流水线技术概述

2020年秋

本讲概要

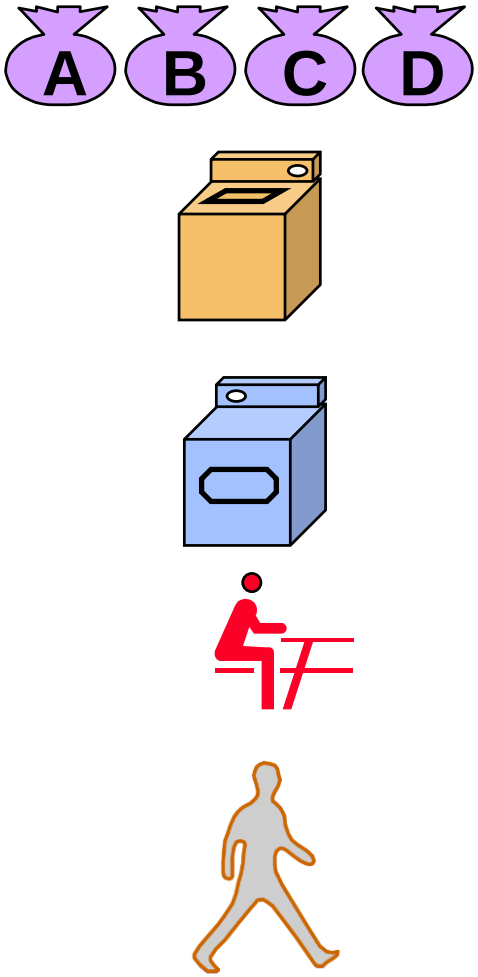
- 流水线概念
- 流水线实现原理
- 流水线性能指标
- RISC-V流水线的实现思路
- 小结

多周期CPU

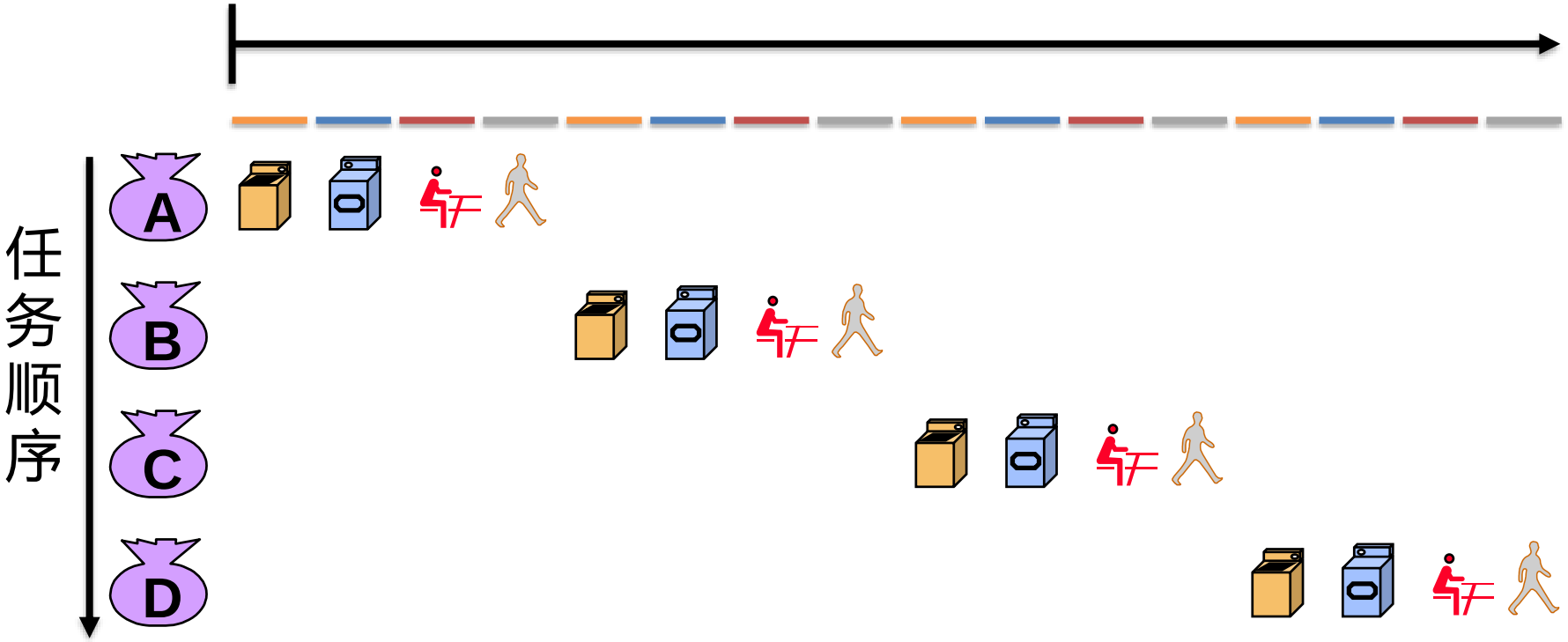
- 多周期CPU特点
  - 指令的执行划分为多个步骤
  - 每个步骤占用一个CPU周期
  - 不同指令的指令周期不同
  - 指令串行执行
  - 提高了整体性能
- 各部件利用率依然偏低
  - CPI > 1
  - 可以如何改进呢?

生活中的流水线

- 例子：洗衣服
- Ann, Brian, Cathy, Dave 每人都要洗、烘干、
- 熨衣服、收拾整理
- 洗衣机：30 分钟
- 烘干机：30 分钟
- 熨衣服：30 分钟
- 收拾并整理：30 分钟

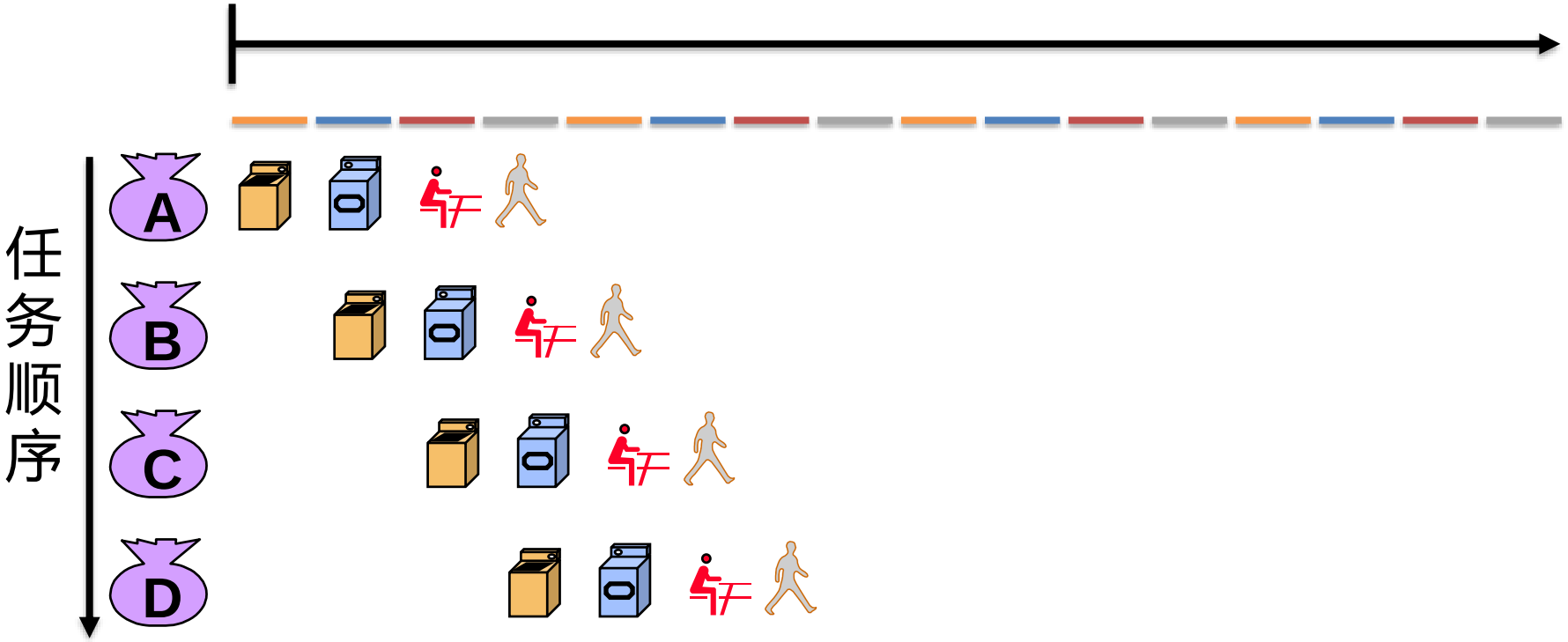


顺序洗衣



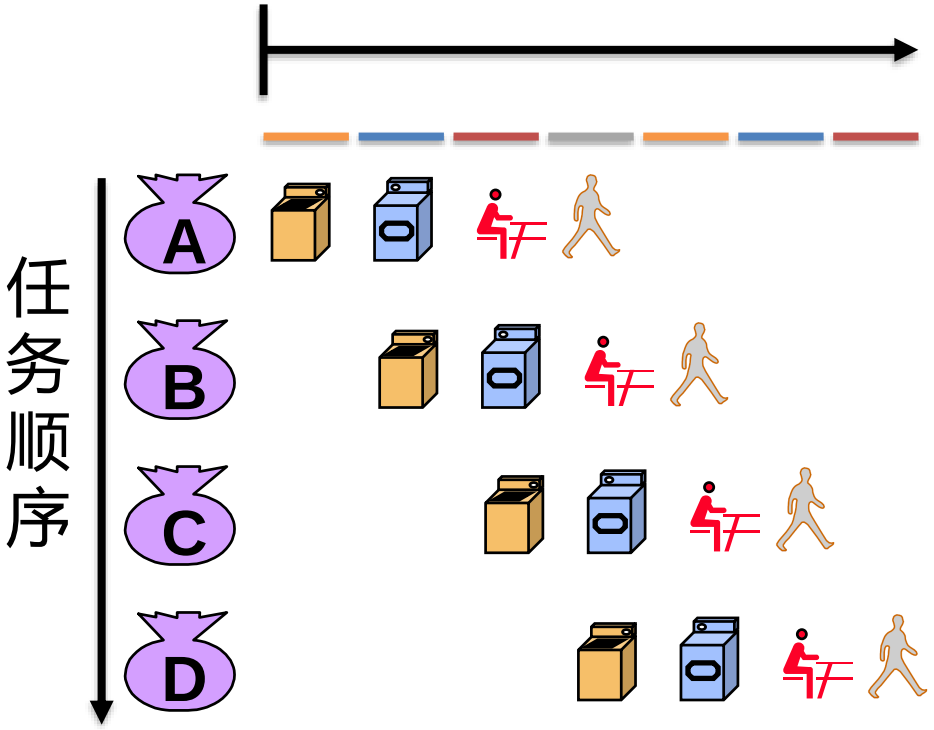
- 顺序洗的话，4人洗完（4个任务）需要8小时
- 如果能流水处理的话，又是什么情况呢?

流水线：尽快启动任务



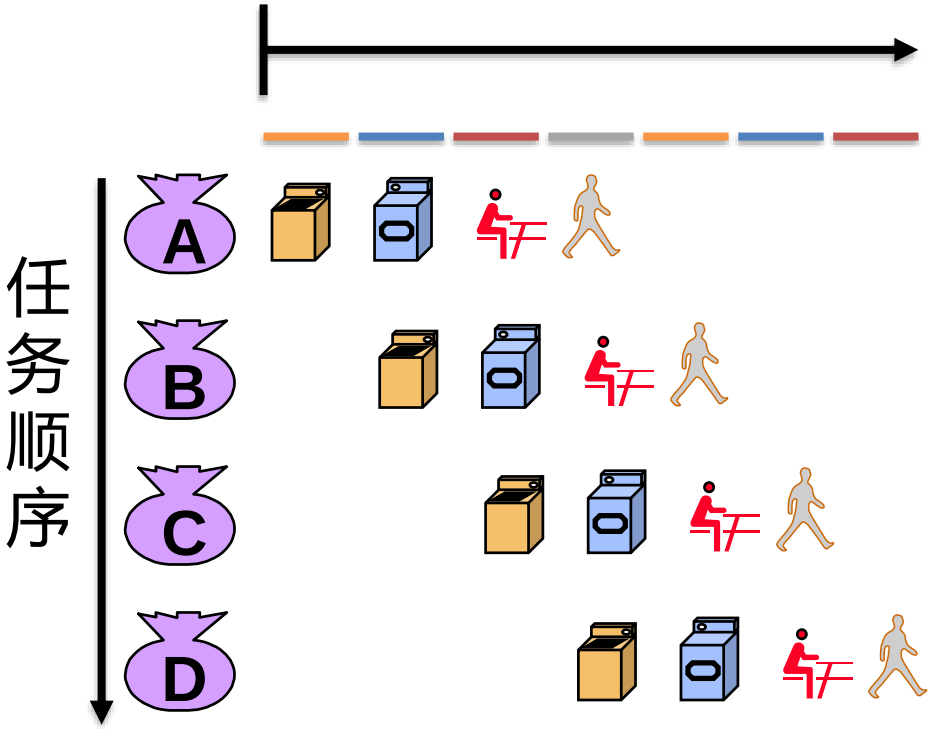
- 流水处理，仅需3.5小时完成!

流水线操作的前提



- 任务分解成多个步骤完成
- 每个步骤使用不同的资源
- 任务内部存在关联，完成每个任务的步骤顺序一致
- 任务之间相互独立，没有依赖关系

流水线操的特性



- 流水线并没有缩短单个任务的延迟，但提高了整个系统的吞吐率。
- 多个任务同时运行，占用不同的资源。
- 可能的加速比= 流水段数
- 流水线效率受限于用时最长的阶段
- 若每个阶段的用时不同，将降低流水线效率
- 装入和排空流水线也可降低加速比
- 冲突将引起流水线的暂停

流水线概念

- 计算机中的流水线是把一个重复的过程分解为若干个子过程，每个子过程与其他子过程并行进行。由于这种工作方式与工厂中的生产流水线十分相似，因此称为流水线技术。
- 提高处理机内部的并行性
  - 空间并行性，即在一个处理机内设置多个独立的操作部件，并且使这些部件并行工作；
  - 时间并行性，就是采用流水线技术。流水线技术是一种非常经济、对提高计算机的运算速度非常有效的技术。采用流水线技术只需增加少量硬件就能把计算机的运算速度提高几倍，成为计算机中普遍使用的一种并行处理技术。
- 计算机各个部分几乎都可以采用流水线技术
  - 指令流水线：指令的执行过程采用流水线
  - 操作部件流水线：运算器中的操作部件，如浮点加法器、浮点乘法器等可以采用流水线
  - 宏流水线：多个计算机之间，通过存储器连接，可以采用流水线

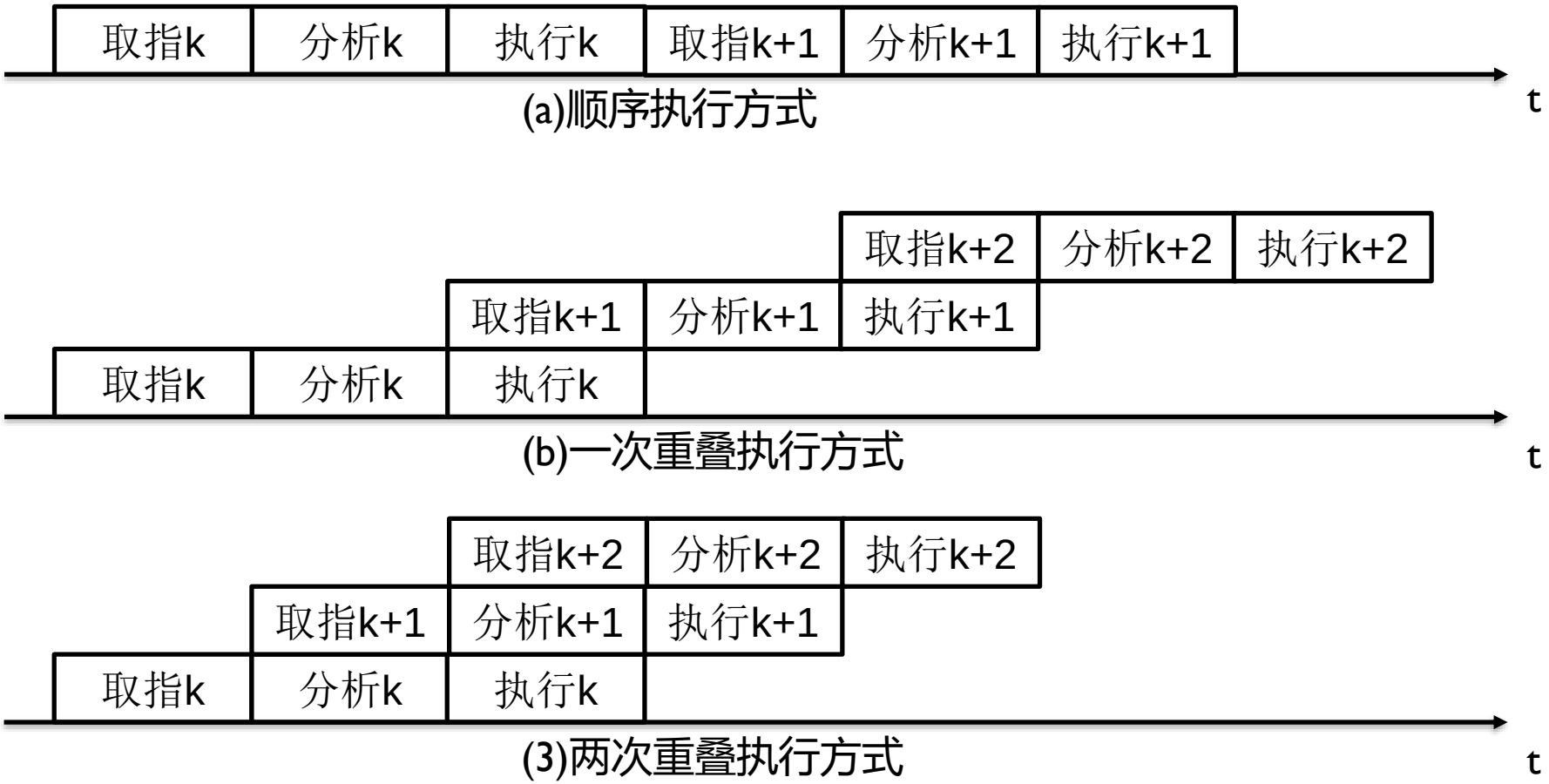
指令流水阶段

- 一条指令的执行过程可以分为多个阶段

- 取指令：按照指令计数器的内容访问主存储器，取出一条指令送到指令寄存器。
- 指令分析：对指令操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数的地址，并用这个地址读取操作数。
- 指令执行：根据操作码的要求，完成指令规定的功能，即把运算结果写到通用寄存器或主存中。

指令执行方式

- 指令多次重叠执行方式实际上就是指令流水线



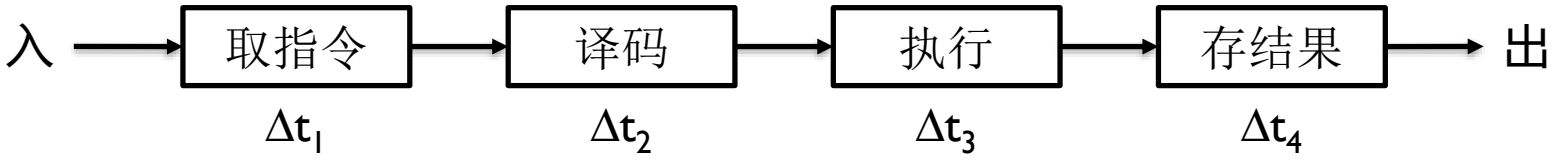
指令流水执行性能比较

- 如果取指令、分析指令、执行指令的时间都相等，每段的时间都为t，则n条指令所用的时间为：
- 顺序执行：T = 3nt
- 一次重叠执行：T = (1 + 2n)t
- 两次重叠执行：T = (n + 2)t

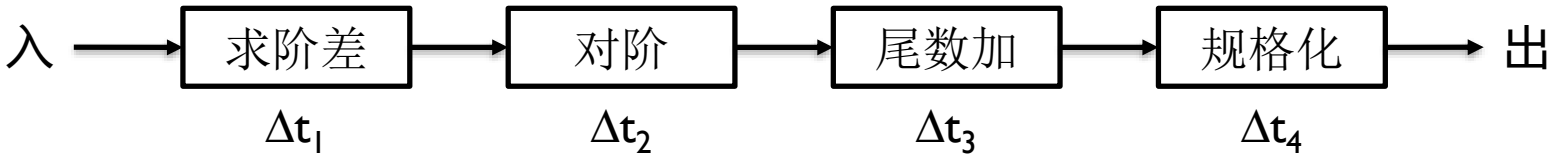
流水线的表示

- 流水线的每一个阶段完成一条指令的一部分，不同阶段并行完成不同指令的不同部分。
- 流水线中的每一个阶段称为一个流水阶段、流水节拍、流水步、流水段、功能段、流水级等。一个流水阶段与另一个流水阶段相连接形成流水线。
- 指令从流水线的一端进入，经过流水线的处理，从另一端流出。目前大部分处理机的指令流水线在3~12段之间。
- 流水线常用的两种表示方法
  - 流水线连接图表示法，各个流水段顺序连接在一起
  - 流水线时空图表示法，直观描述流水线工作过程

流水线连接图表示法



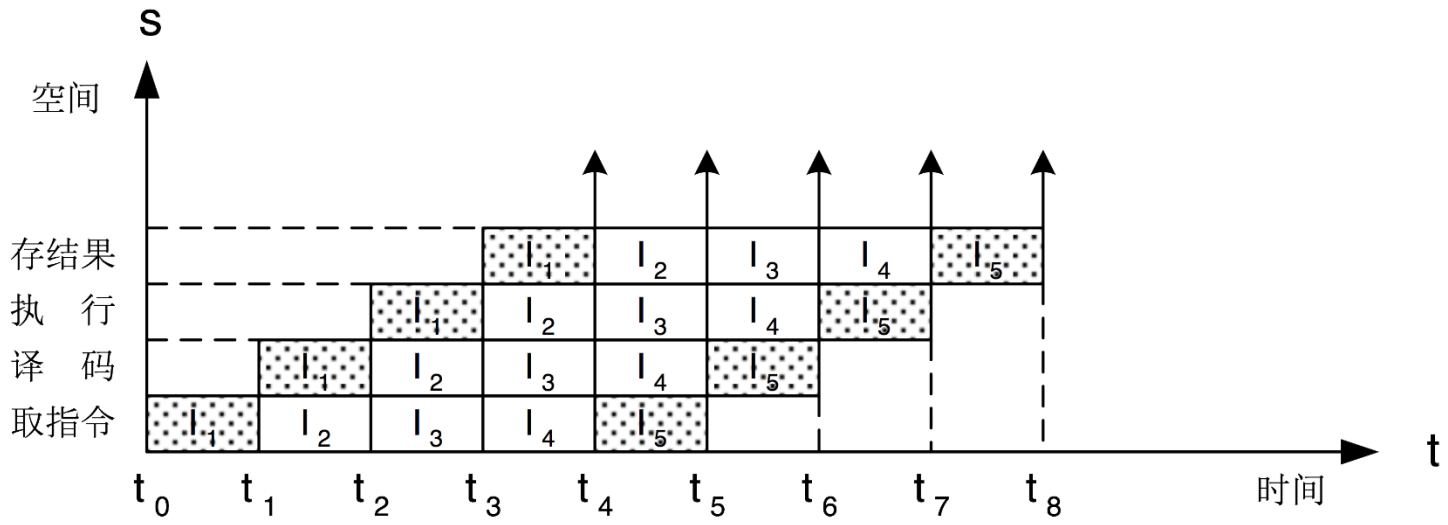
4段指令流水线



浮点加法器流水线

流水线时空图表示法

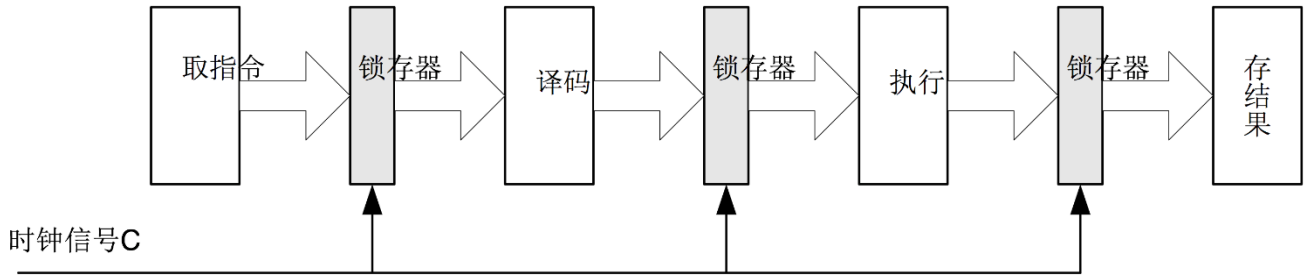
- 横坐标表示时间，也就是输入到流水线中的各个任务在流水线中所经过的时间。纵坐标表示空间，即流水线的每一个流水段。



指令流水线时空图

流水线的特点

- 把一个任务（一条指令或一个操作）分解为几个有联系的子任务，每个子任务由一个专门的功能部件来实现。
- 流水线每一个功能段部件后面都要有一个缓冲寄存器，或称为锁存器，其作用是保存本流水段的结果。



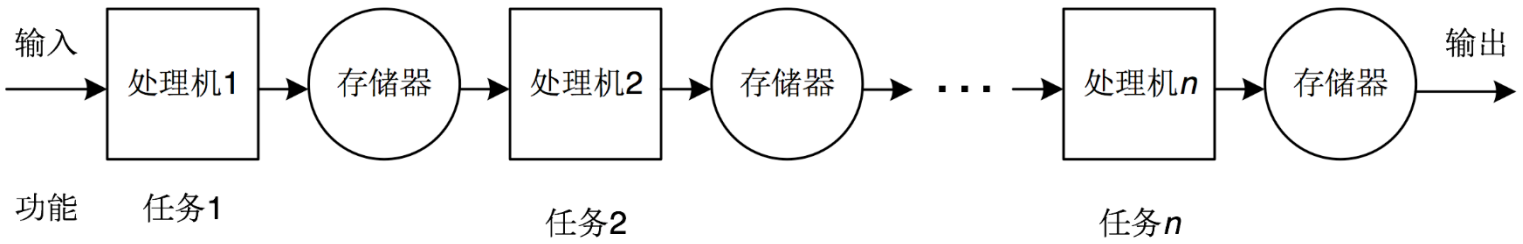
流水线中的锁存器

流水线的特点

- 流水线中各功能段的时间应尽量相等，否则将引起堵塞、断流。要求流水线的时钟周期不能短于最慢的流水段。
- 只有连续不断地提供同一种任务时才能发挥流水线的效率，所以在流水线中处理的必须是连续任务。
- 流水线需要有装入时间和排空时间。装入时间是指第一个任务进入流水线到输出流水线的时间。排空时间是指第n个（最后一个）任务进入流水线到输出流水线的时间。

流水线分类

- 部件功能级流水线
  - 所谓功能部件级流水线也可以称为运算操作流水线（Arithmetic Pipelines）。浮点加法器就是一种典型的功能部件级流水线。
- 处理机级流水线
  - 所谓处理机级流水线，又叫指令流水线（Instruction Pipelines），它是把解释指令的过程按照流水方式处理，使处理机能够重叠地解释多条指令。
- 处理机间级流水线
  - 所谓处理机间流水线，又被称为宏流水线（Macro Pipelines）。这种流水线由两个或者两个以上的处理机通过存储器串行连接起来，每个处理机完成整个任务的一部分。



一种宏流水线

流水线的性能指标

- 衡量流水线性能的主要指标
  - 吞吐率：单位时间执行指令的数量
  - 加速比：与串行执行时速度提高的比率
- 流水线设计中，流水线的最佳段数选择也是一个重要问题。

流水线的实现原理

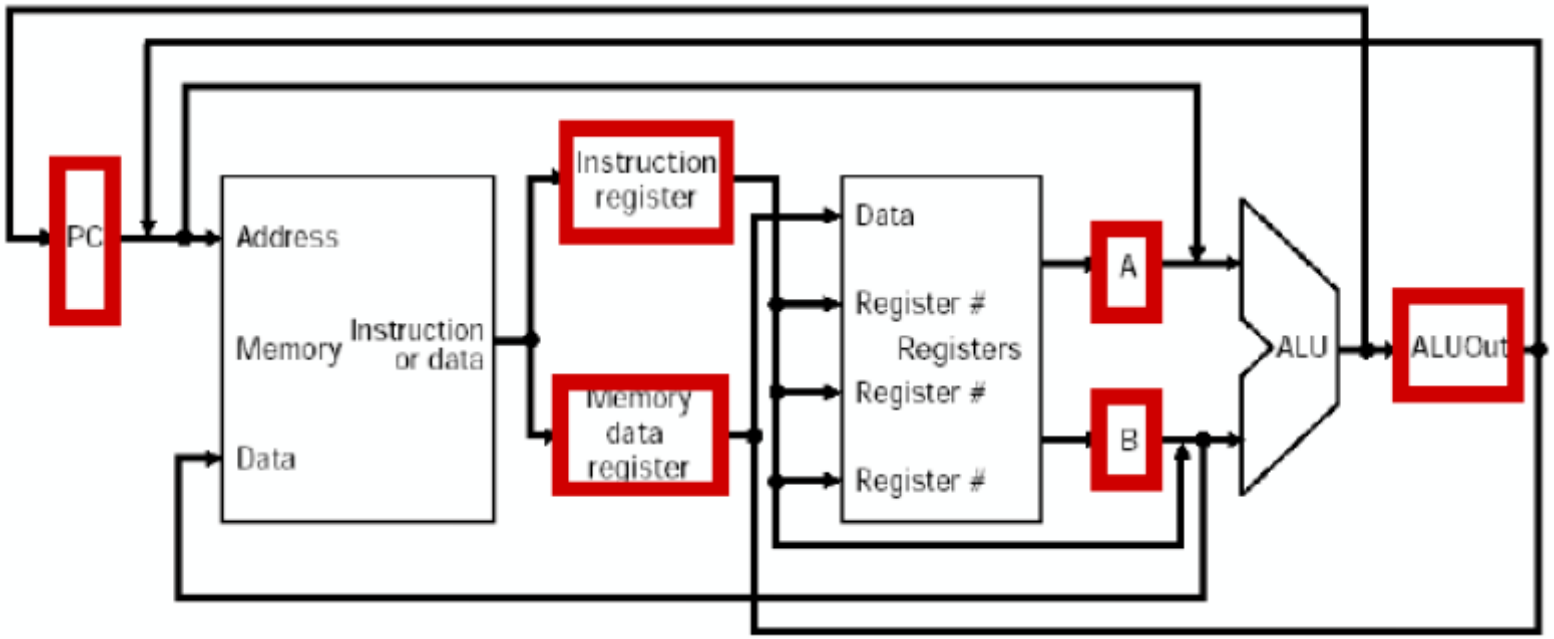
- 每一条指令的实现至多需要5个时钟周期。这5个时钟周期如下：
  - 取指令周期（IF）
  - 指令译码 / 读寄存器周期（ID）
  - 执行 / 有效地址计算周期（EX）
  - 存储器访问 / 分支完成周期（MEM）
  - 写回周期（WB）
  - 不同类型的指令在以上5个时钟周期中进行的操作各不相同

RISC-V指令的流水实现

- 指令执行步骤
  - 取指令（IF）
  - 指令译码（ID/RF）
  - 指令执行（EXE）
  - 读存储器（MEM）
  - 写回（WB）
- 各步骤占用的资源
  - IF：IM、PC、总线
  - ID/RF：寄存器组、控制信号生成部件
  - EXE：ALU
  - MEM：DM、总线
  - WB：寄存器组

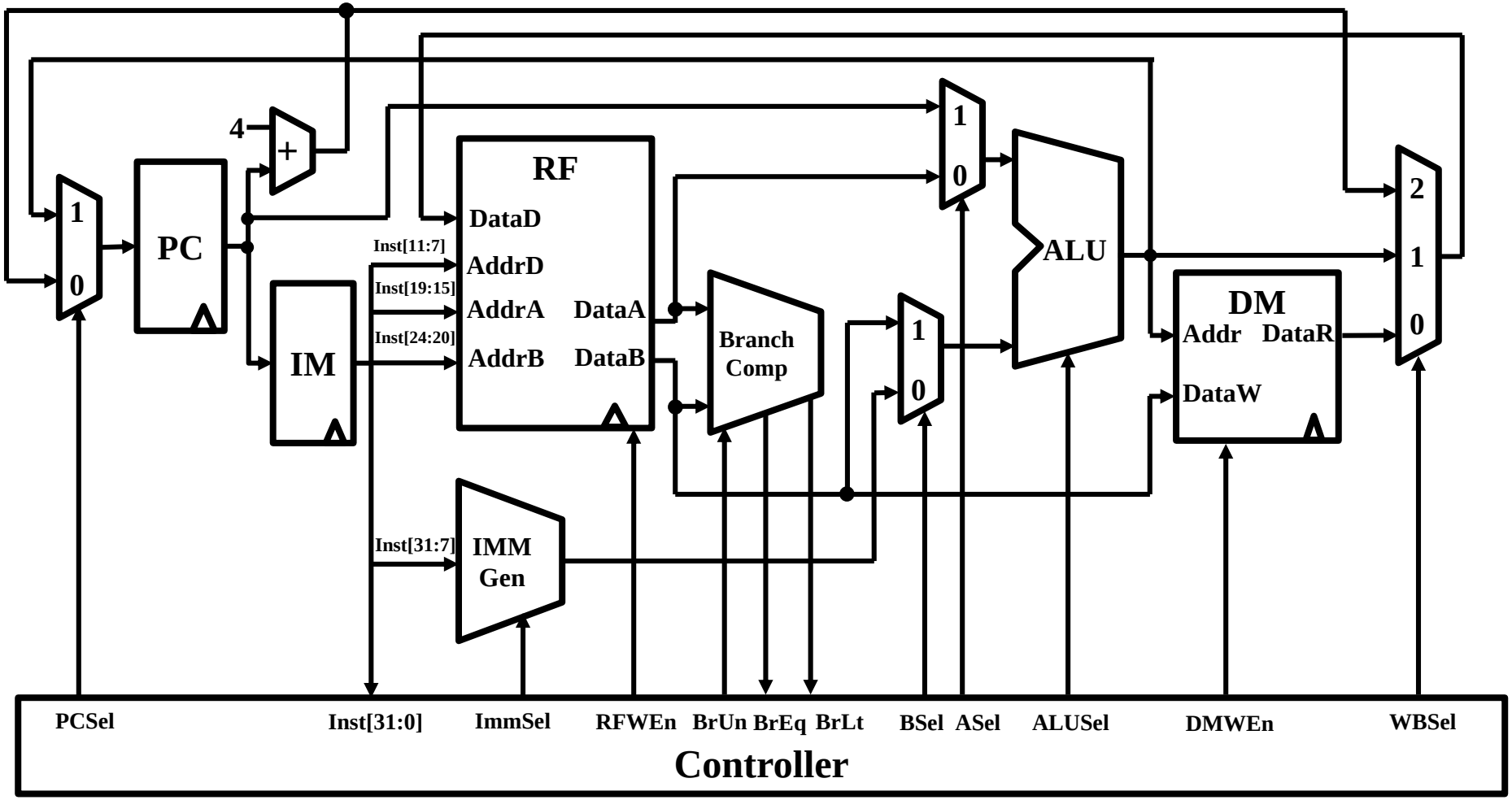
多周期CPU

- 多周期CPU适合指令流水实现吗？
  - 已分解为多个步骤
  - 但是，步骤间的资源冲突比较频繁

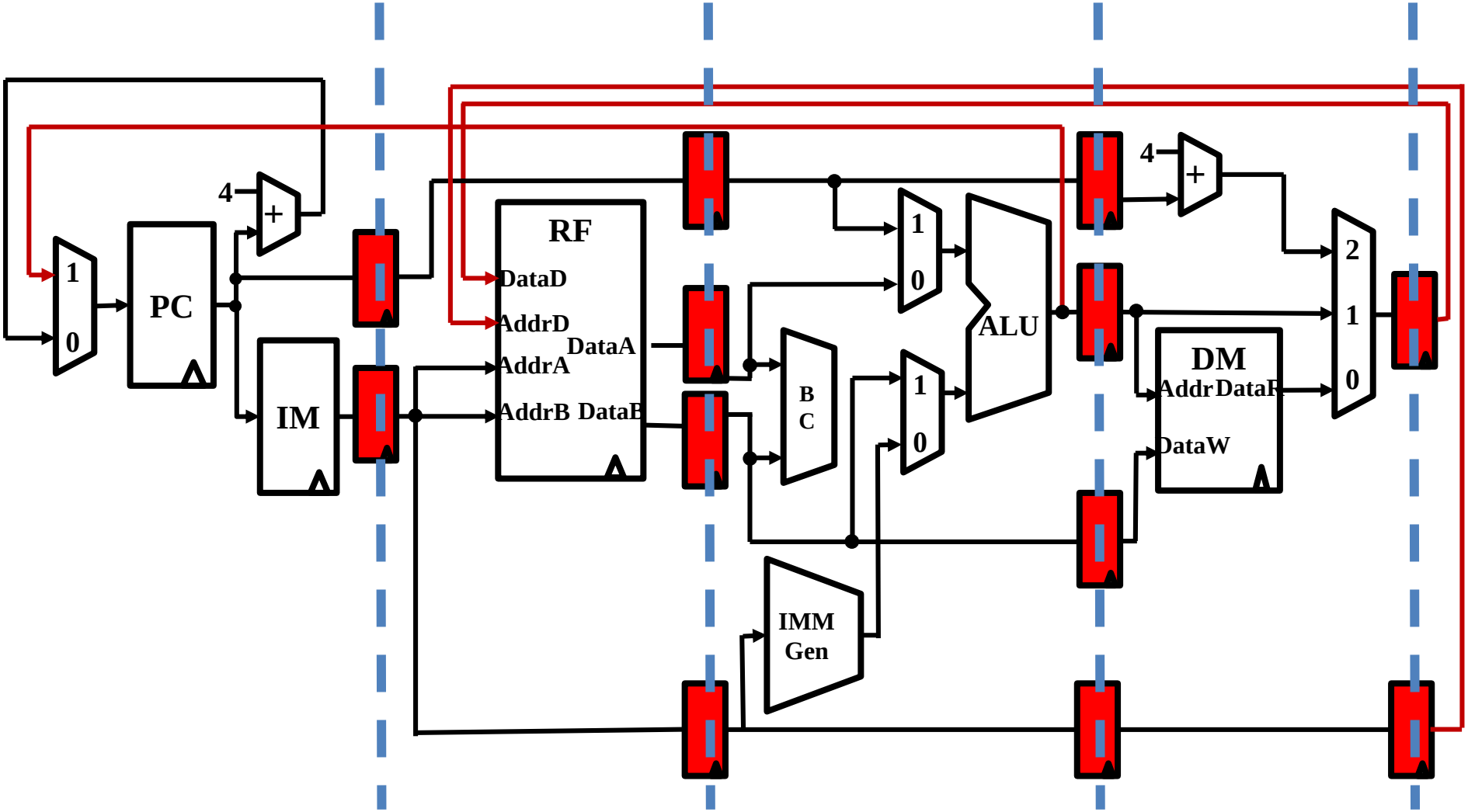


单周期CPU

- 是否容易实现指令流水?



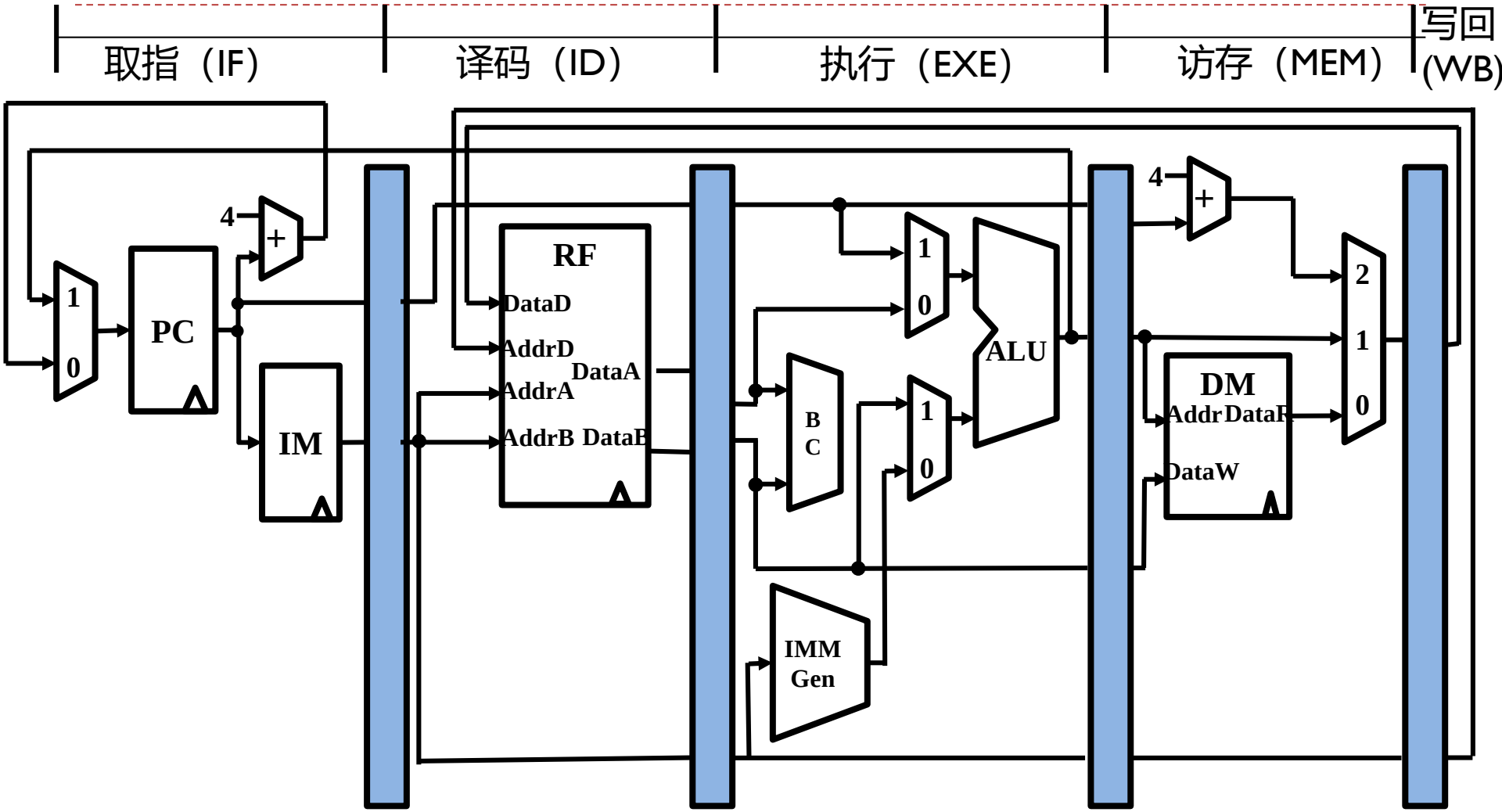
切分流水线


4
+
1
0
PC
IM
RF
DataD
AddrD
DataA
AddrA
AddrB DataB
B
C
1
0
ALU
4
+
DM
Addr DataR
DataW
2
1
0
IMM
Gen

流水线的实现

- 在流水线的各个流水段之间加入被称为流水线寄存器（流水线锁存器）的寄存器堆，并在这些寄存器堆上标明所连接的流水段。
  - 所有用于在同一条指令的各个时钟周期之间保存临时数据的寄存器，都归入流水线寄存器这一类中。
  - 流水线寄存器保存着从一个流水段传送到下一个流水段的所有数据和控制信息。
- PC值多路选择器被放到IF段，这样做的目的是保证对PC值的写操作只出现在一个流水段内，否则当分支转移成功的时候，流水线中两条指令都试图在不同的流水段修改PC值，从而发生写冲突。
- 每个时刻，每条指令都只在一个流水段上是活动的，因此，任何指令所作的任何动作都发生在一对流水线寄存器之间，具体操作由指令类型决定。

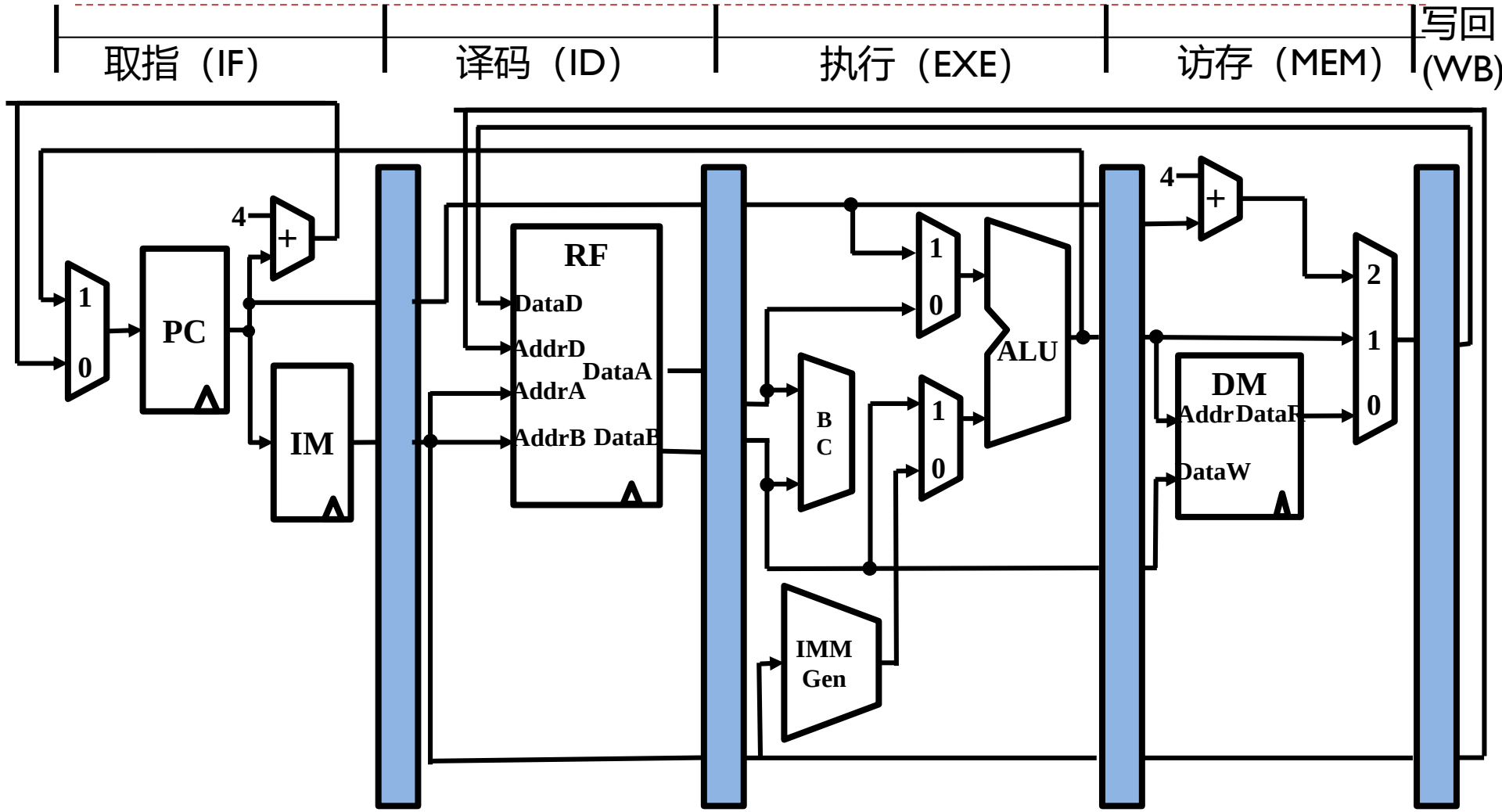
流水线的数据通路划分


取指（IF）
译码（ID）
执行（EXE）
访存（MEM）
写回（WB）
4
+
1
0
PC
IM
RF
DataD
AddrD
DataA
AddrA
AddrB DataB
B
C
1
0
ALU
IMM
Gen
4
+
DM
Addr DataR
DataW
2
1
0

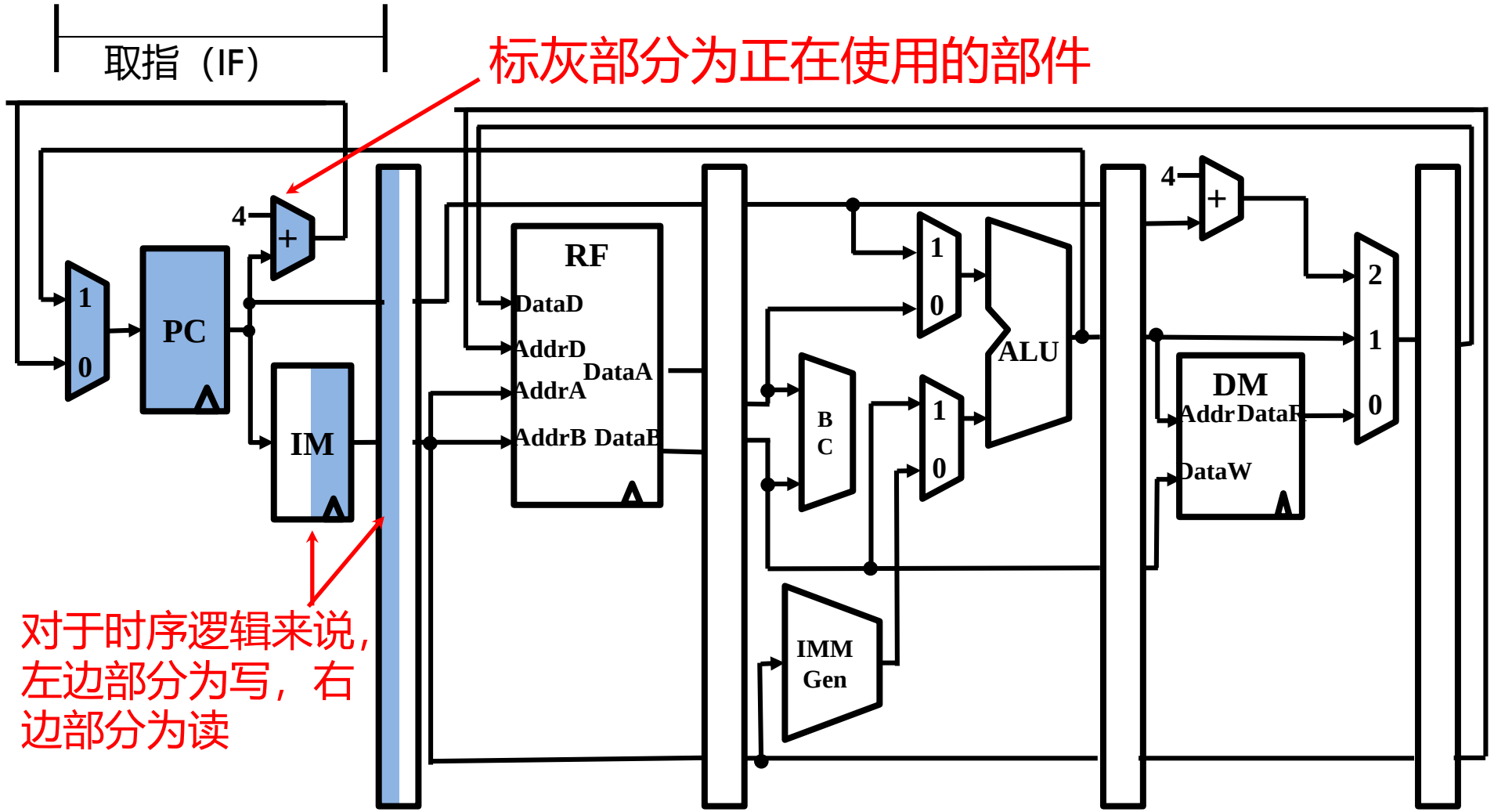
各阶段寄存器保存的值

- IF/ID
  - PC
  - IR
- ID/EXE
  - PC、A、B
  - inst
- EXE/MEM
  - PC、ALU结果、B
  - inst
- MEM/WB
  - DataD数据：PC+4、ALU结果、存储器读出的结果
  - AddrD地址（inst）

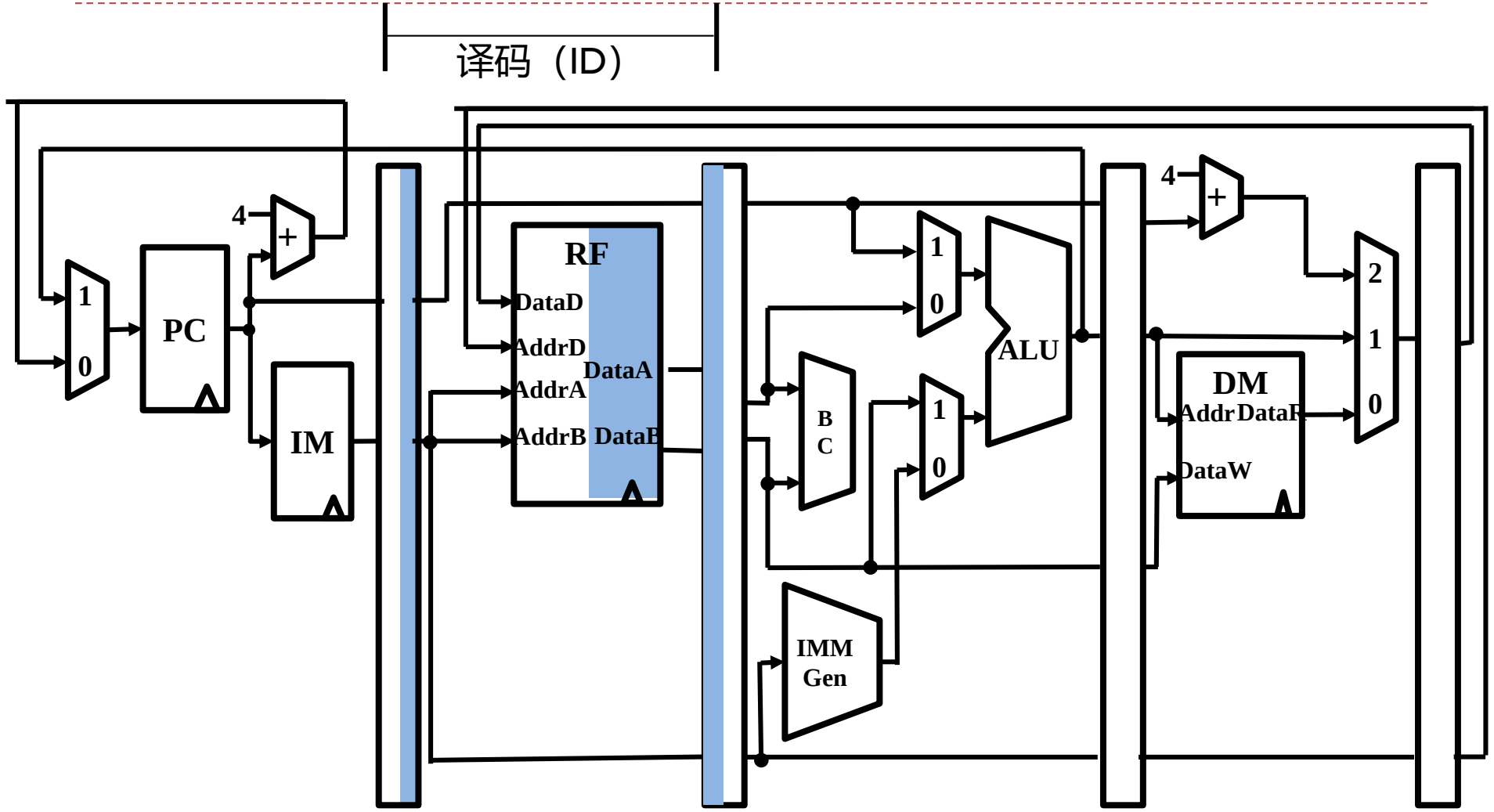
LW指令在流水线中的执行



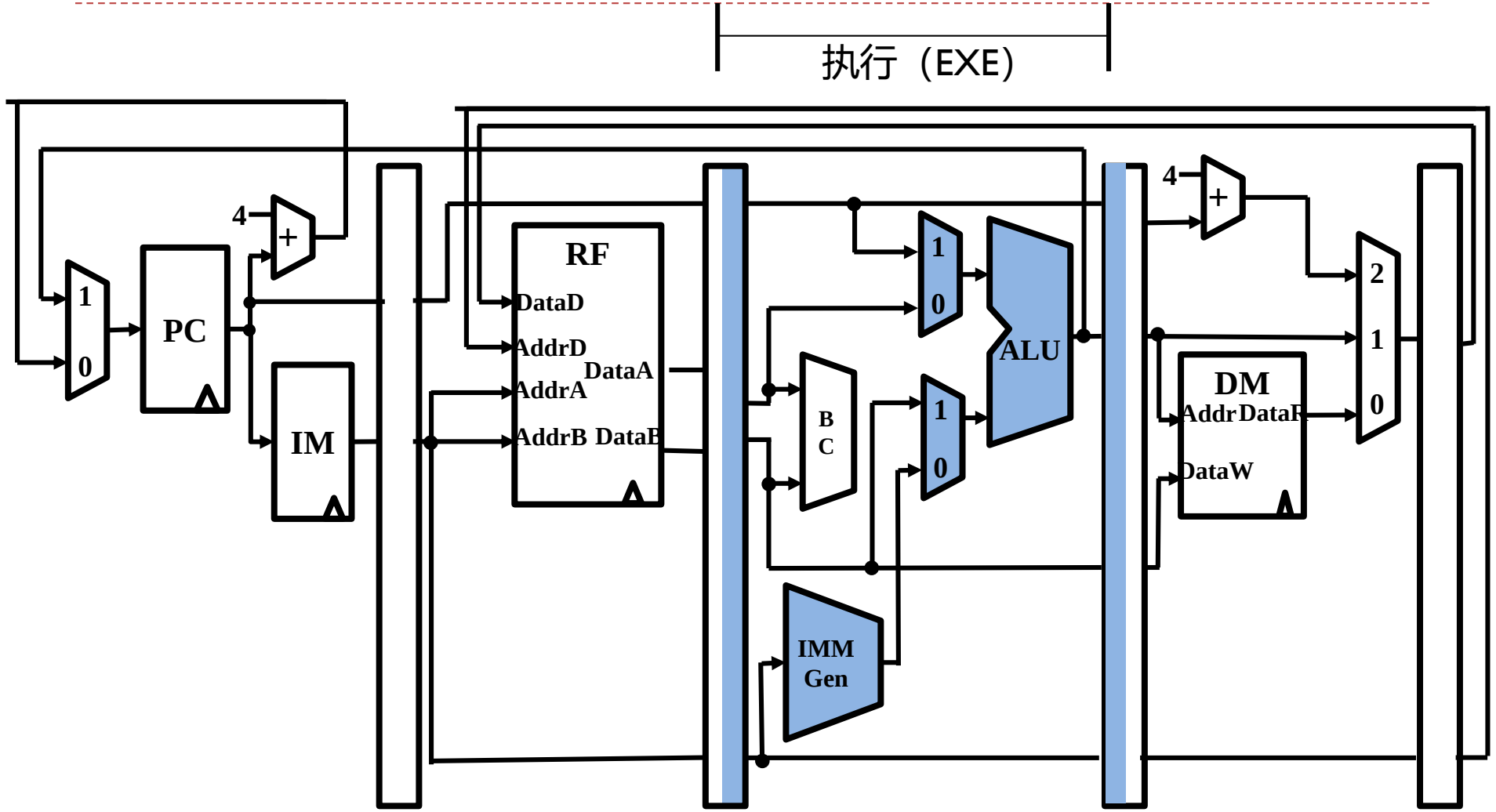
lw指令的取指部分


取指（IF）
标灰部分为正在使用的部件
4
+
1
0
PC
IM
RF
DataD
AddrD
DataA
AddrA
AddrB DataB
B
C
1
0
ALU
4
+
DM
Addr DataR
DataW
2
1
0
IMM
Gen
对于时序逻辑来说，左边部分为写，右边部分为读

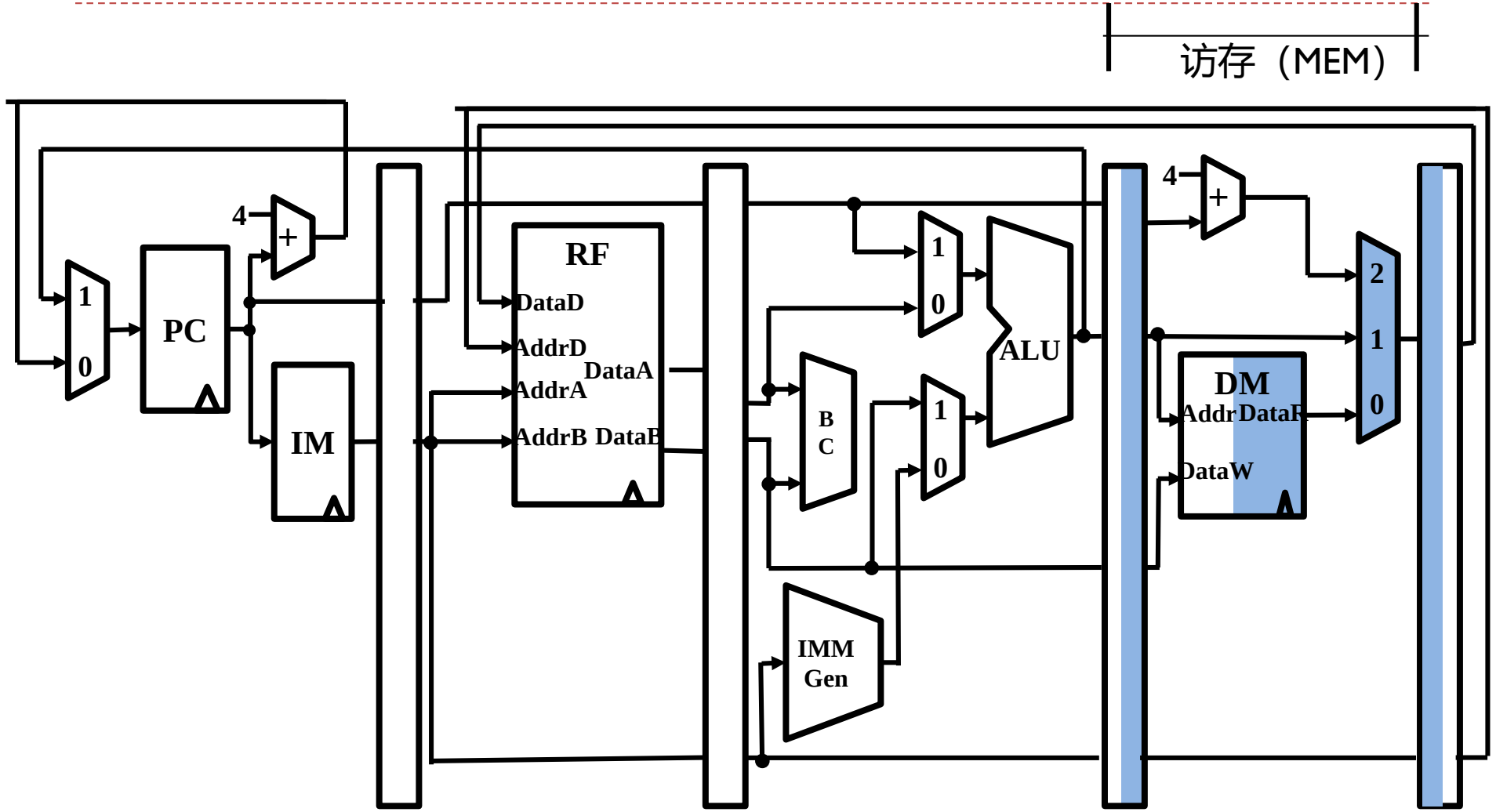
LW指令的译码阶段


译码（ID）
4
+
1
0
PC
IM
RF
DataD
AddrD
AddrA
AddrB
DataA
DataB
B
C
1
0
ALU
IMM
Gen
4
+
DM
Addr
DataR
DataW
2
1
0

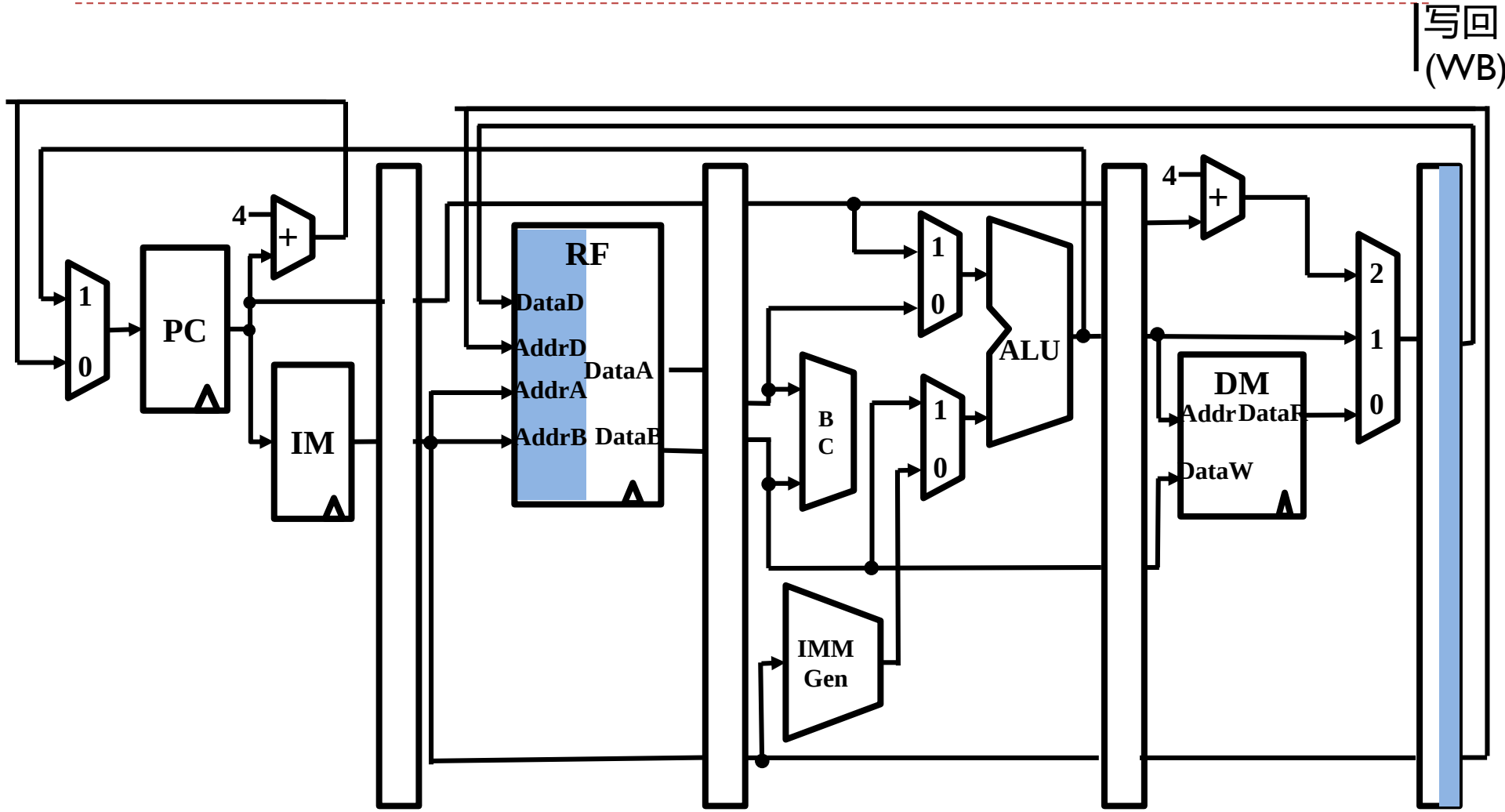
lw指令的执行阶段


执行（EXE）
4
+
1
0
PC
IM
RF
DataD
AddrD
DataA
AddrA
AddrB DataB
B
C
1
0
ALU
IMM
Gen
4
+
DM
Addr DataR
DataW
2
1
0

lw指令的访存阶段


访存（MEM）
4
+
1
0
PC
IM
RF
DataD
AddrD
DataA
AddrA
AddrB DataB
B
C
1
0
ALU
IMM
Gen
4
+
DM
Addr DataR
DataW
2
1
0

lw的写回阶段


写回
(WB)
4
+
1
0
PC
IM
RF
DataD
AddrD
AddrA
AddrB
DataA
DataB
B
C
1
0
ALU
IMM
Gen
4
+
DM
Addr
DataR
DataW
2
1
0

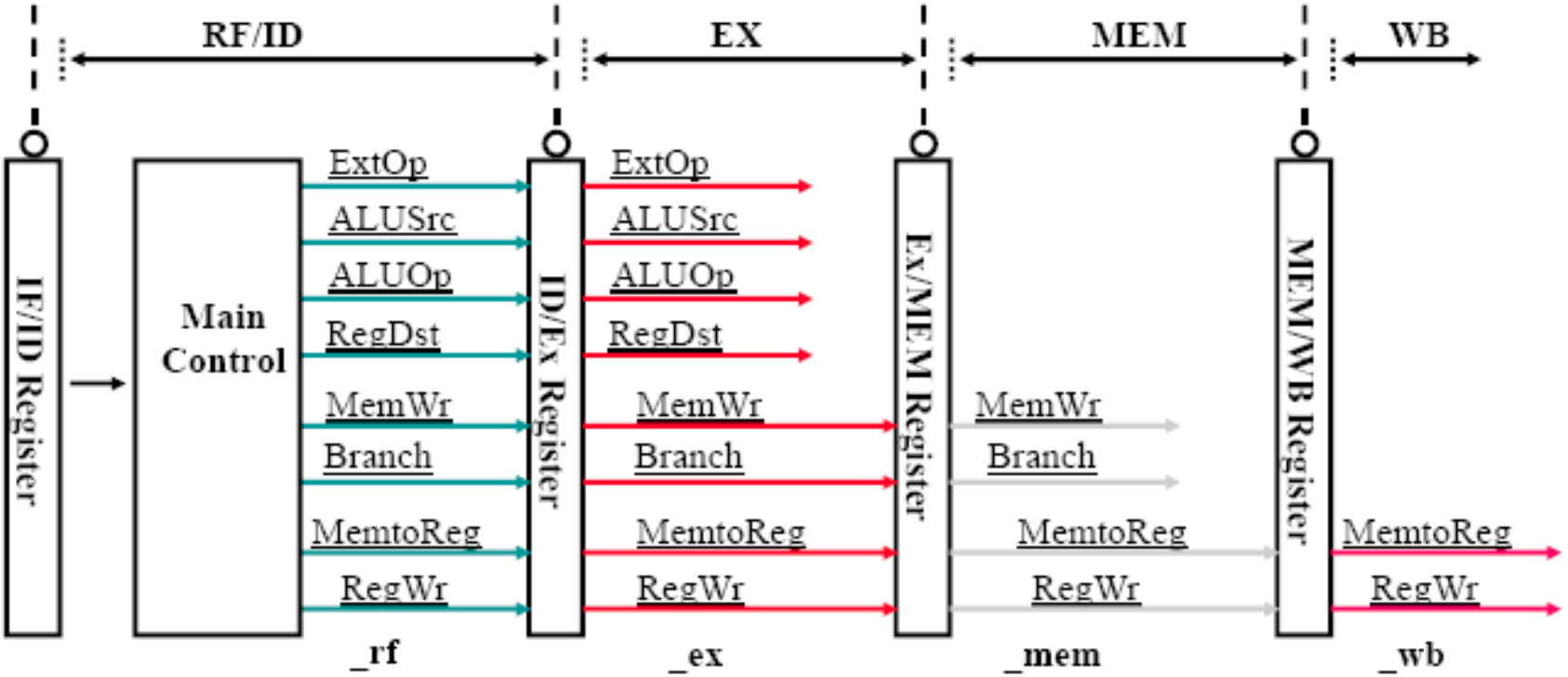
指令系统的配合

- 指令字长
  - 尽管在第2个步骤才进行指令译码
  - 定长指令字简化了流水的实现
- 指令格式
  - 使访问寄存器组的操作可以尽早开始
- 访存方式
  - 其他指令不需要在进行运算前计算存储器地址
  - 地址计算完成后可直接访问存储器
- 数据对齐
  - 一次访存可得到一个完整的字

流水线的控制

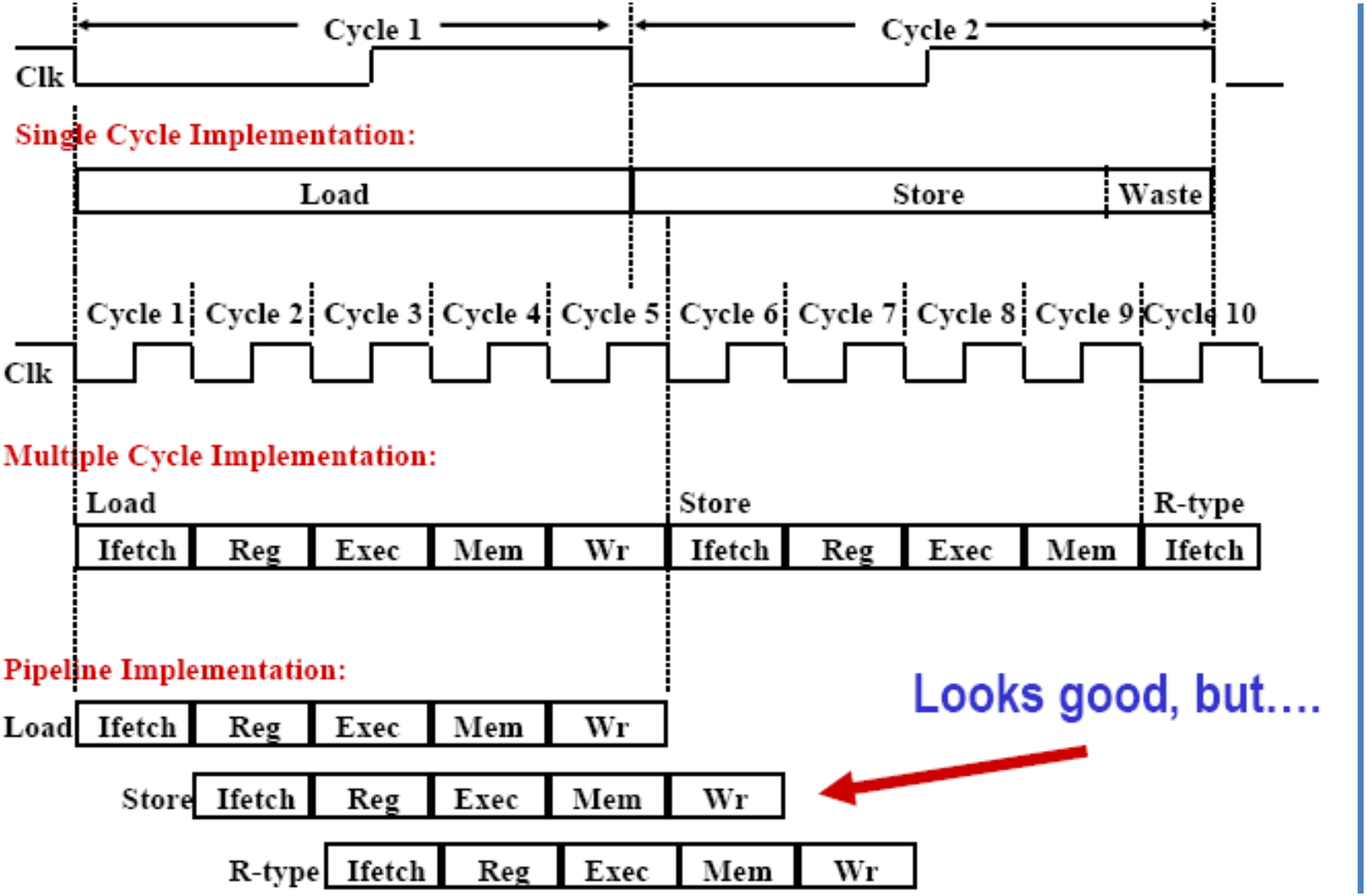
- 控制到每个功能部件
  - 但是，每个部件执行的不是同一条指令
- 解决方案
  - 把控制信号也和数据一样流动起来
  - 为区分起见，可以把控制信号前面加上标记，如_IF等
  - 每个时钟周期往下一步骤传递控制信号
    - 使正确的控制信号到达正确的位置

流水线控制的实现



- 在RF/ID阶段生成控制信号
  - 1个时钟周期后使用EX要用的控制信号
  - 2个时钟周期后使用MEM要用的控制信号
  - 3个时钟周期后使用WB要用的控制信号

比较


Cycle 1
Cycle 2
Clk
Single Cycle Implementation:
Load
Store
Waste
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Cycle 9
Cycle 10
Clk
Multiple Cycle Implementation:
Load
Store
R-type
Ifetch
Reg
Exec
Mem
Wr
Ifetch
Reg
Exec
Mem
Ifetch
Pipeline Implementation:
Looks good, but....
Load
Ifetch
Reg
Exec
Mem
Wr
Store
Ifetch
Reg
Exec
Mem
Wr
R-type
Ifetch
Reg
Exec
Mem
Wr

流水线的实现原理

- 简单的基本流水线

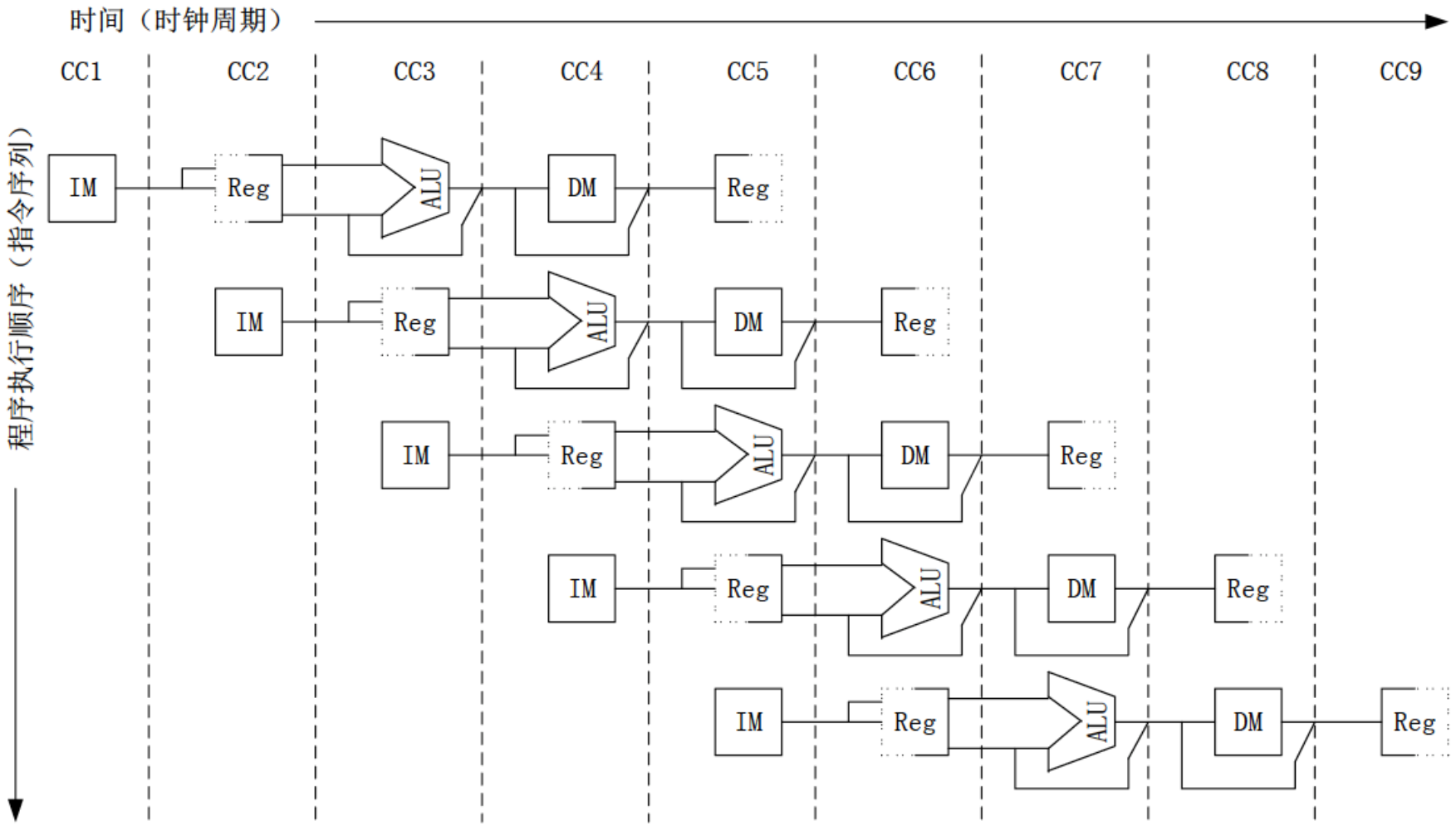
指令	时钟								
	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

- 每一个时钟周期启动一条新的指令，就可以使数据通路成功流水，每一个时钟周期就是流水线的一个流水段。每一条指令经过5个时钟周期执行完成，而在每一个时钟周期内，硬件将启动一条新的指令并执行5条不同指令的某个部分。

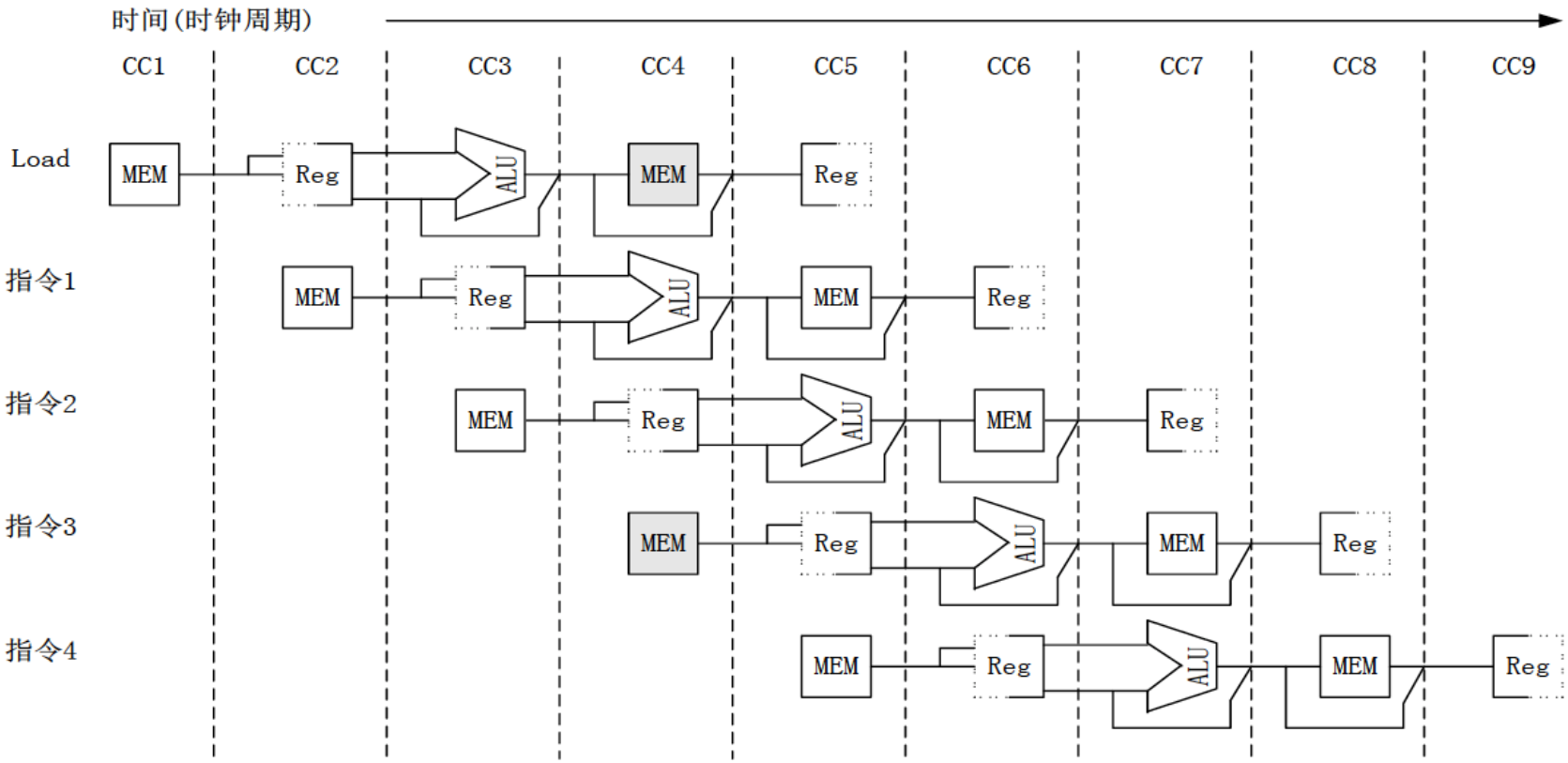
流水线的实现原理

- 实际的流水线就这么简单吗？——NO！
  - 必须保证在指令重叠执行时不会存在任何流水线资源冲突问题，即要保证流水线的各段在同一个时钟周期内不会使用相同的数据通路资源。
- 简化的流水线数据通路
  - 下图从使用流水线资源的角度描述上述流水线的流水过程，显示了不同数据通路的重叠，周期5表示稳定状态。
  - 在包围每个流水段的线框中，如果实线在右侧，说明是读操作；如果实线在左侧，说明是写操作；其他部分用虚线。
  - 主要的功能部件都在不同的时钟周期内使用，因而多条指令重叠执行时引入的冲突很少。
    - 访存：分开的指令存储器（IM）和数据存储器（DM）。
    - 在两个流水线段都使用了寄存器组：ID段读，WB段写。
    - 没有考虑PC的问题，流水要求IF段要形成新的PC值。

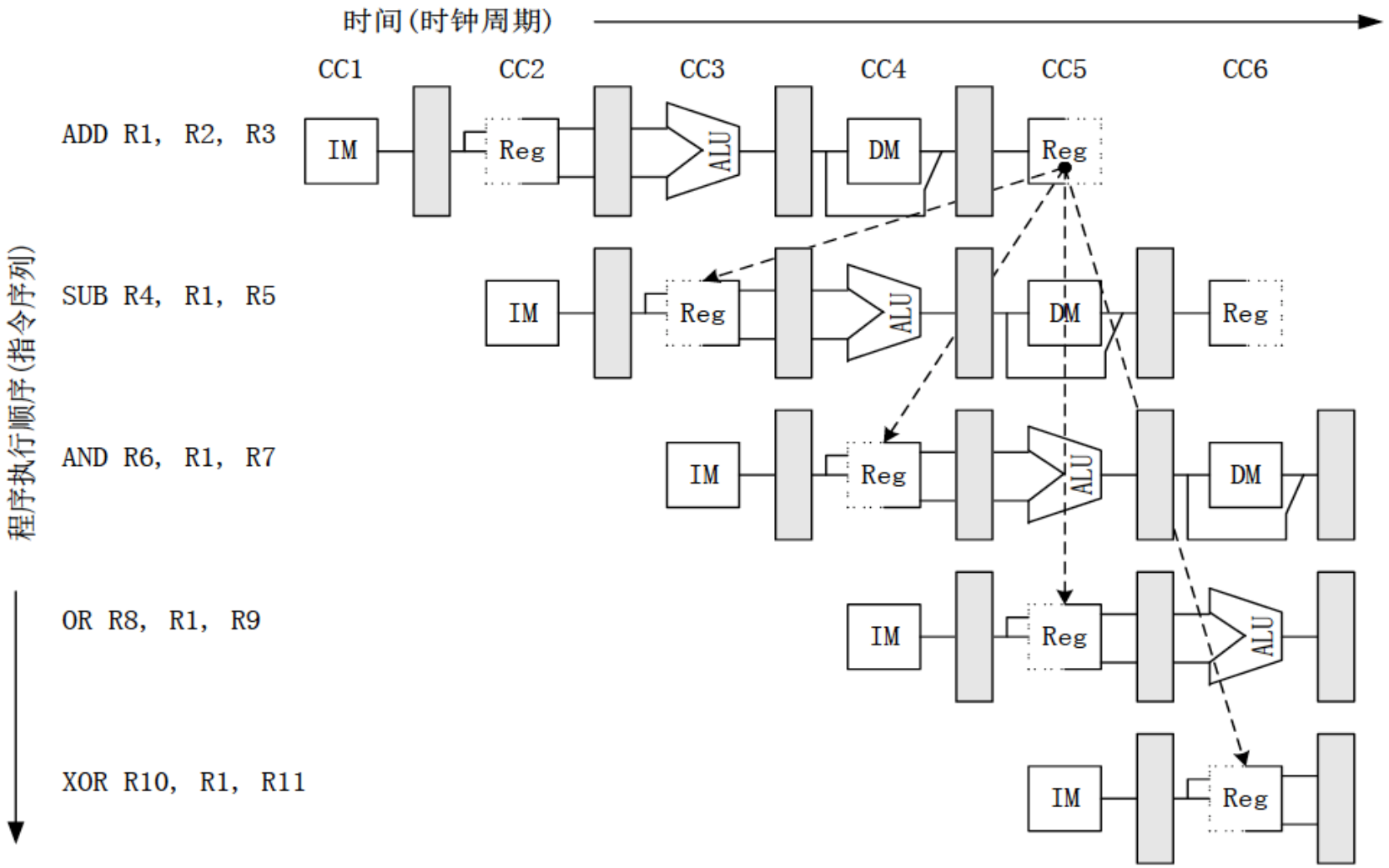
流水线的实现原理


时间（时钟周期）
CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC9
程序执行顺序（指令序列）
IM
Reg
ALU
DM
Reg

资源发生冲突


时间(时钟周期)
CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC9
Load
指令1
指令2
指令3
指令4
MEM
Reg
ALU

数据发生冲突


时间(时钟周期)
CC1
CC2
CC3
CC4
CC5
CC6
ADD R1, R2, R3
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
程序执行顺序(指令序列)
IM
Reg
ALU
DM

转移地址的问题

- 转移指令的地址
  - 一般在EXE阶段计算
- 下一条指令
  - 取哪一条?

流水线的冲突（Hazard）

- 什么是流水线中的“冲突”？
  - 在流水线中经常有一些被称为“冲突”的情况发生，它使得指令序列中下一条指令无法按照设计的时钟周期执行，这些“冲突”可能会降低流水线可以获得的理想性能。
- 流水线中的冲突
  - 第一种是结构冲突，是指令在重叠执行的过程中，硬件资源满足不了指令重叠执行的要求，发生硬件资源冲突而产生的冲突。
  - 第二种是数据冲突，是指在同时重叠执行的几条指令中，一条指令依赖于前面指令执行结果数据，但是又得不到时发生的冲突。
  - 第三种是控制冲突，它是指流水线中的分支指令或者其他需要改写PC的指令造成的冲突。
- 解决流水线中“冲突”问题的重要性
  - 流水线冲突问题是流水线执行过程中的主要障碍，会给流水线中指令序列的顺利执行带来许多不利的影响。如果不能较好的处理流水线冲突问题，就可能影响流水线的性能，甚至使程序运行产生错误的结果。

小结

- 指令流水
  - 在不过多增加硬件投入的情况下，提高系统效率
  - 单任务的速度不提高，但提高系统的吞吐率
- 流水线评价指标
  - 吞吐率、加速比
- 指令流水的实现
  - 分段、锁存
- 冲突问题
  - 结构冲突
  - 数据冲突
  - 控制冲突

阅读和思考

- 阅读

- 思考
  - 实现指令流水的基本条件是什么?
  - 如何检测和避免流水中冲突问题的发生?
- 实践
  - 根据ThinPAD RISC-V指令系统的要求，设计ALU的功能并进行实现。
  - 为ThinPAD RISC-V指令系统设计Datapath，划分每条指令的执行步骤，以及每个步骤具体的功能。
  - 如果支持流水，数据通路要进行怎样的修改?

谢谢