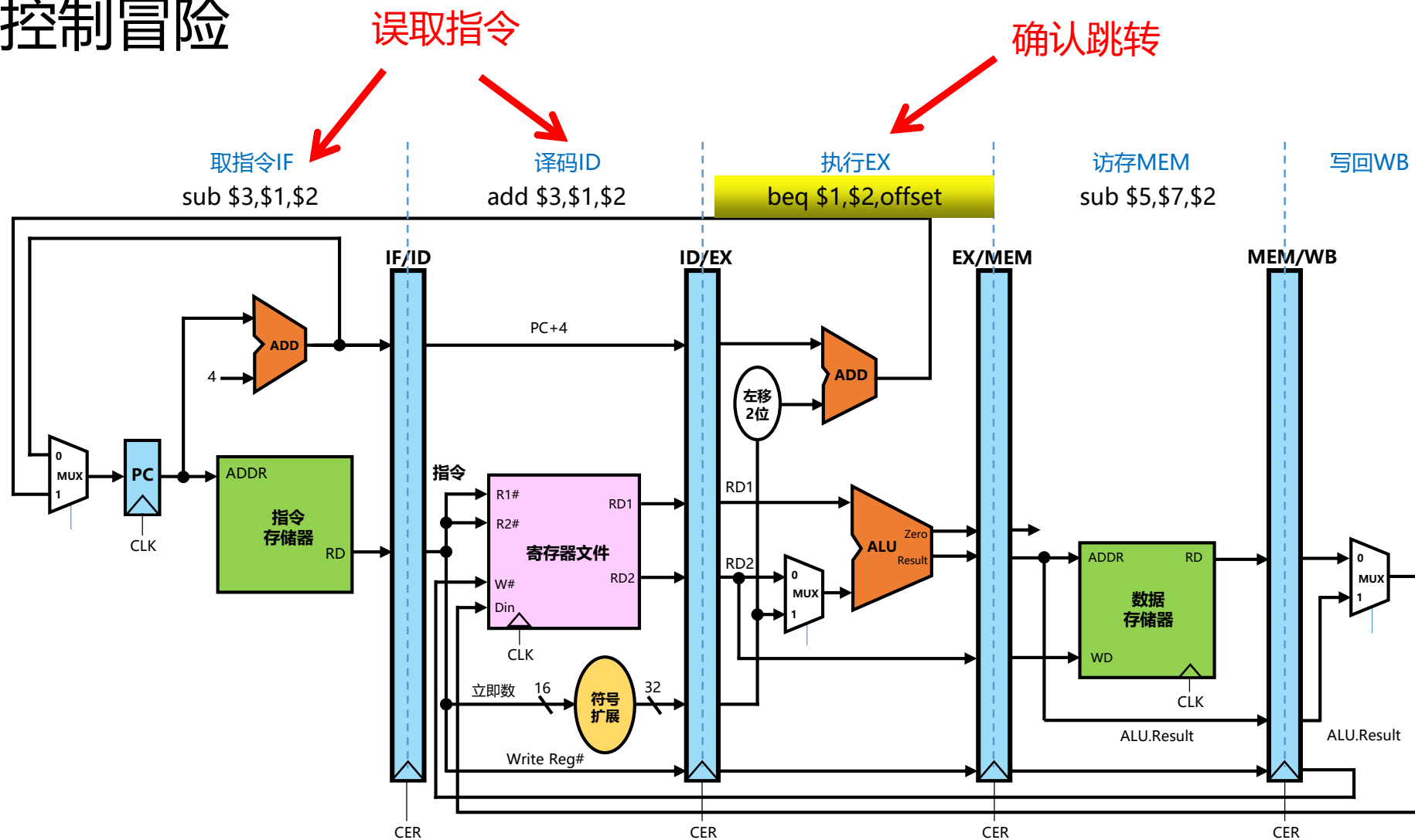


思考：这种情形的检测逻辑又应该是什么呢？
既要插气泡又要暂停，应该缓冲寄存器组什么控制信号呢？

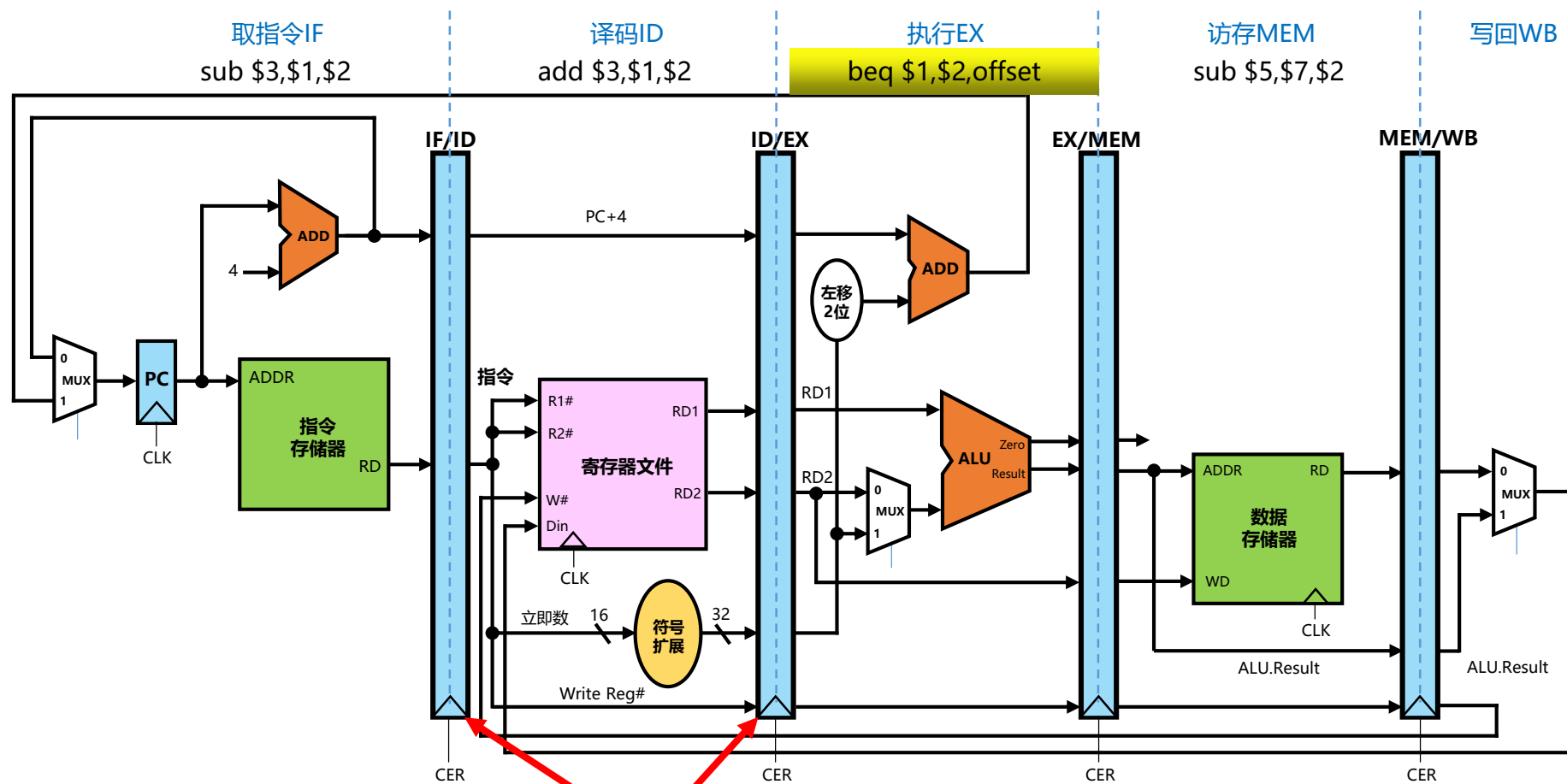
大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

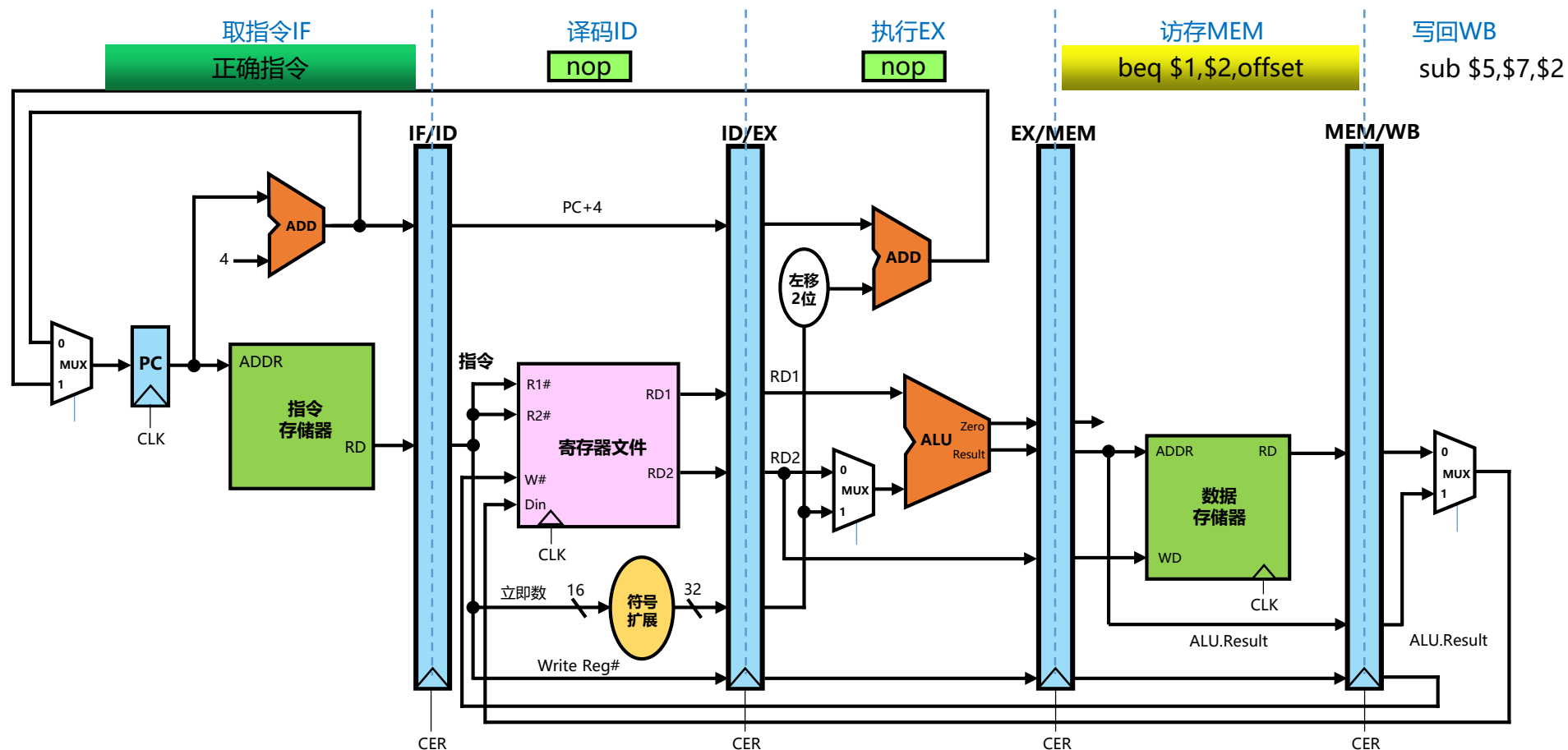
1.3.1 控制冒险



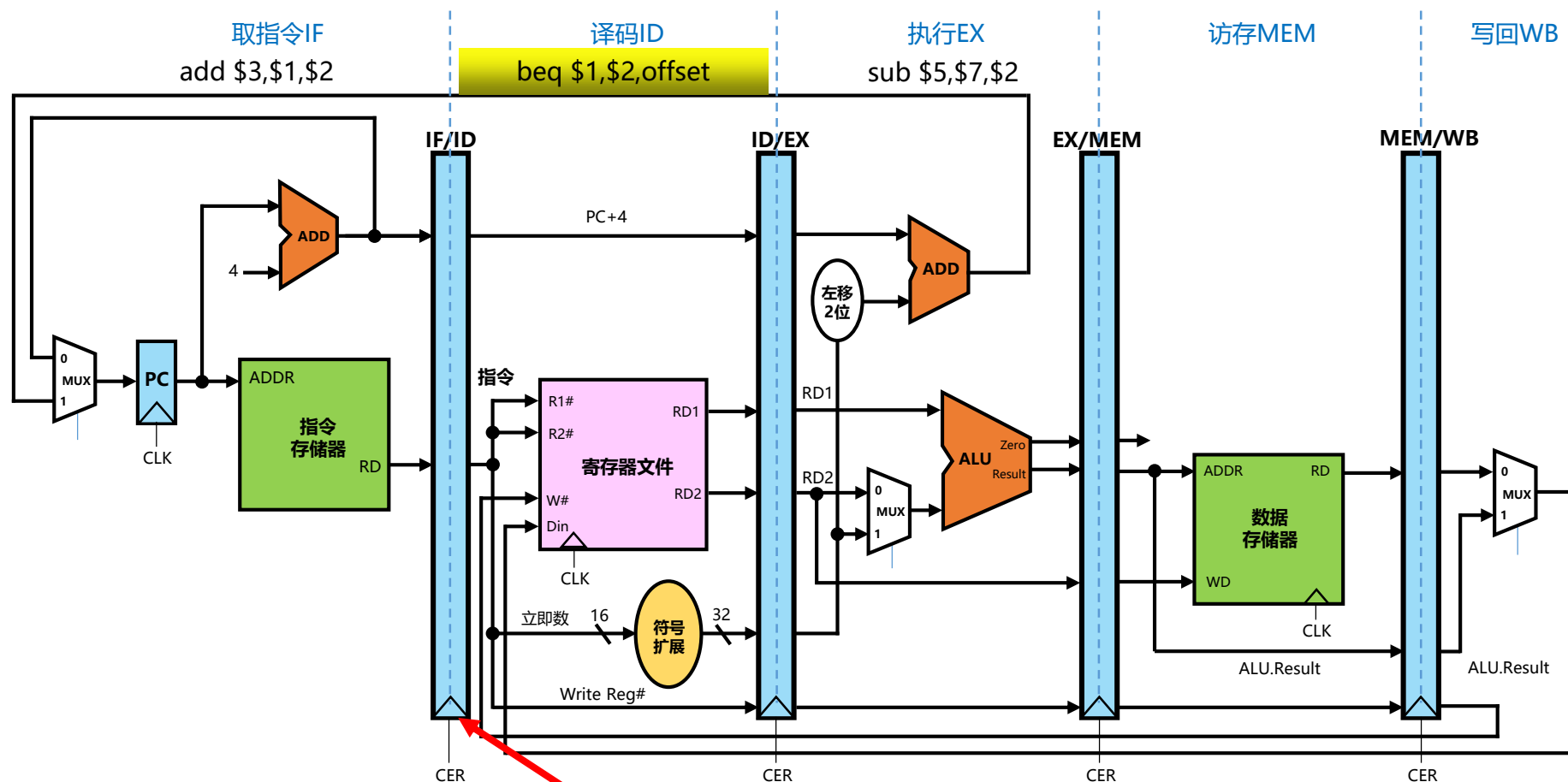
1.3.2 气泡方案0



1.3.2 气泡方案1



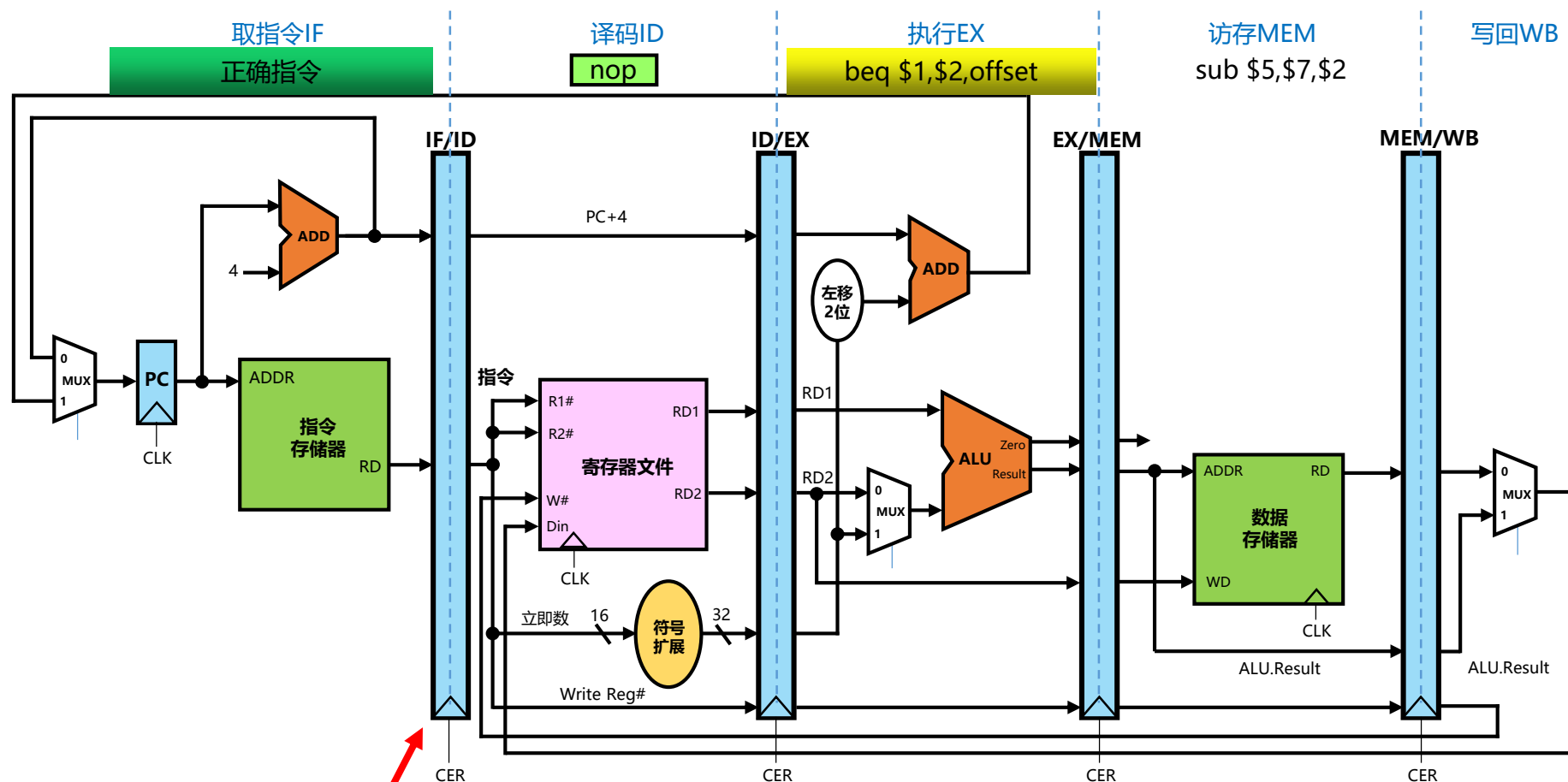
1.3.3 控制冒险的优化思路0



▲ 要点：提前修改机器状态

插入气泡

1.3.3 控制冒险的优化思路1



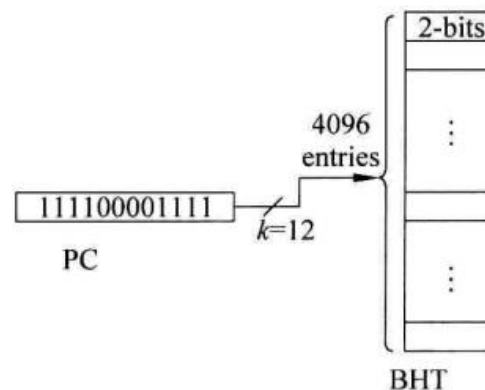
▲ 要点：减少一条空指令

1.3.3 控制冒险的优化思路2

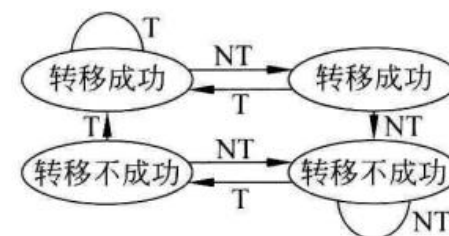
- 根据转移指令的**转移历史**，预测该指令未来的跳转方向与转移目标
- PHT, Pattern History Table, 转移模式历史表

for(i=0;i<10;i++) {...}

B1转移模式 (1111111110)



(a) PHT表原理



(b) 两位饱和计数器

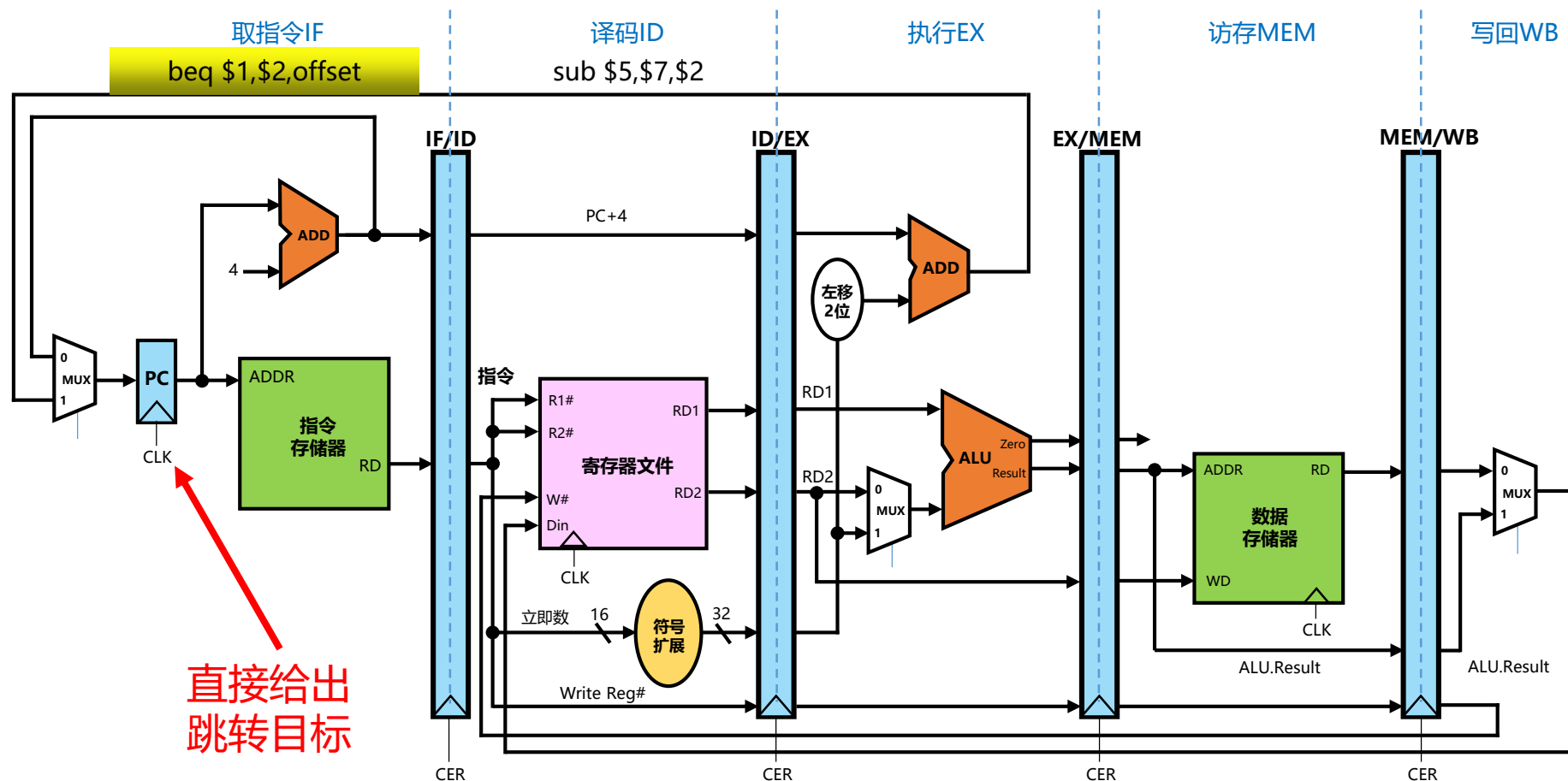
图 8.12 两位 PHT 原理

for(i=0;i<10;i++) for(j=0;j<10;j++) {...}

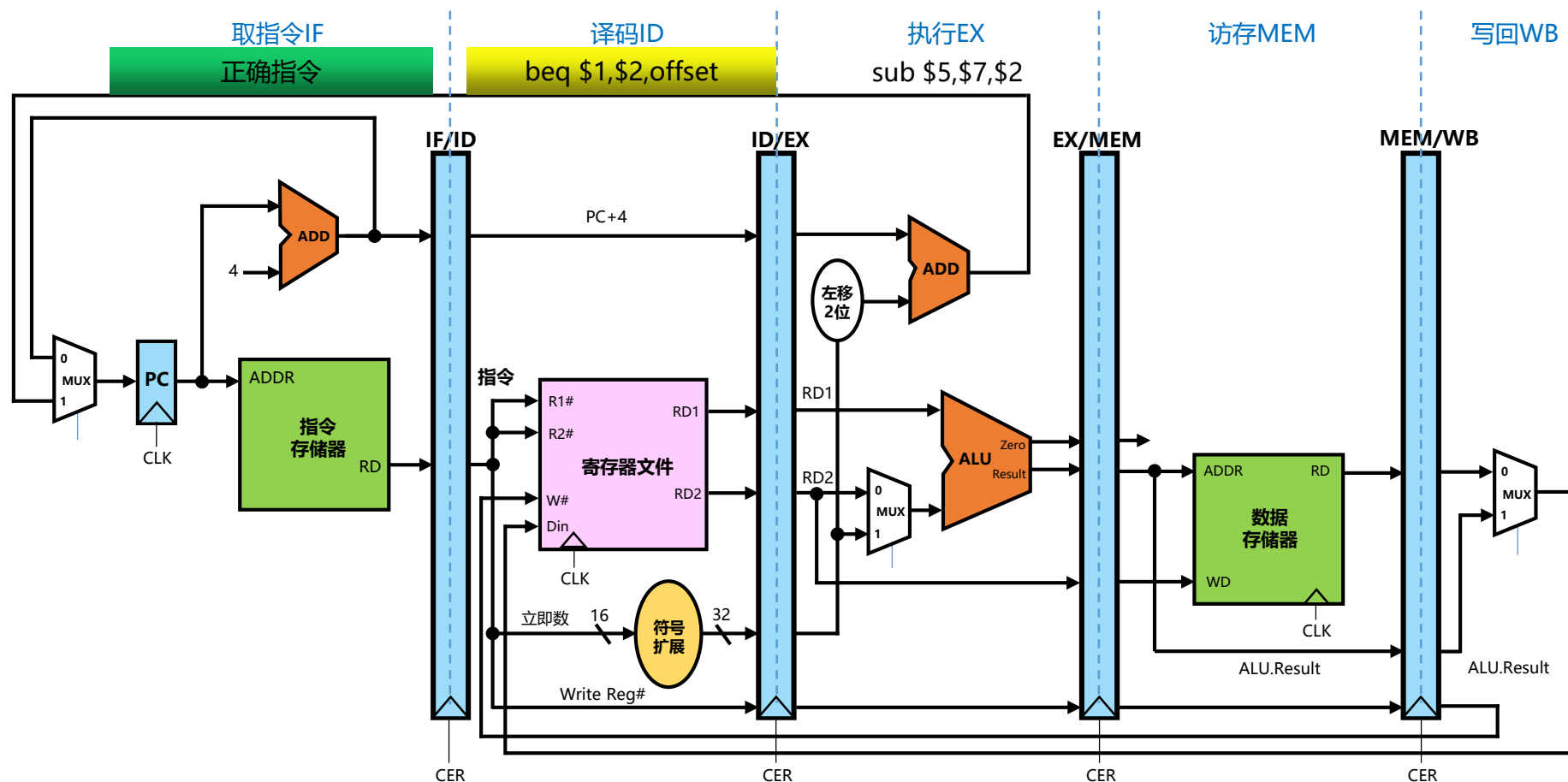
B1转移模式 (1111111110)

B2转移模式 (1111111110 1111111110 1111111110 1111111110 ...1111111110)

1.3.3 控制冒险的优化思路3



1.3.3 控制冒险的优化思路4



- 要点：越早修改机器状态越好，不惜用猜的
- (1) 识别出这是一条分支指令；
 - (2) 猜错了要能改。

大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

2.1 乱序技术概述

- 乱序流水线是指，允许后面不相关的指令提前执行，不必等待前面执行较慢的指令。
- 多发射是指，复制多份功能部件，一拍同时取回多条指令，一拍同时执行多条指令。
- 一个是“允许超车”，一个是“多车道”，两项技术经常结合起来一起使用。
- 可乱序多发执行的理想指令序列：
 - MUL R3, R4, R5
 - ADD R6, R7, R8
 - SUB R2, R4, R7
 - SLL R9, R7, R5

2.1 乱序多发技术参考图

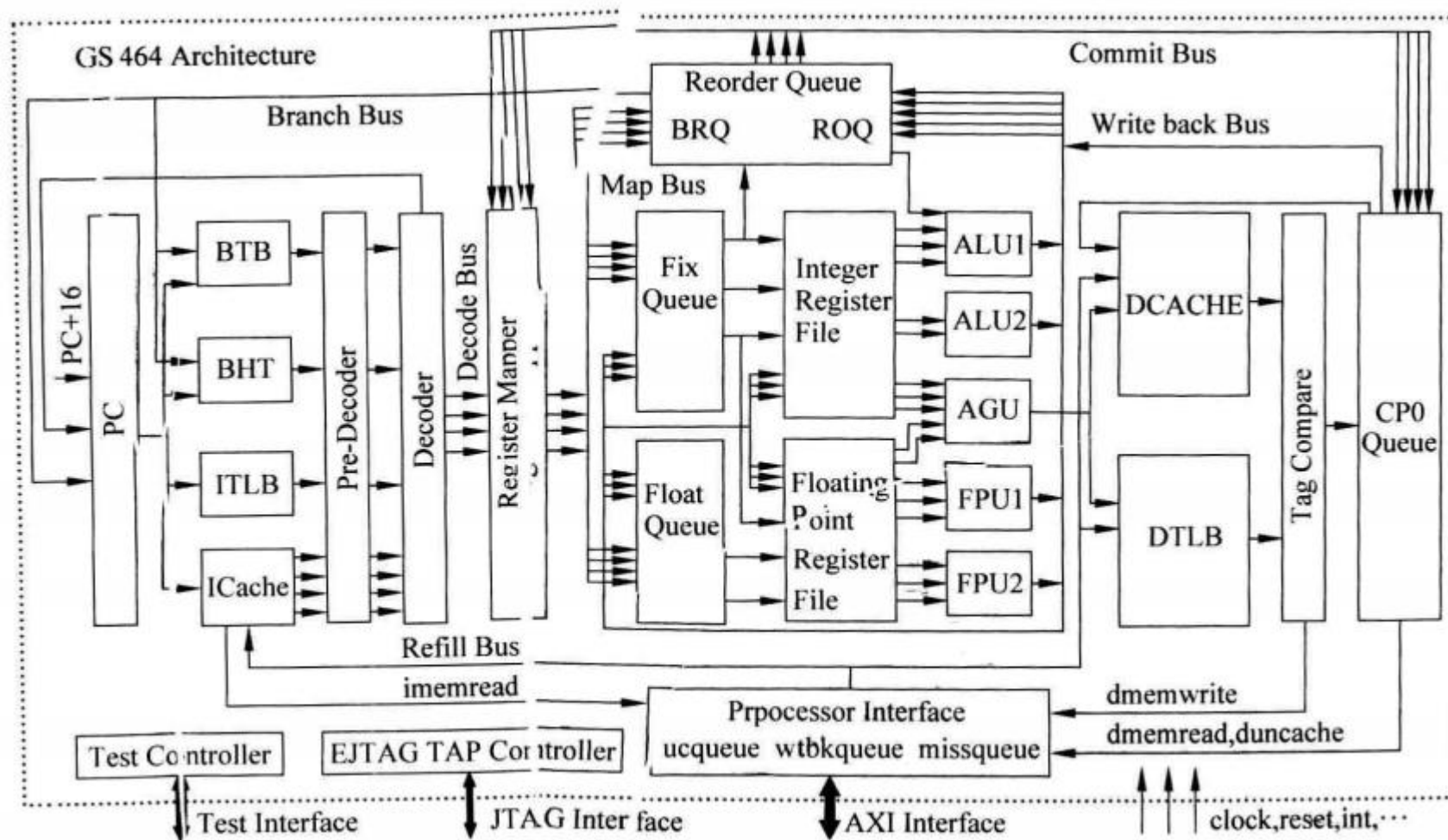
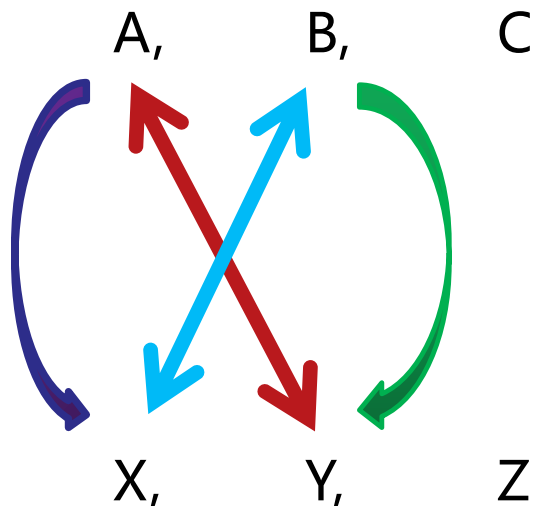


图 7.9 龙芯 2 号处理器结构框图

- 关注模块：
 - Fix Queue
 - Reorder Queue
 - Register Mapper

2.1 乱序流水线的挑战

- MUL



- ADD

示例代码:

- 00 DIV F0, F1, F2
- 04 MUL F3, F0, F2
- 08 ADD F0, F1, F2
- 0C MUL F3, F0, F2

- 任意两条指令都不可以乱序

- 红箭头, $A == Y$

- RAW相关 (Read After Write)
- 乱序导致ADD指令读到错误的旧值

- 蓝箭头, $B == X$

- WAR相关 (Write After Read)
- 乱序导致MUL指令读到错误的新值

- 紫箭头, $A == X$

- WAW相关 (Write After Write)
- 乱序导致寄存器被写入错误的旧值

- 绿箭头, $B == Y$

- RAR不会出错, 可以乱序执行

大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

2.2 Tomasulo算法详解0

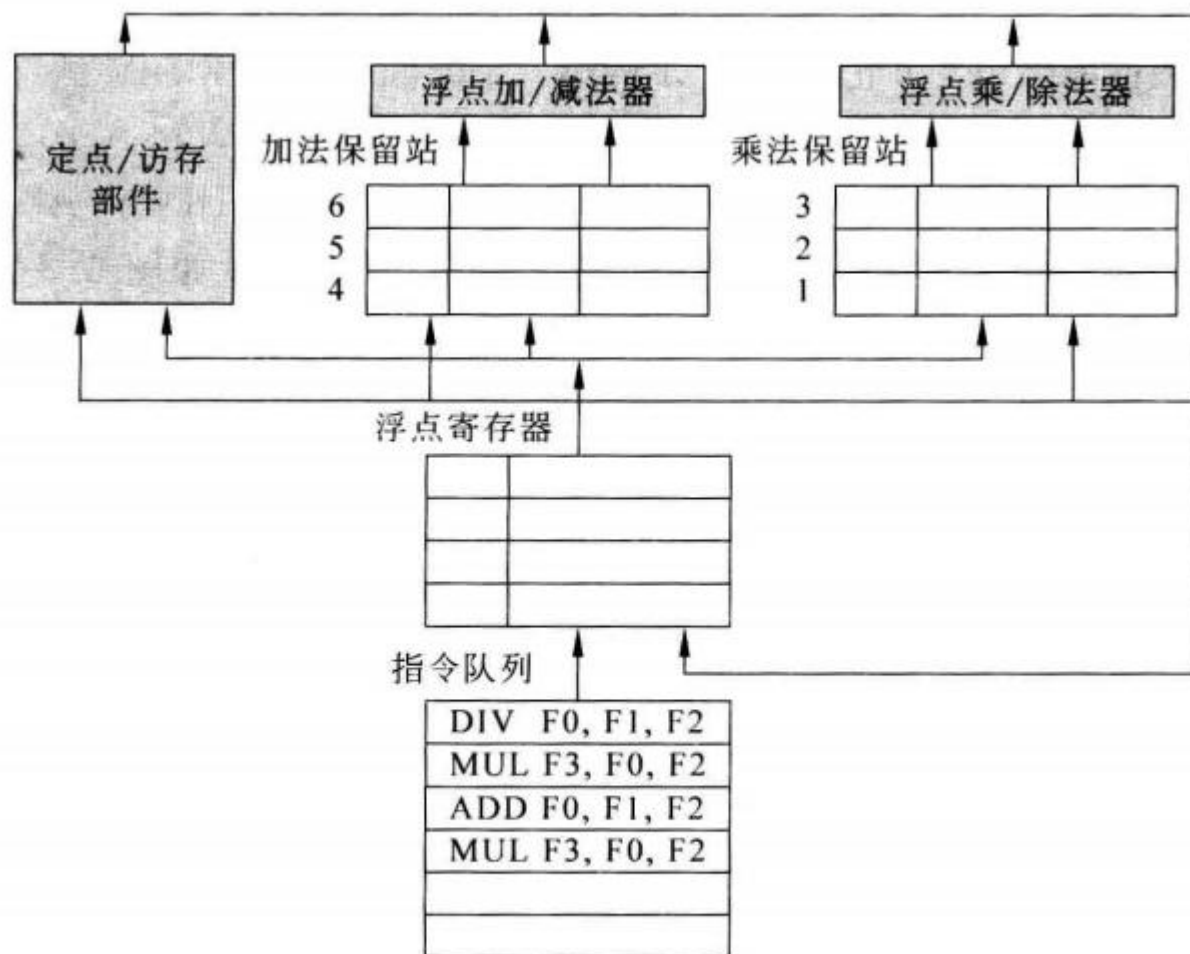


图 6.4 Tomasulo 结构

- 保留站，又称发射队列，基本功能是**暂存操作数未就绪的指令**，至少包括以下信息：
 - Busy：是否有指令正在等待
 - Op：操作码
 - Vj：第一个操作数的值
 - Vk：第二个操作数的值
 - Qj：第一个操作数是否就绪
 - Qk：第二个操作数是否就绪
- 每一个寄存器多一个状态位

2.2 Tomasulo算法详解1

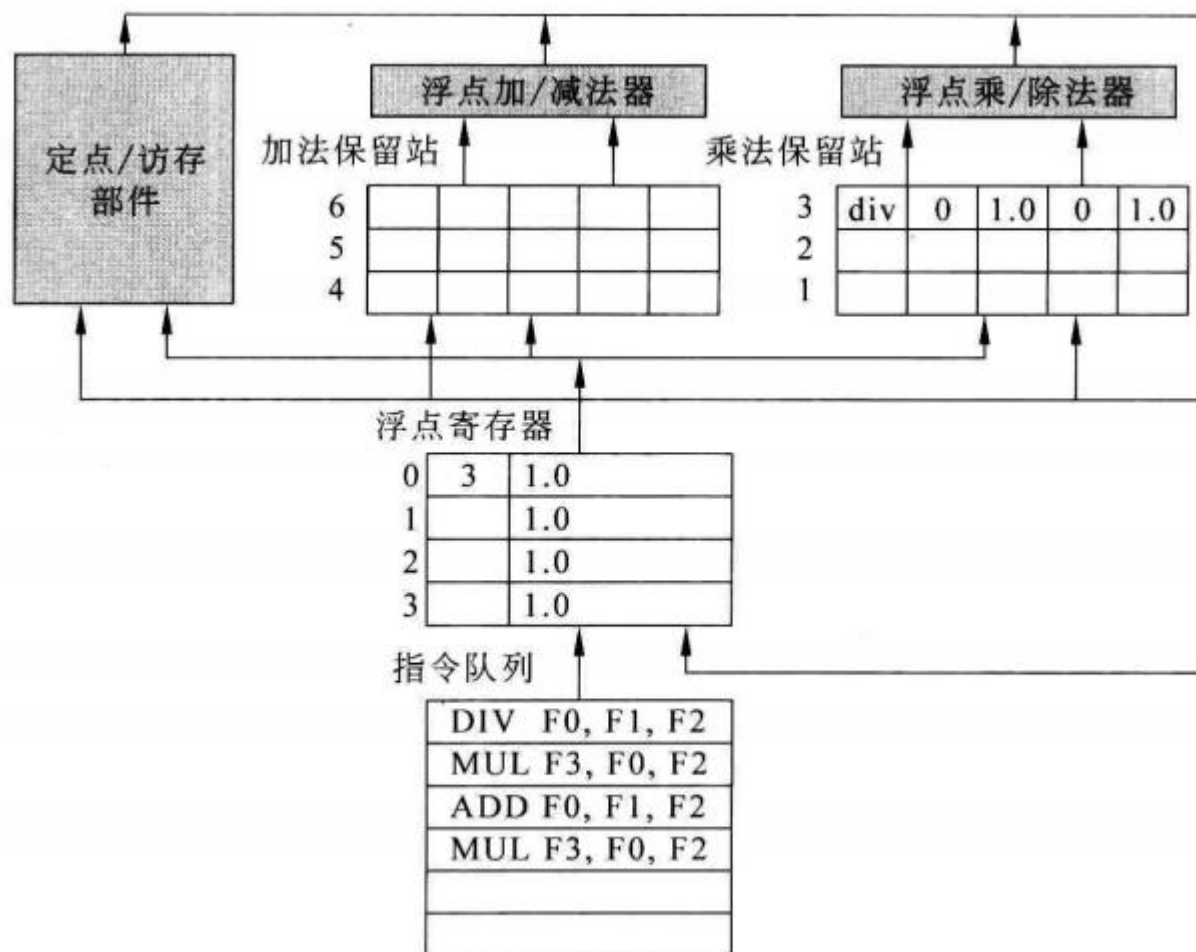


图 6.5 DIV 指令发射后的状态

- Q_j 为0, 表示对应操作数已经就绪
- Q_k 为0, 表示对应操作数已经就绪
- 寄存器F0的状态位为3
 - 表示正在等待保留站标号为3的指令的运算结果

2.2 Tomasulo算法详解2

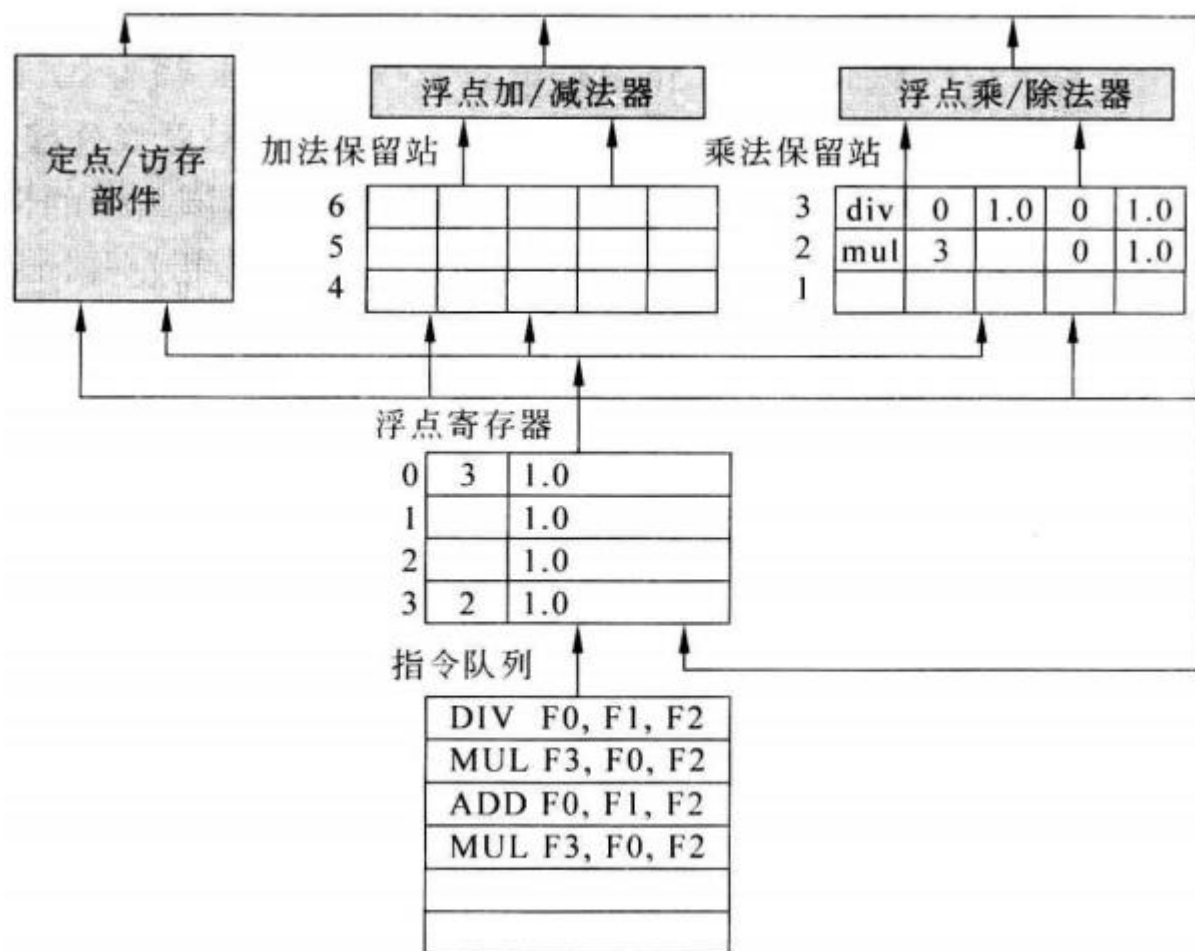


图 6.6 第一条 MUL 指令发射后的状态

- 第一条mul的Qj值为3, Qk值为0
 - 表示Vj正在等待保留站标号为3的指令的运算结果
 - 表示Vk已经准备就绪
- 寄存器F3的状态位为2
 - 表示正在等待保留站标号为2的指令的运算结果

2.2 Tomasulo算法详解3

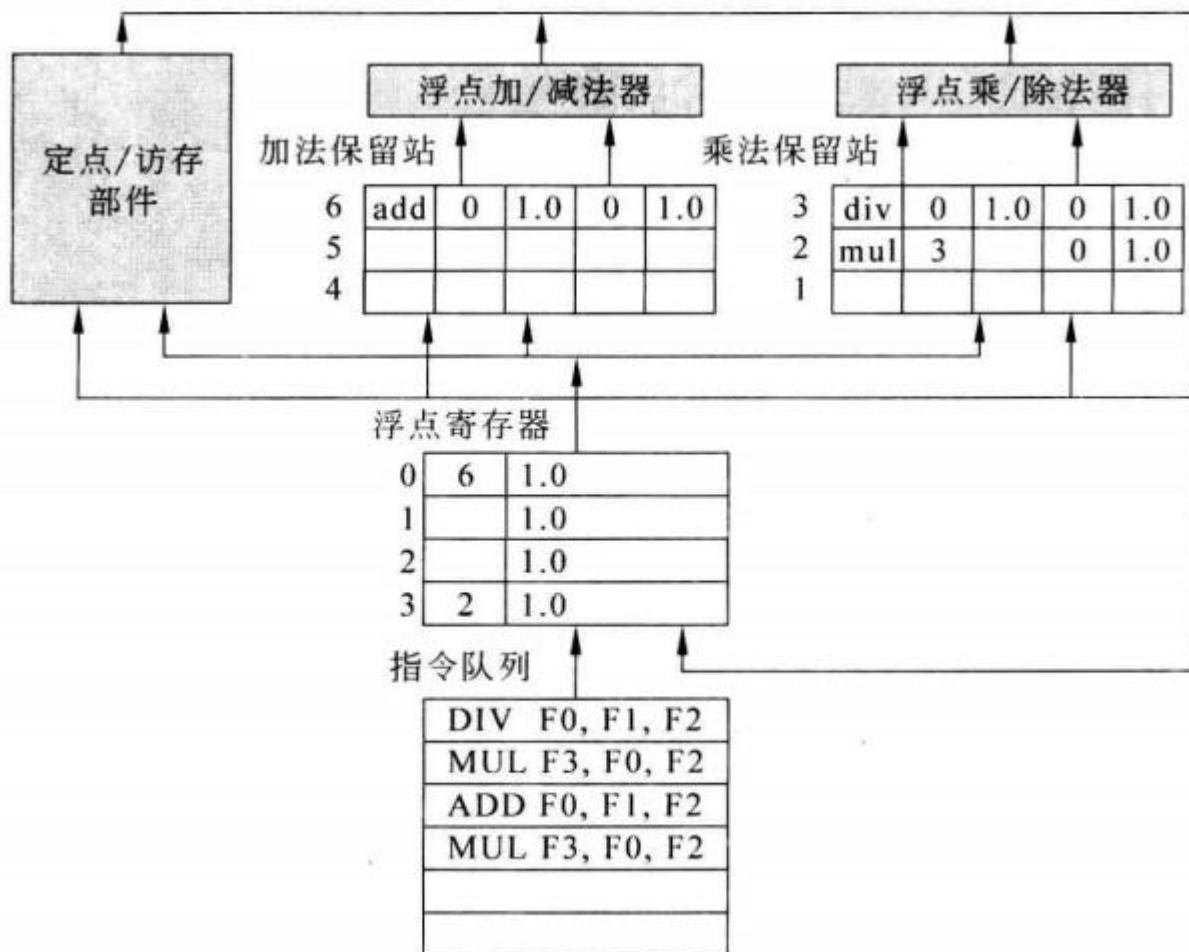


图 6.7 ADD 指令发射后的状态

- ADD指令的两个操作数都已经就绪
- 寄存器F0的状态位改为6
 - 表示正在等待保留站标号为2的指令的运算结果
 - 即可解决WAW相关
- 思考：WAR相关、RAW相关是否解决

2.2 Tomasulo算法详解4

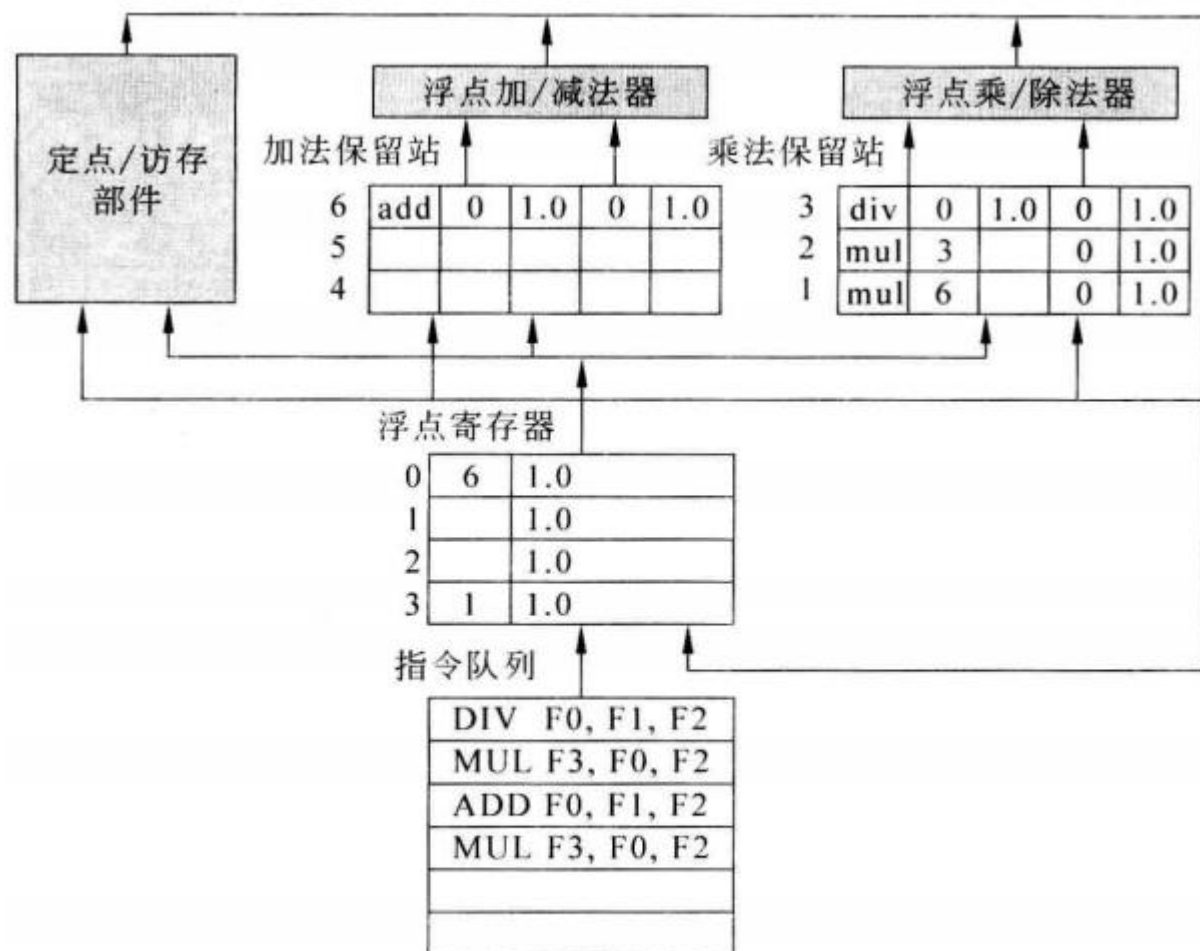


图 6.8 第 2 条 MUL 指令发射后的状态

- 第二条mul的Qj值为6，Qk值为0
 - 表示Vj正在等待保留站标号为6的指令的运算结果
 - 表示Vk已经准备就绪
- 寄存器F3的状态位为1
 - 表示正在等待保留站标号为1的指令的运算结果

2.2 Tomasulo算法详解5

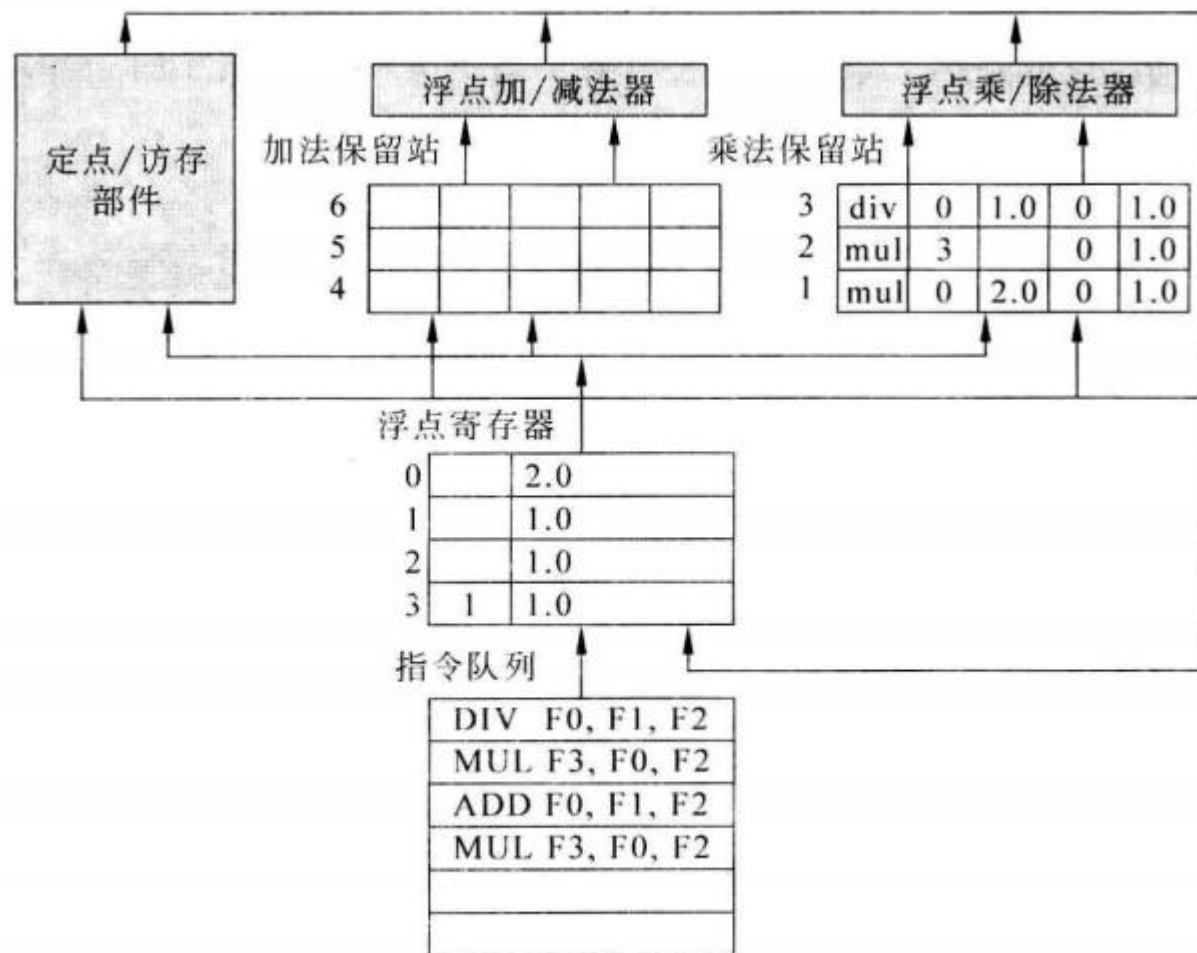


图 6.9 ADD 指令写回后的状态

- ADD指令写回，寄存器F0的状态位为空
 - 表示寄存器F0里的值是可用的新值
- 没有寄存器等待DIV指令的运算结果
 - 该指令不会再修改机器状态

2.2 Tomasulo算法详解6

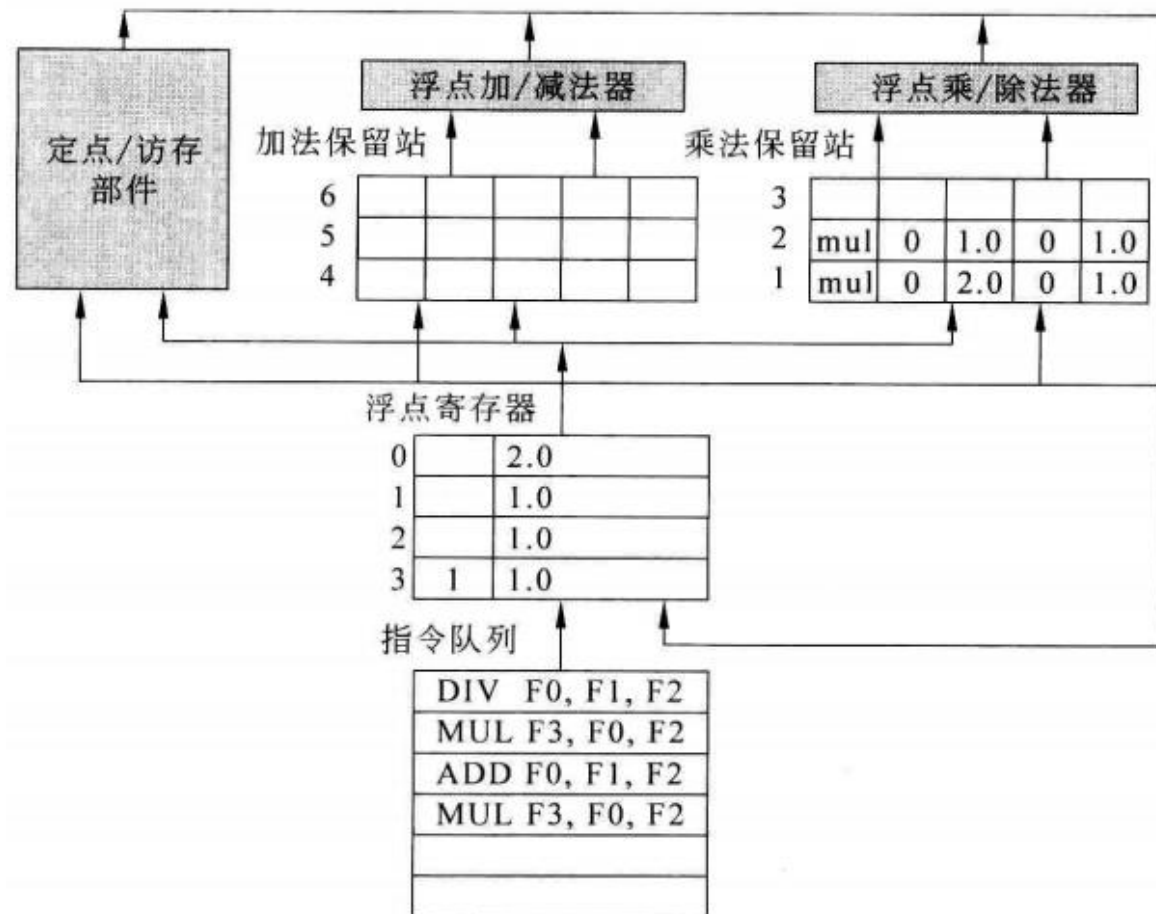


图 6.10 DIV 指令写回后的状态

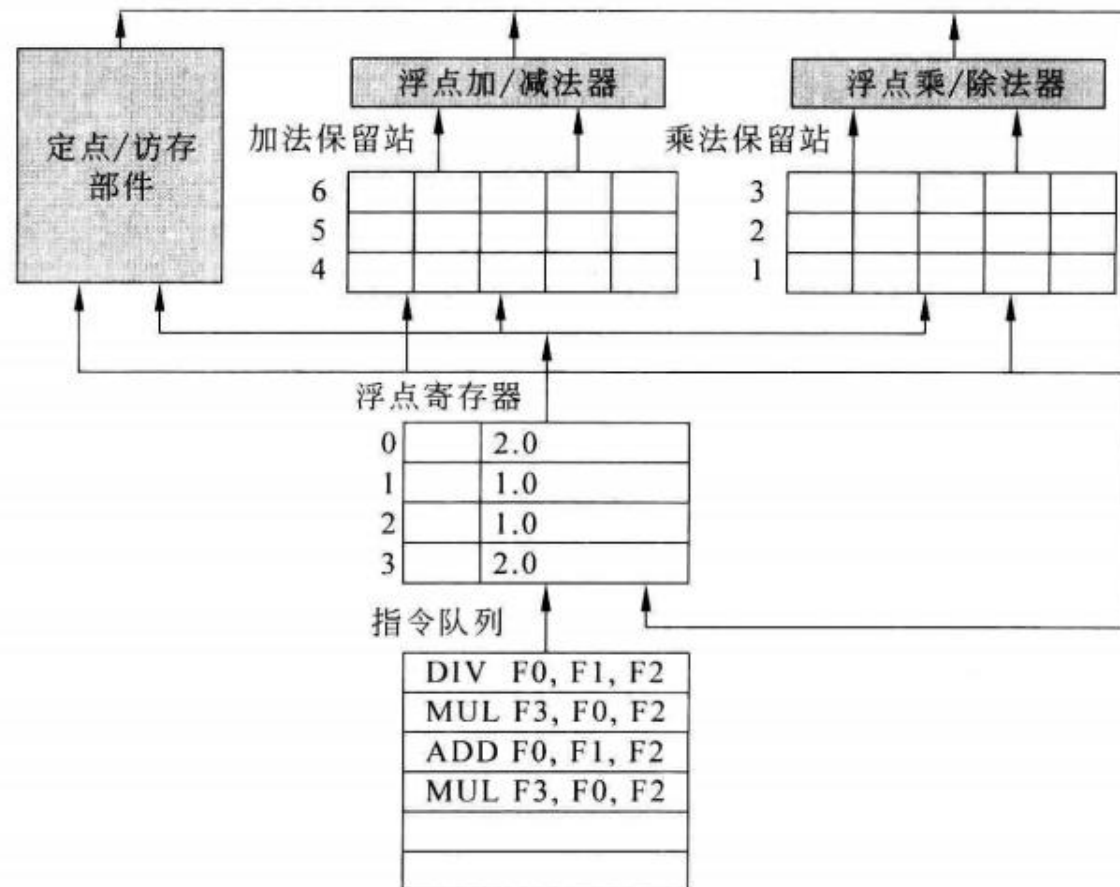
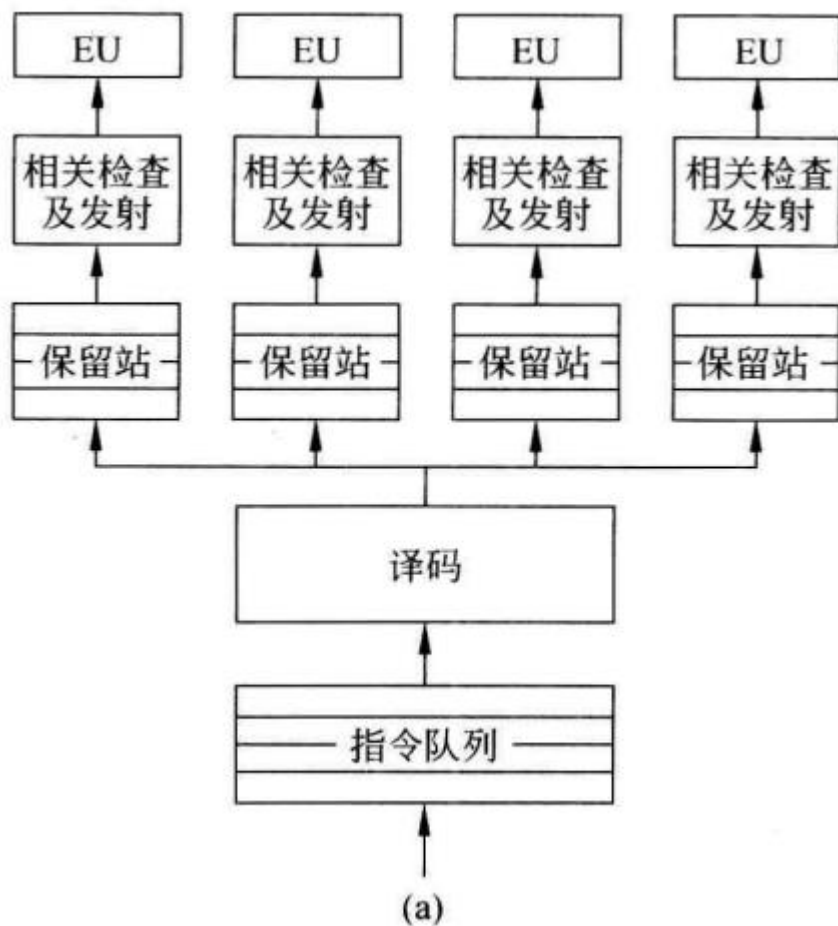


图 6.11 最后的状态

大纲：

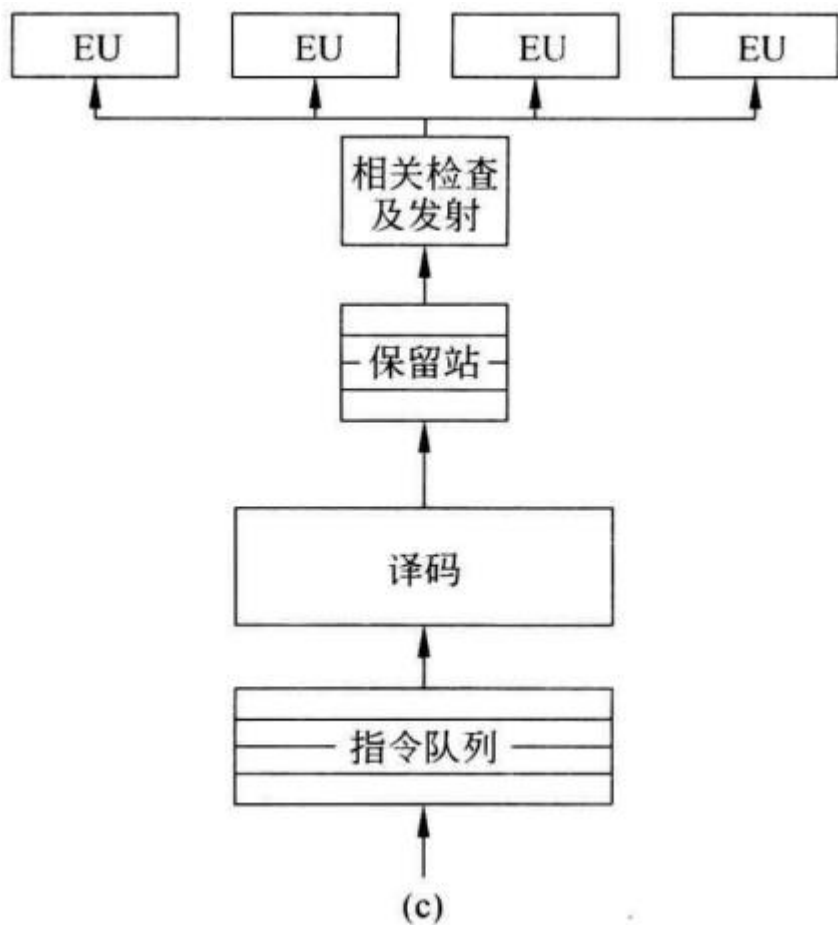
- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

2.3.1 独立保留站



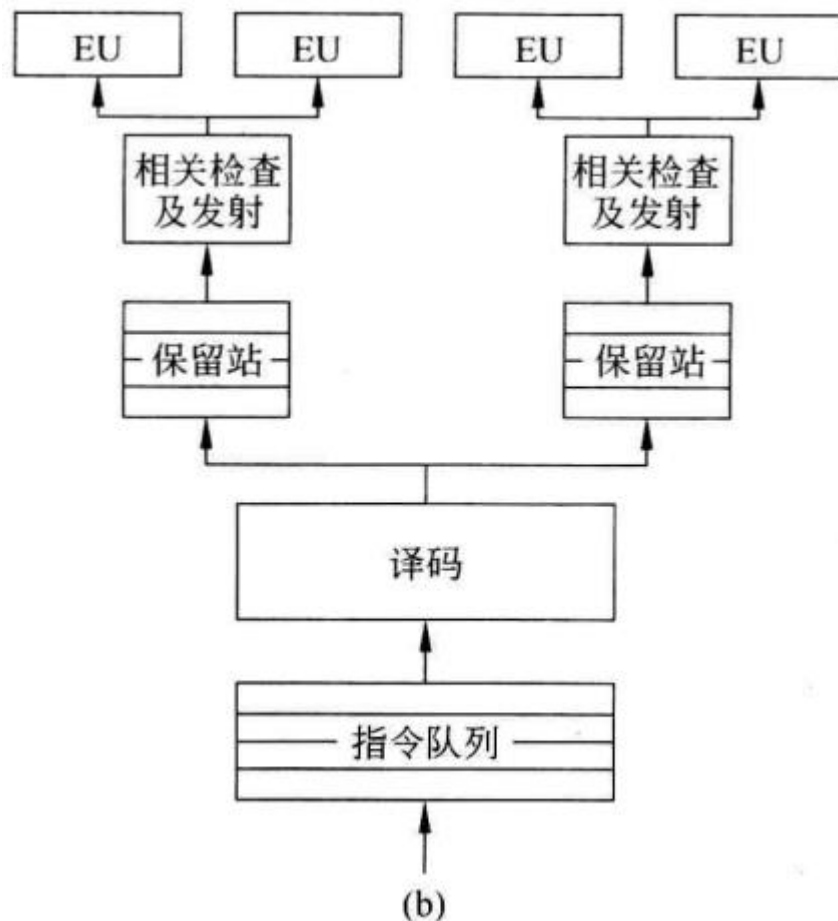
- 每一个功能模块配有一个独立保留站
- 每个保留站的结构较为简单，访问延迟小
- 利用率较低，有可能“忙的忙死，闲的闲死”
- 早期计算机使用较多
 - CDC 6600
 - PowerPC 620
 - 龙芯1号

2.3.2 全局保留站



- 所有功能模块共用一个保留站
- 利用率最高，同时读写端口的延迟也最大
- 似乎是Intel的偏爱：
 - Pentium Pro / II / III
 - Pentium M
 - Core

2.3.3 分组保留站



- 功能模块分组，同组功能模块共用一个保留站
- 相较于独立保留站，访问延迟较大，利用率较高
- 分组策略，见仁见智
 - MIPS R10000 定点、浮点、访存各16项
 - Alpha 21264 定点/访存 20项，浮点14项
 - PA 8700 定点/浮点 28项，访存28项
 - 龙芯2号 定点/访存 16项，浮点16项

2.3.4 保留站与寄存器文件

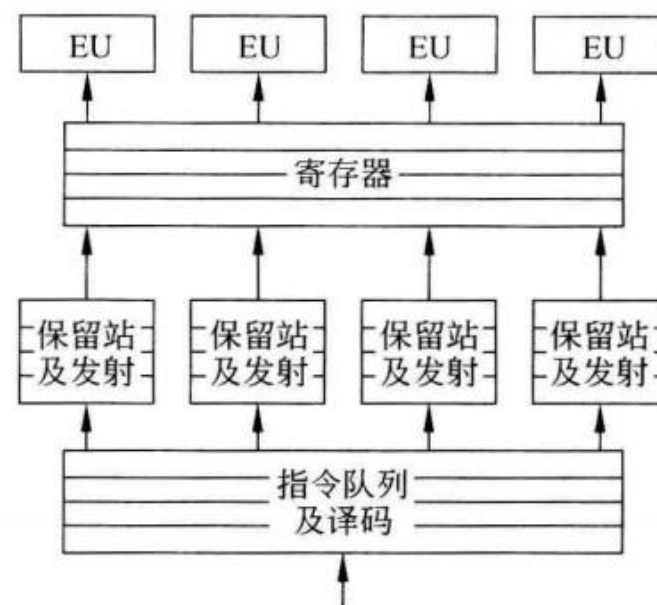
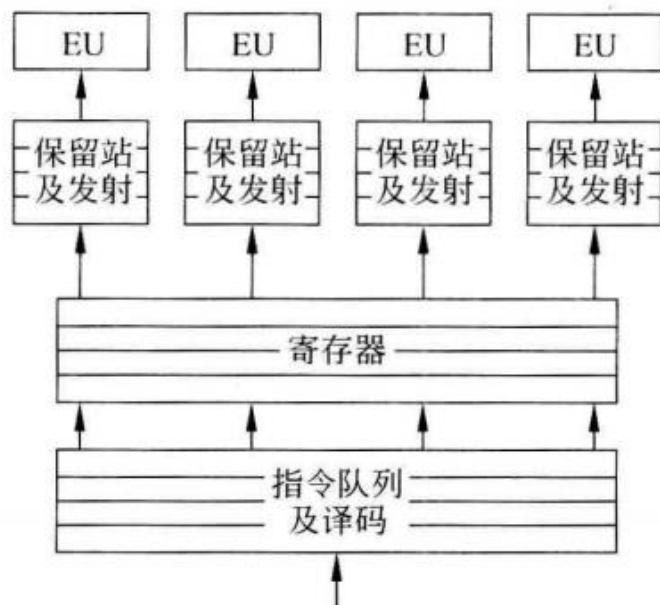


图 7.3 保留站和寄存器的关系

- 保留站前读寄存器
 - Tomasulo算法示例
 - 需要保存寄存器的值和就绪标记位
 - 操作数未就绪的指令需要侦听结果总线

- 保留站后读寄存器
 - 每次读取寄存器时，寄存器的值都是有效的
 - 不需要保留寄存器的值

2.3.5 回顾参考图

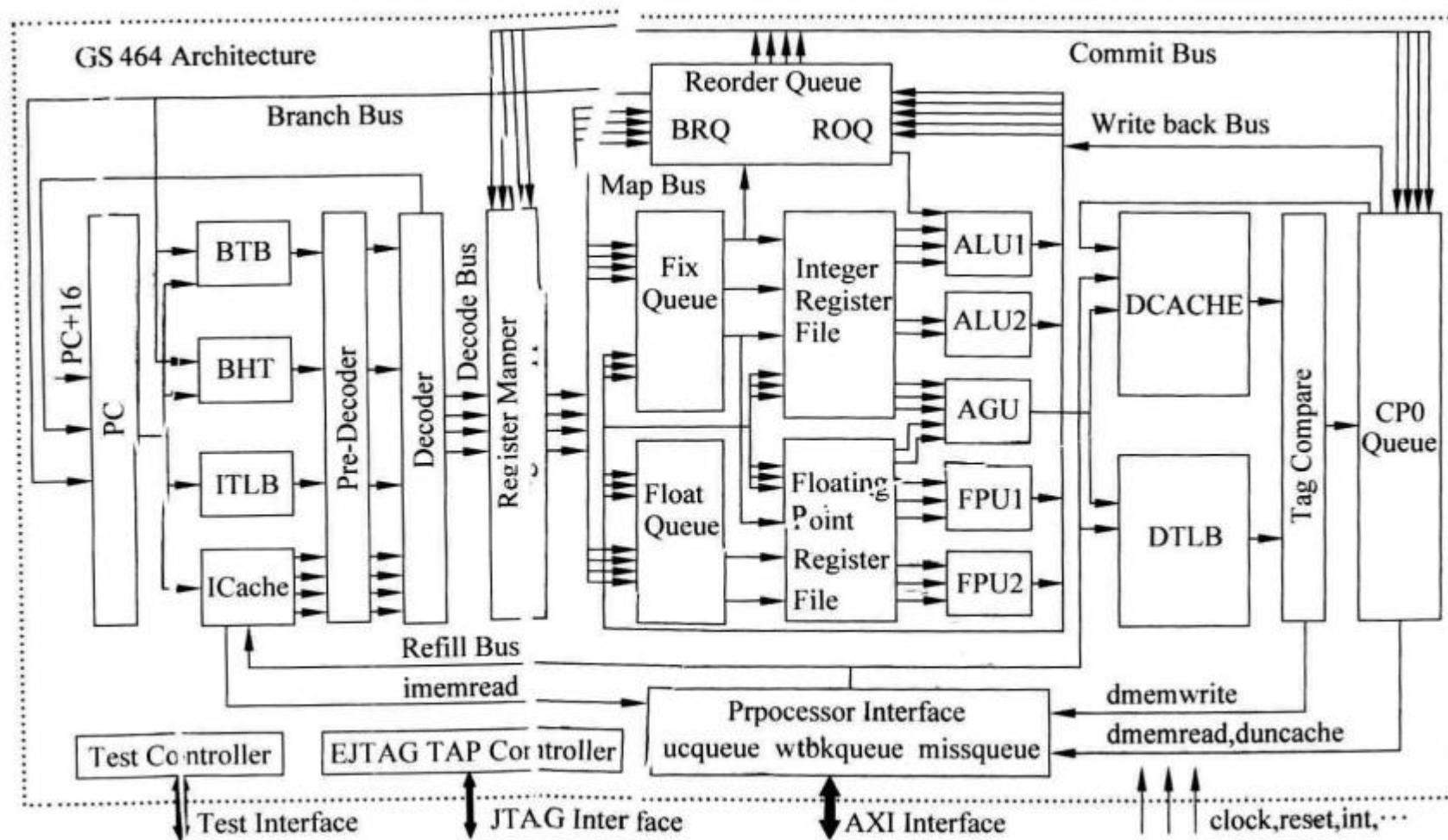


图 7.9 龙芯 2 号处理器结构框图

- 关注模块:
 - Fix Queue
 - Reorder Queue
 - Register Mapper

大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

2.4.1 精确异常

- 精确异常
 - 操作系统期望处理器能够提供精确例外，给软件一个干净的现场
 - 要求异常指令之前的指令都已经执行完毕
 - 要求异常指令之后的指令都没有执行
- 乱序流水时
 - 指令的机器状态的修改延迟到前面的指令都已经执行完毕
 - 增加一个流水级，提交阶段 (commit)
 - 增加一个结构重排序缓存 (ROB, ReOrder Buffer)

2.4.2 重排序缓存0

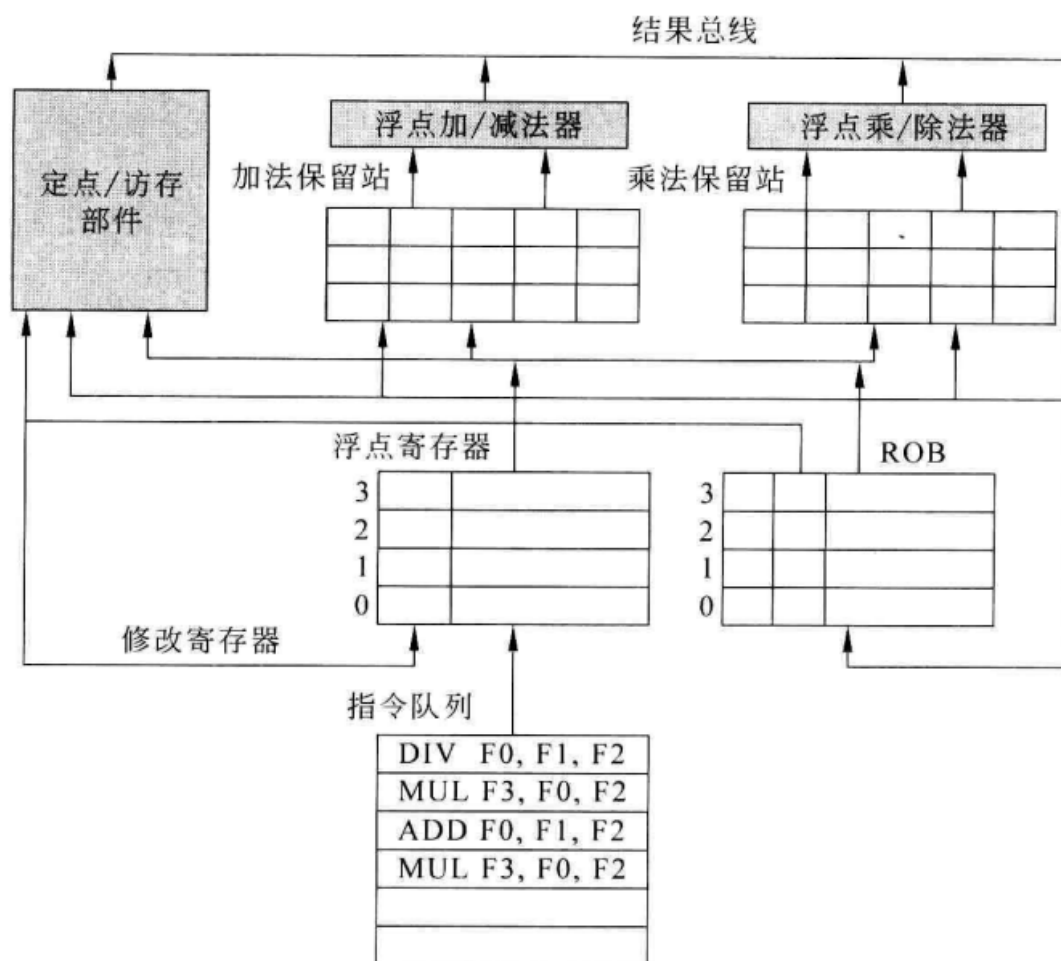


图 6.13 增加 ROB 后的动态流水线结构

- 保留站的标号可以取消，但是需要保存ROB的索引号
- ROB至少包括以下信息
 - 指令标识（操作码）
 - 写回的寄存器号（地址）
 - 写回的数据

2.4.2 重排序缓存1

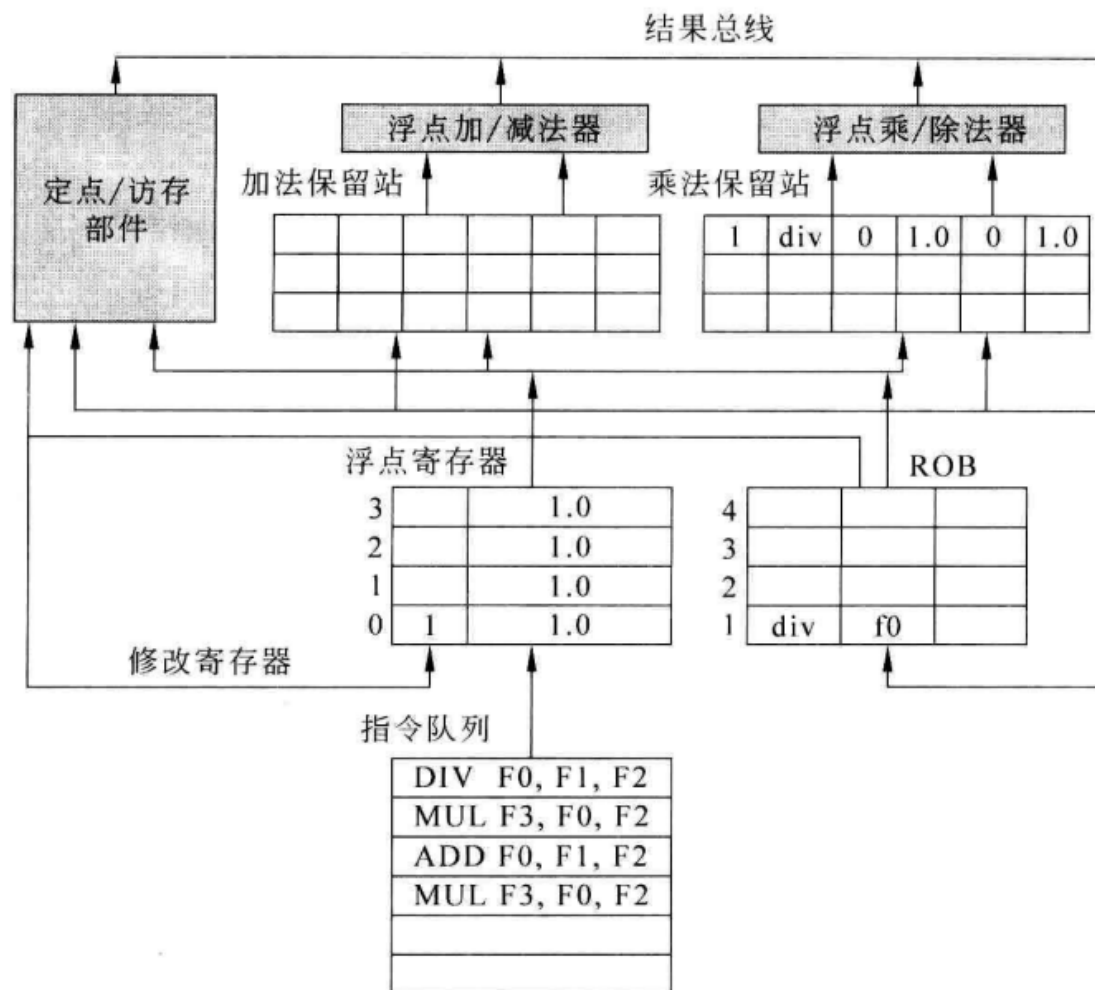


图 6.14 DIV 指令发射后的流水线状态

- 发射第一条指令，同时将对应的信息放进ROB中
- 此时保留站需要保存DIV指令的ROB索引，即1
- 寄存器F0的状态位设为1

2.4.2 重排序缓存2

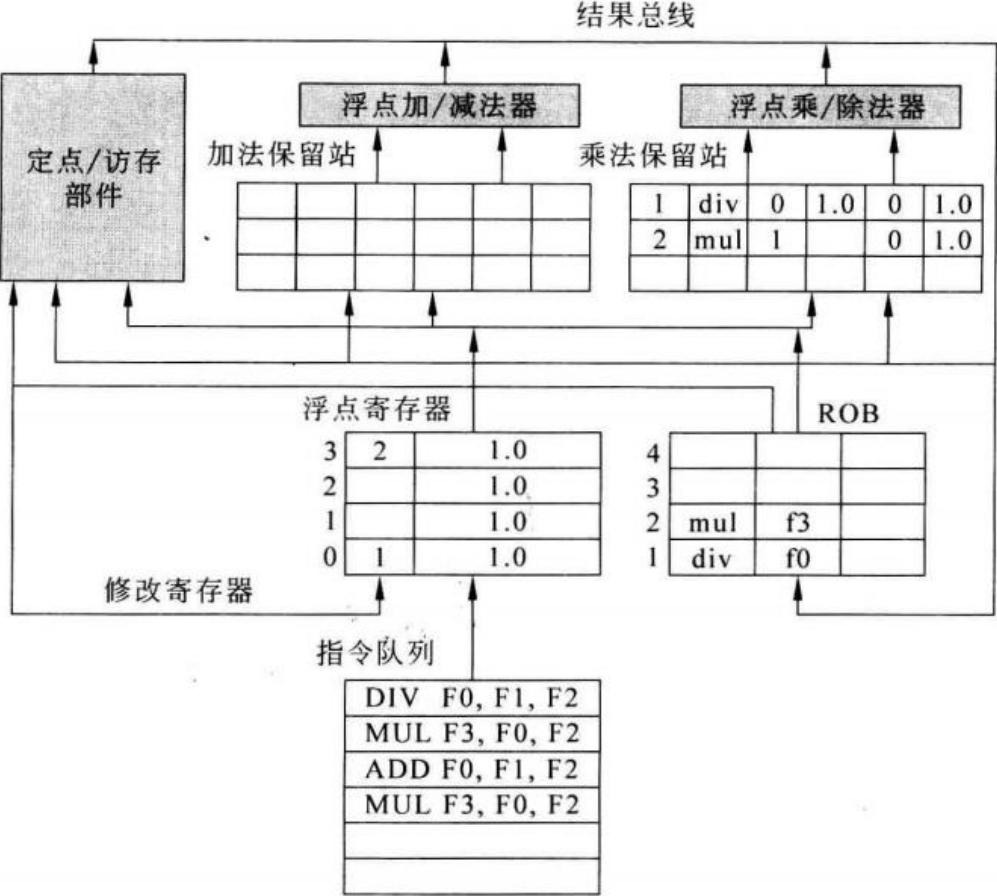


图 6.15 第 1 条 MUL 指令发射后的流水线状态

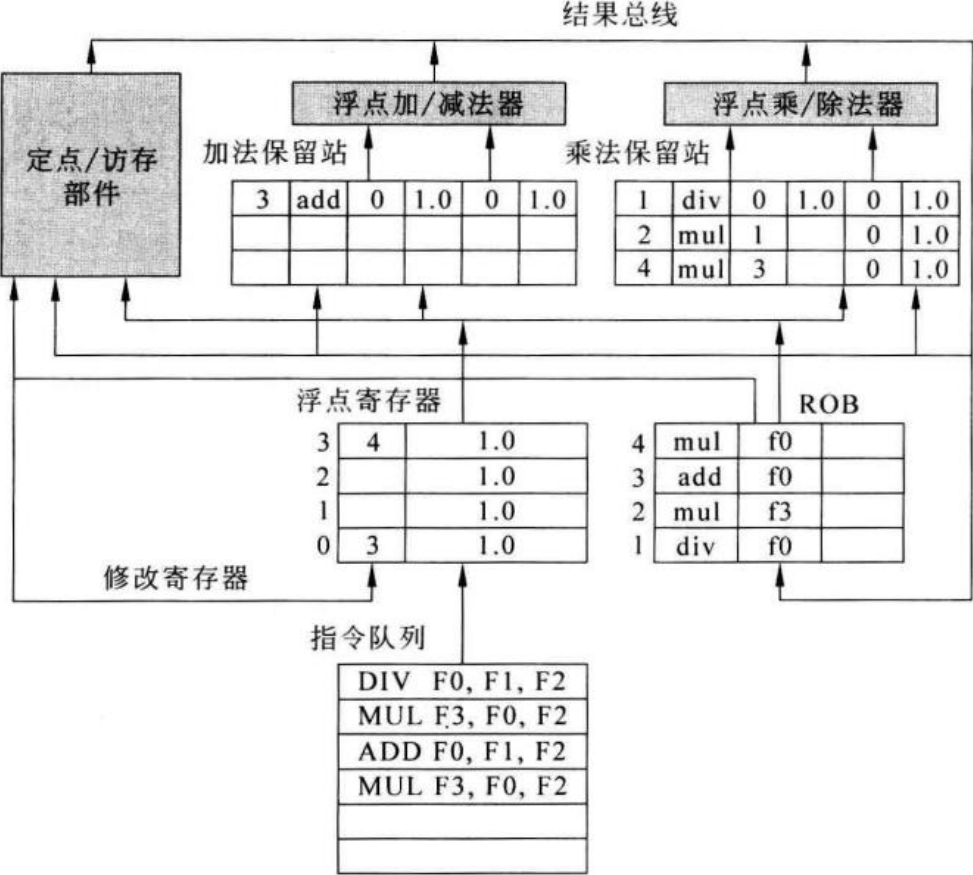


图 6.17 第 2 条 MUL 指令发射后的流水线状态

2.4.2 重排序缓存3

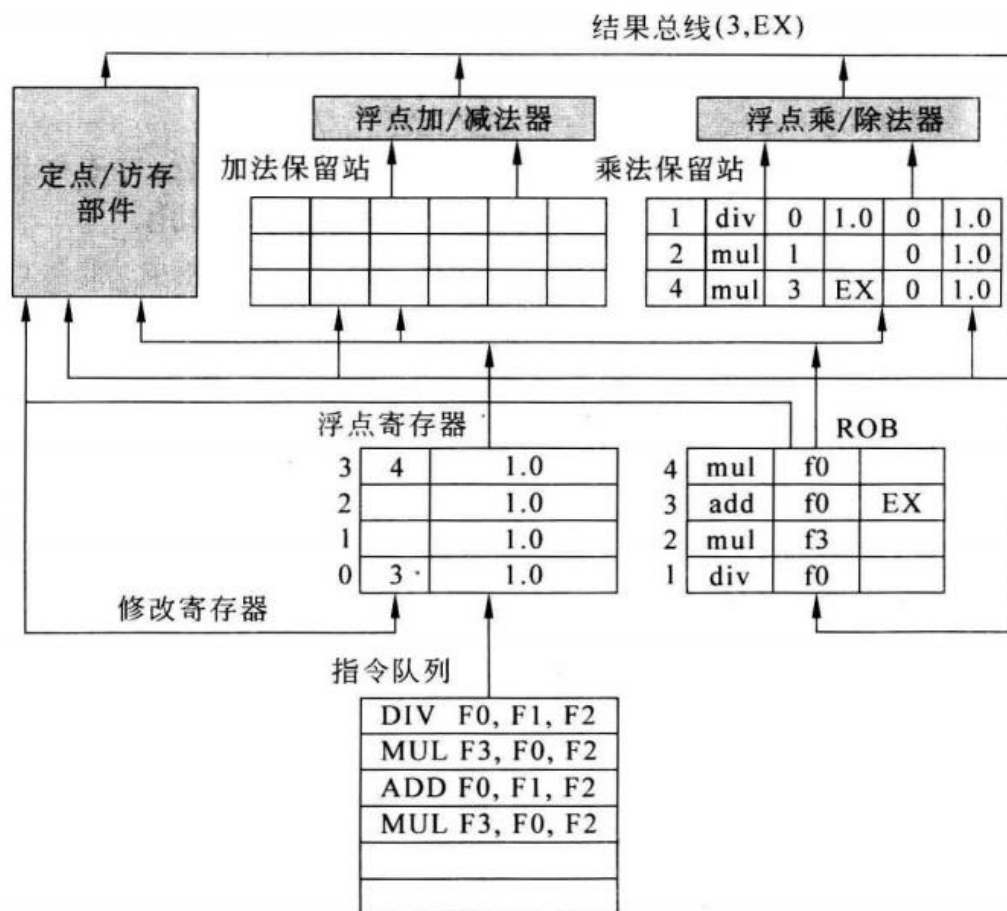


图 6.23 ADD 发生例外写回后的流水线状态

- 假如ADD指令发生加法溢出例外*
 - 将例外情况保存在ROB中，包括例外编码、EPC等
- DIV、MUL指令正常提交，修改机器状态
- 轮到ADD指令提交时，处理器响应例外
 - 跳转到例外处理程序入口
 - 修改对应CSR
 - 清空流水线
- *保证例外指令之前的指令都执行完毕，例外指令之后的指令都没有执行，即精确异常

2.4.3 回顾参考图

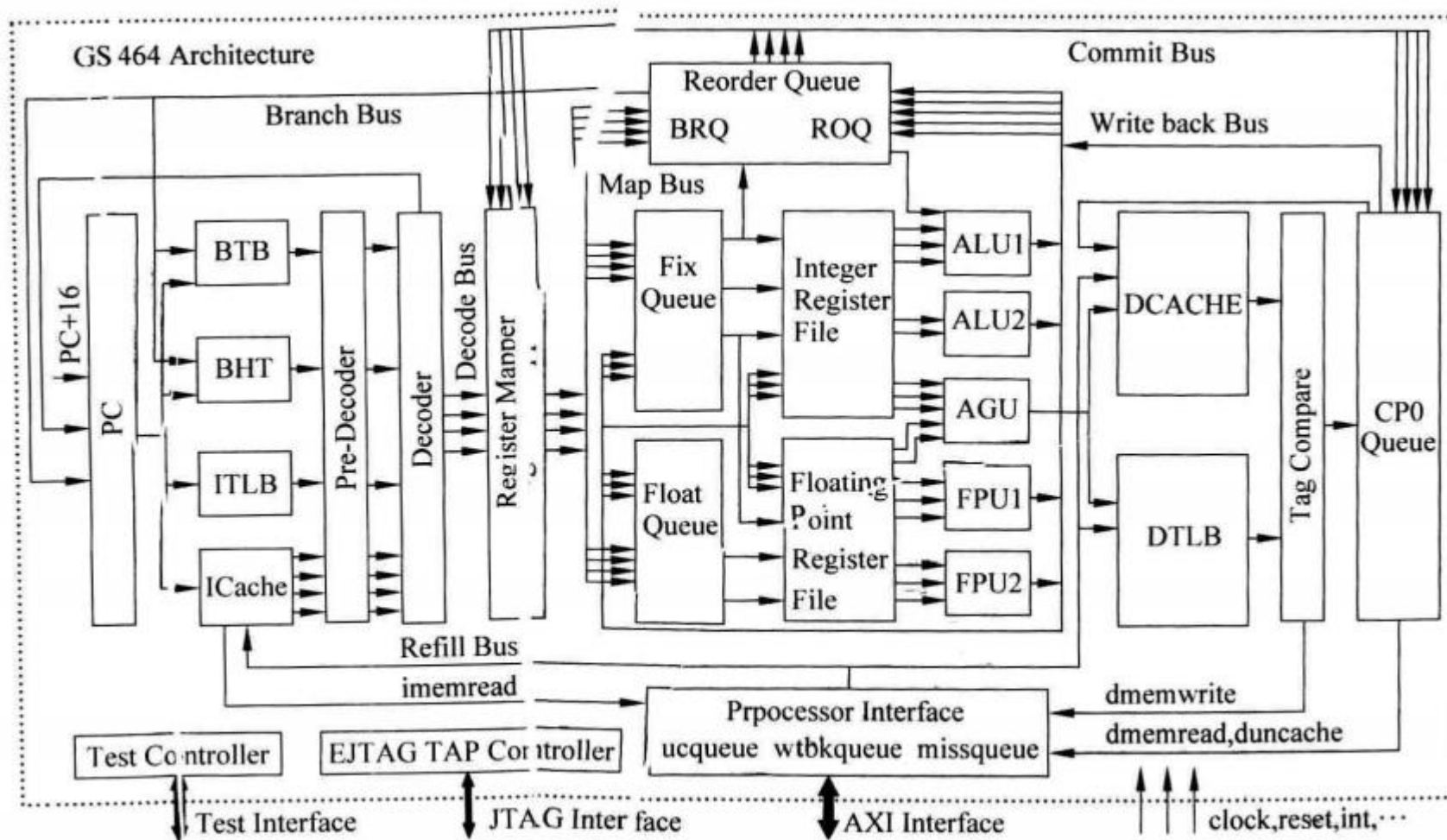


图 7.9 龙芯 2 号处理器结构框图

- 关注模块:
 - Fix Queue
 - Reorder Queue
 - Register Mapper

大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

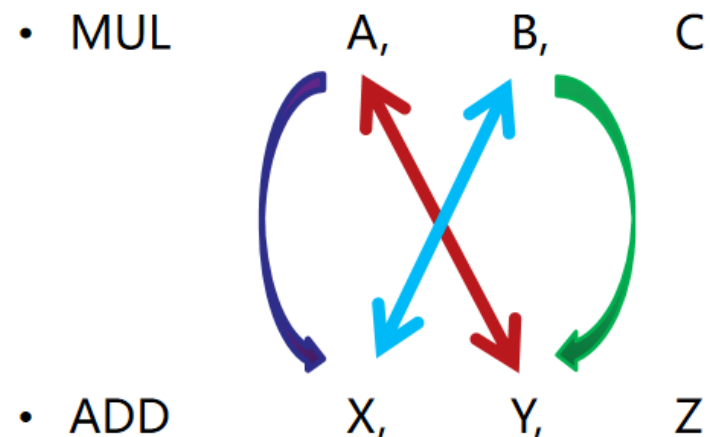
2.5.1 寄存器重命名概述

- 目的：消除指令间WAW、WAR相关

- 示例代码：

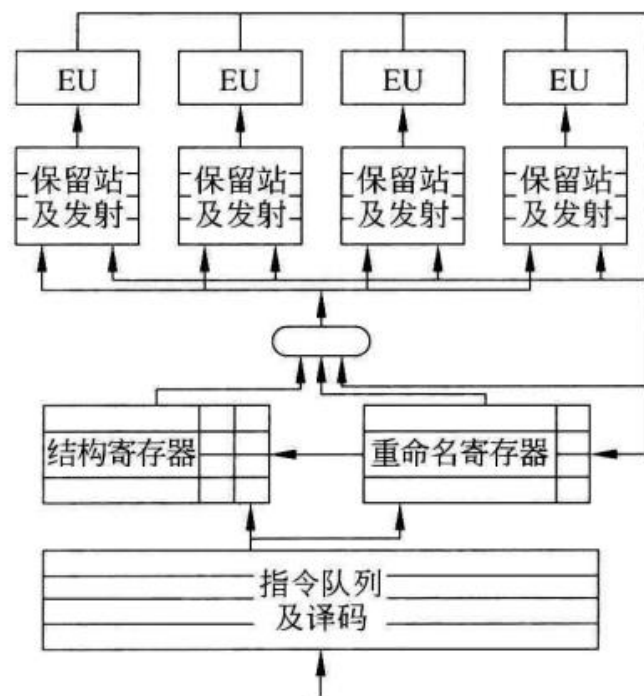
• DIV	R1,	R2,	R3
• ADD	R4,	R1,	R5
• ADD	R5,	R6,	R7
• ADD	R1,	R8,	R9

• DIV	P32,	P2,	P3
• ADD	P33,	P32,	P5
• ADD	P34,	P6,	P7
• ADD	P35,	P8,	P9

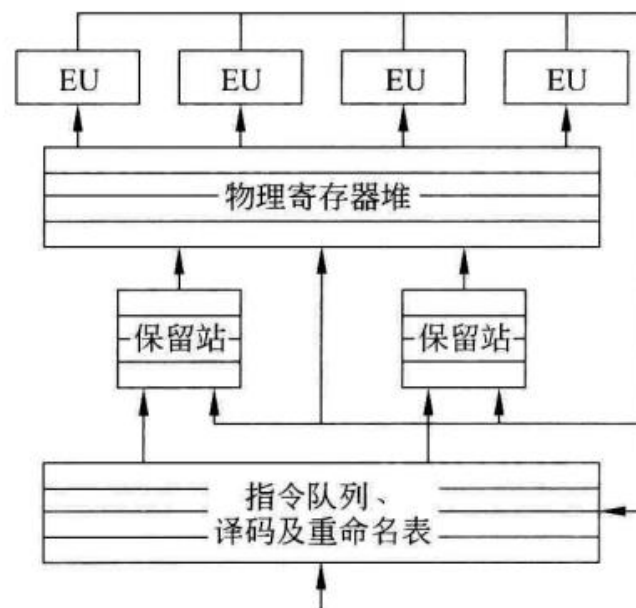


▲ 逻辑寄存器, R1, R2, R3
▲ 物理寄存器, P1, P2, P3

2.5.2 两类技术路线



- 独立重命名寄存器
 - 指令的运算结果只写到重命名寄存器，等到指令提交时才写回结构寄存器
 - 有的专设重命名寄存器文件
 - 有的重命名到ROB



- 物理寄存器堆
 - 将重命名寄存器与结构寄存器合并，组成物理寄存器堆
 - 以重命名表为核心构造映射逻辑

2.5.3 物理寄存器的分配0

- 重命名表, Rename Table
 - 32项, 用逻辑寄存器号索引
 - 表示当前的逻辑寄存器对应的物理寄存器
- 寄存器状态表, RegState Table
 - 64项, 用物理寄存器号索引
 - 表示当前的物理寄存器的状态
 - 11, Assigned状态, 即已经与某逻辑寄存器绑定
 - 00, Free状态, 即可以分配给其他逻辑寄存器

Rename Table

0	P0
1	P1
2	P2
3	P3
...	...
31	P31
32	P32

RegState Table

0	11
1	11
2	11
3	11
...	...
30	11
31	11
32	00
33	00
34	00
...	...
61	00
62	00
63	00

2.5.3 物理寄存器的分配1

Rename Table

0	P0
1	P32
2	P2
3	P3
...	...
31	P31
32	P32

RegState Table

0	11
1	11
2	11
3	11
...	...
30	11
31	11
32	01
33	00
34	00
...	...
61	00
62	00
63	00



DIV R1, R2, R3

- ADD R4, R1, R5
- ADD R5, R6, R7
- ADD R1, R8, R9

DIV P32, P2, P3

Rename Table

0	P0
1	P32
...	...
4	P33
...	...
31	P31
32	P32

RegState Table

0	11
1	11
2	11
3	11
...	...
30	11
31	11
32	01
33	01
34	00
...	...
61	00
62	00
63	00



DIV R1, R2, R3



ADD R4, R1, R5

- ADD R5, R6, R7
- ADD R1, R8, R9

DIV P32, P2, P3

ADD P33, P32, P5




2.5.3 物理寄存器的分配2

Rename Table

0	P0
1	P32
...	...
4	P33
5	P34
...	...
31	P31
32	P32

RegState Table

0	11
1	11
2	11
3	11
...	...
30	11
31	11
32	01
33	01
34	01
...	...
61	00
62	00
63	00

 DIV R1, R2, R3 DIV P32, P2, P3
 ADD R4, R1, R5 ADD P33, P32, P5
 ADD R5, R6, R7 ADD P34, P6, P7
• ADD R1, R8, R9

▲ 重命名策略小结:

- 每一个目的寄存器，都分配新的物理寄存器，自然就消除了WAW和WAR
- 每一个源寄存器，都重命名到最近一次分配给它的物理寄存器，保证RAW不会出错

2.5.4 物理寄存器的例外处理与释放0

Rename Table

0	P0
1	P32
...	...
4	P33
5	P34
...	...
31	P31
32	P32

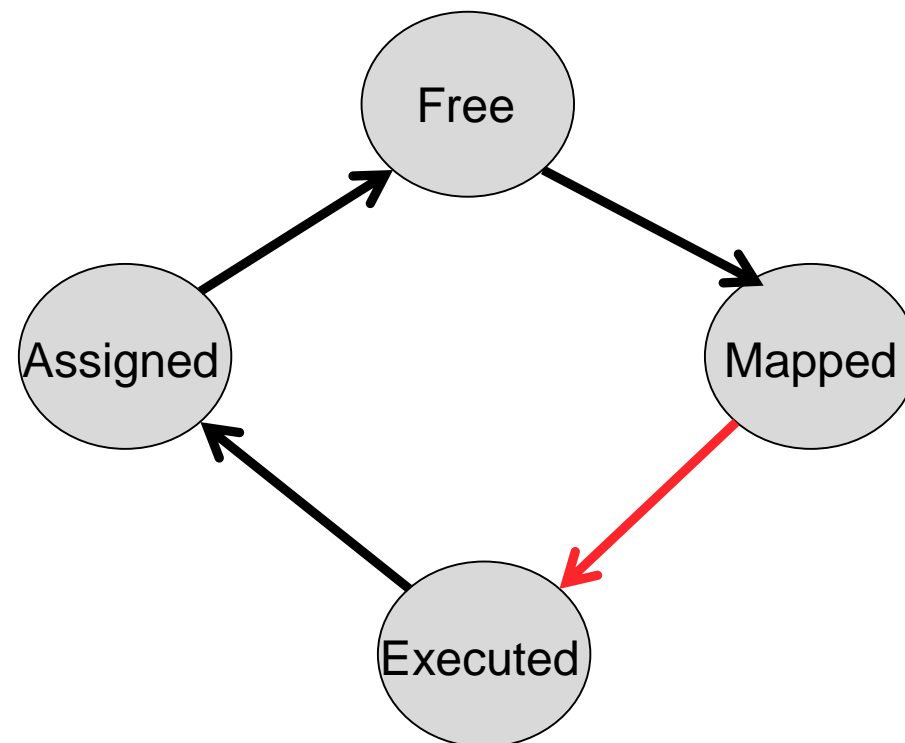
RegState Table

0	11
1	11
2	11
3	11
...	...
30	11
31	11
32	10
33	01
34	01
...	...
61	00
62	00
63	00

- 物理寄存器状态

- 00, Free状态, 即非分配给任何逻辑寄存器
- 01, Mapped状态, 即已经分配出去, 尚未完成运算
- 10, Executed状态, 即已经完成运算, 尚未提交
- 11, Assigned状态, 即已经提交, 与某一个逻辑寄存器绑定

DIV指令完成运算



DIV R1, R2, R3

ADD R4, R1, R5

ADD R5, R6, R7

- ADD R1, R8, R9

DIV P32, P2, P3

ADD P33, P32, P5

ADD P34, P6, P7

2.5.4 物理寄存器的例外与释放1

Rename Table

0	P0
1	P32
...	...
4	P33
5	P34
...	...
31	P31
32	P32

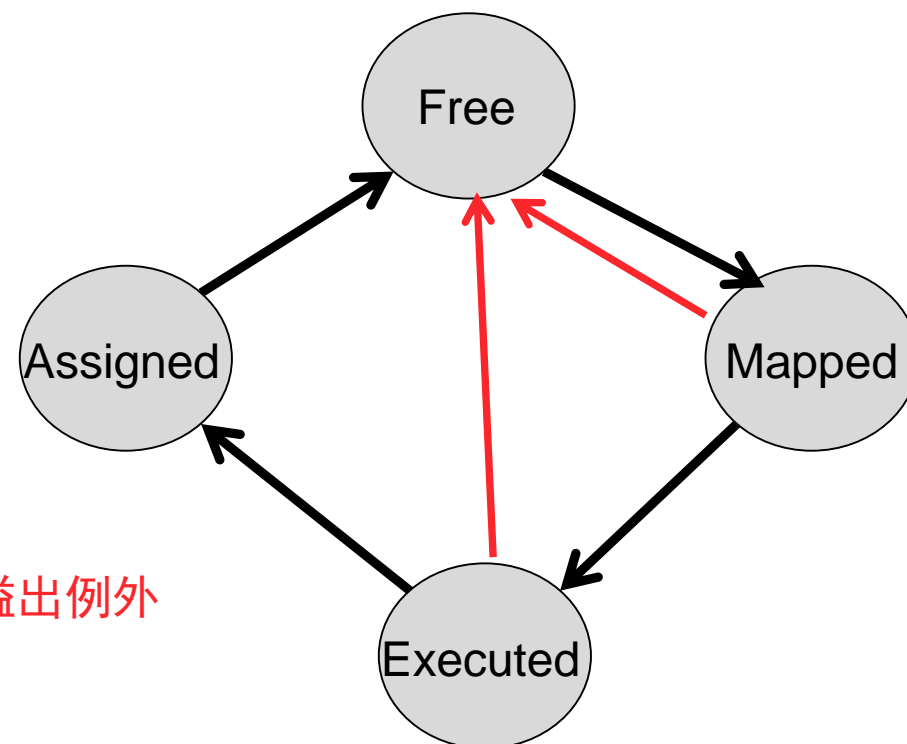
RegState Table

0	11
1	00
2	11
3	11
...	...
30	11
31	11
32	11
33	00
34	00
...	...
61	00
62	00
63	00

- DIV R1, R2, R3
- **ADD R4, R1, R5**
- ADD R5, R6, R7
- ADD R1, R8, R9

DIV P32, P2, P3
ADD P33, P32, P5
ADD P34, P6, P7

ADD指令加法溢出例外



▲ 思考：例外发生时，重命名表的映射该如何取消，保证例外指令之后的指令都没有重命名

2.5.4物理寄存器的例外与释放2

Rename Table

0	P0	P0
1	P1	P1
2	P2	P2
3	P3	P3
4	P4	P4
5	P5	P5
...
31	P31	P31
32	P32	P32

- 新设计的重命名表，多出一个数据，表示以前分配的、处于稳定状态下的物理寄存器号

Rename Table

0	P0	P0
1	P32	P1
2	P2	P2
3	P33	P3
4	P34	P4
5	P5	P5
...
31	P31	P31
32	P32	P32

完成3条指令的重命名

- DIV R1, R2, R3 DIV P32, P2, P3
- ADD R4, R1, R5 ADD P33, P32, P5
- ADD R5, R6, R7 ADD P34, P6, P7
- ADD R1, R8, R9

2.5.4 物理寄存器的例外与释放3

Rename Table

0	P0	P0
1	P32	P32
2	P2	P2
3	P3	P3
4	P33	P4
5	P34	P5
...
31	P31	P31
32	P32	P32

DIV指令提交

- DIV R1, R2, R3
 - ADD R4, R1, R5
 - ADD R5, R6, R7
 - ADD R1, R8, R9
- DIV P32, P2, P3
ADD P33, P32, P5
ADD P34, P6, P7

Rename Table

0	P0	P0
1	P32	P32
2	P2	P2
3	P3	P3
4	P4	P4
5	P5	P5
...
31	P31	P31
32	P32	P32

ADD指令加法溢出例外
重命名表撤回映射

- DIV R1, R2, R3
 - ADD R4, R1, R5
 - ADD R5, R6, R7
 - ADD R1, R8, R9
- DIV P32, P2, P3
ADD P33, P32, P5
ADD P34, P6, P7

2.5.5 寄存器重命名小结

- 寄存器重命名技术的目的是，消除指令间的WAW，WAR相关，更彻底地实现乱序
- 寄存器重命名技术大致分两种技术路线：独立重命名寄存器、物理寄存器堆
- 物理寄存器重命名的工程重点是两个关键模块：Rename Table、RegState Table
- 物理寄存器的分配、释放和例外处理：
 - 分配策略是，**每一个目的寄存器**，都分配新的物理寄存器，自然就消除了WAW和WAR，**每一个源寄存器**，都重命名到最近一次分配给它的物理寄存器，保证RAW不会出错。
 - 释放寄存器的逻辑是，一条指令提交，就绑定(11)当前分配的物理寄存器，释放(00)上一次分配到的物理寄存器。
 - 例外处理要求取消例外指令，及其之后指令的所有寄存器映射关系。RegState Table中，不是Assigned状态的寄存器全部改成Free状态，也要求在Rename Table带有稳定状态下的物理寄存器号，可供撤销。

2.5.6 回顾参考图

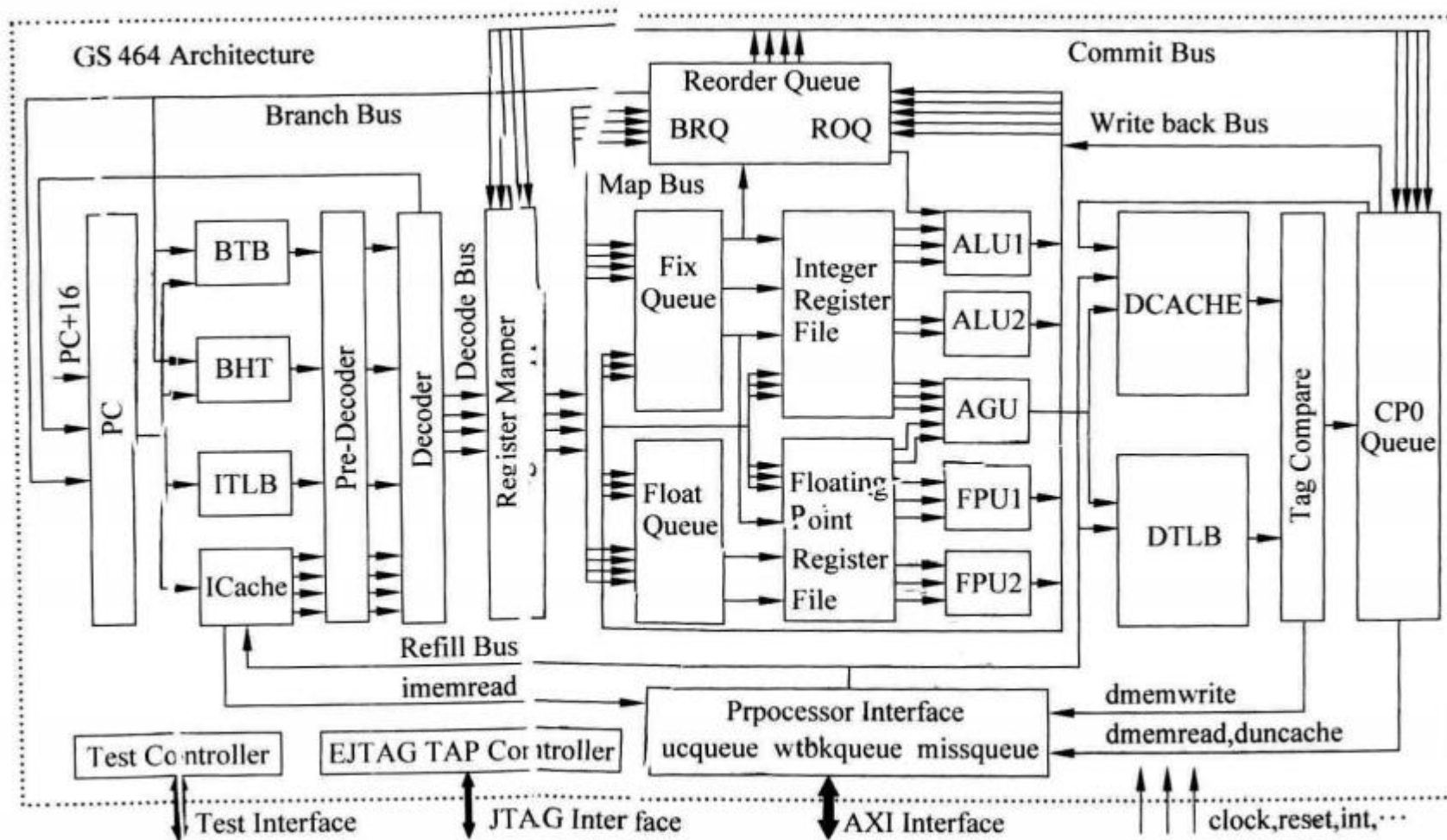


图 7.9 龙芯 2 号处理器结构框图

- 关注模块：
 - Fix Queue
 - Reorder Queue
 - Register Mapper

大纲：

- 经典五级流水线
 - 流水线基础
 - 数据冒险
 - 控制冒险
- 乱序多发流水线原理介绍
 - 乱序技术概述
 - Tomasulo算法
 - 保留站的组织形式
 - ROB与精确异常
 - 寄存器重命名技术
 - 乱序多发tips分享*

2.6 乱序多发tips分享*

- 读端口与cluster
- 预测读
- StoreBuffer hit
- 长流水与分支预测
- ROB回滚恢复

参考文献：

- 《计算机原理与设计》，李亚民著，清华大学出版社
- 《计算机体系结构》，胡伟武，陈云霁，肖俊华，章隆兵
- 《计算机体系结构：量化研究方法》（第5版），John L. Hennessy , David A. Patterson
- 《计算机组成原理与设计：硬件/软件接口》（第5版），John L. Hennessy , David A. Patterson

谢谢！