

计算机系统结构

Computer Architecture

Ch.03 – CPU Pipeline

流水线技术

Dr. Borui Li
李博睿 博士

2026年秋季·东南大学





- **本章回答的问题**

- Q1 (What): 什么是流水线?
- Q2 (Why): 为什么要学习流水线?
- Q3 (How): 如何设计流水线?

- **本章目标**

- **理解**指令流水线的组成&工作原理, **掌握**流水线冒险的处理技术





Part I. 为什么要学习流水线?

从CPU性能公式讲起



• CPU性能公式

$$\text{程序CPU时间} = \underbrace{\text{IC}}_{\text{指令条数}} \times \underbrace{\text{CPI}}_{\text{平均指令周期}} \times \text{时钟周期时间}$$

• CISC与RISC

- CISC: 强化指令功能, 减少IC、增加CPI
- RISC: 简化指令功能, 减小CPI、增加IC

• RISC占优

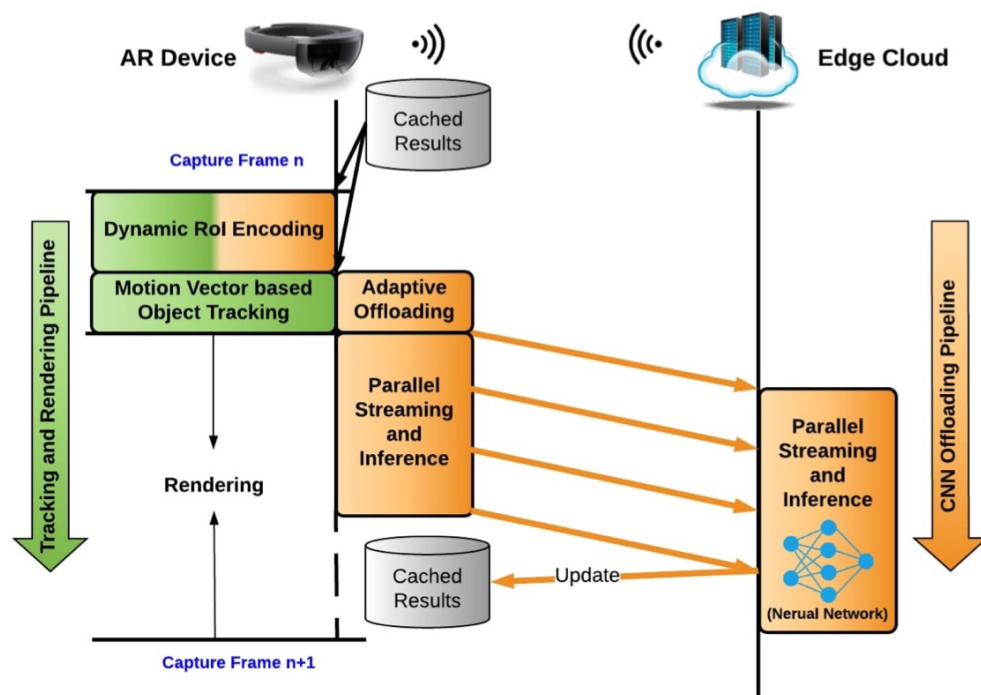
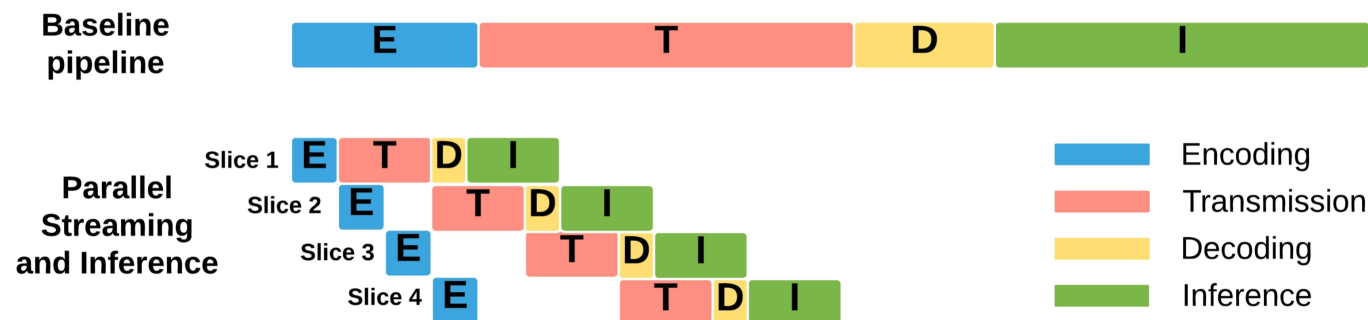
- 利于并行处理、VLSI



流水线思想在计算机其他方向中的应用



• 计算协同



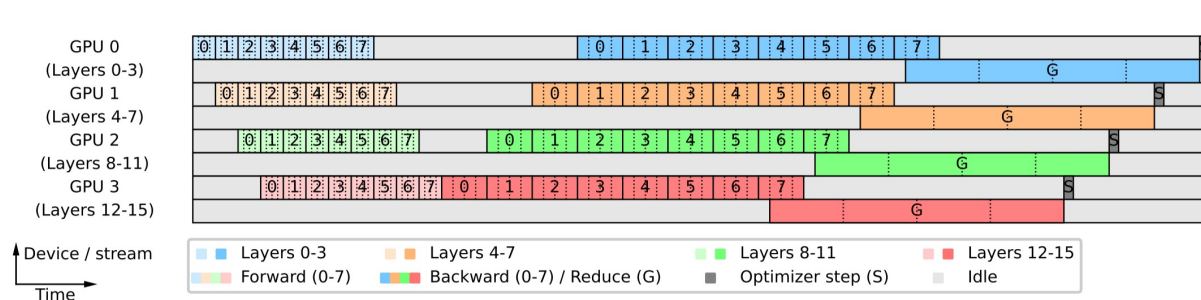
Liu, L., Li, H. and Gruteser, M., 2019, August. Edge assisted real-time object detection for mobile augmented reality. In *ACM MobiCom 2019* (pp. 1-16).



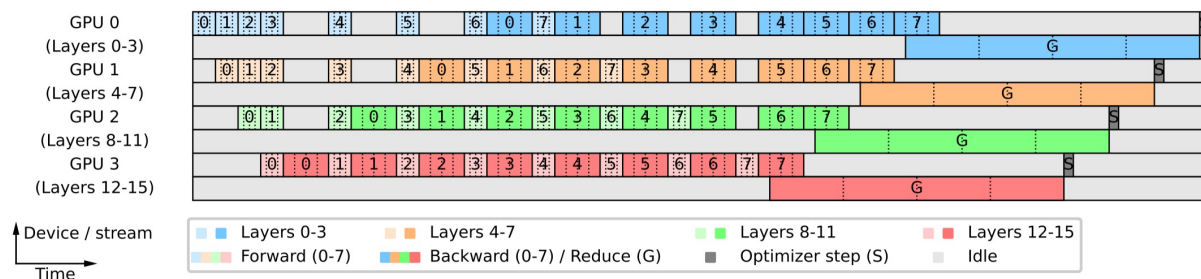
流水线思想在计算机其他方向中的应用



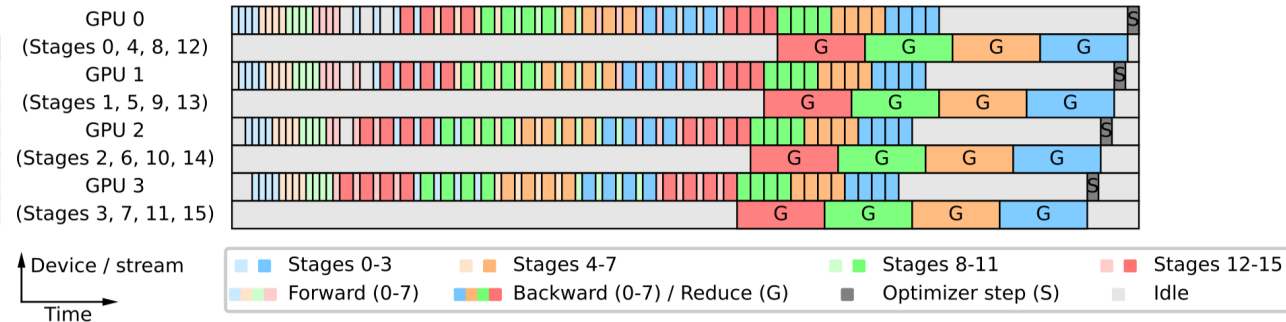
• 神经网络模型训练优化



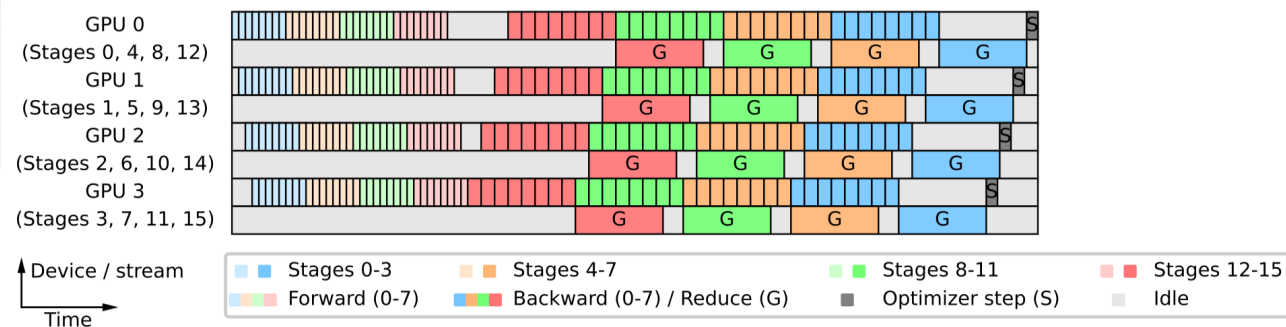
(a) Non-looped pipeline, GPipe schedule (PP_{gpipe}): large bubble, poor overlap



(b) Non-looped pipeline, 1F1B schedule (PP_{1f1b}): large bubble, poor overlap



(c) Looped pipeline, depth-first schedule (PP_{DF}): small bubble, moderate overlap



(d) Looped pipeline, breadth-first schedule (PP_{BF}): small bubble, best overlap

Lamy-Poirier, J., 2023. Breadth-First Pipeline Parallelism. *Proc. of MLSys 2023*.

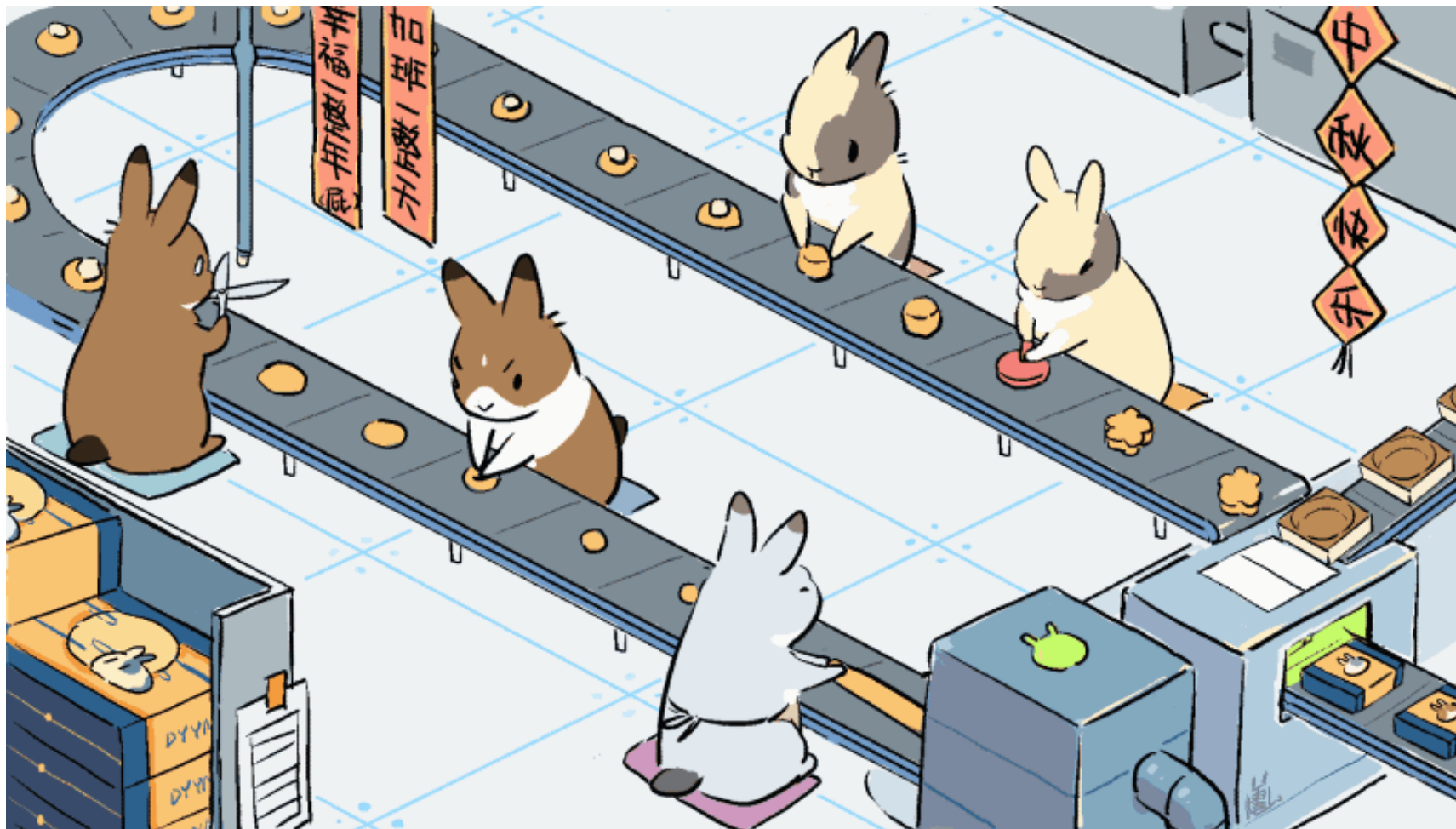




Part I. 为什么要学习流水线?

***Part II.* 什么是流水线?**

生活中的流水线



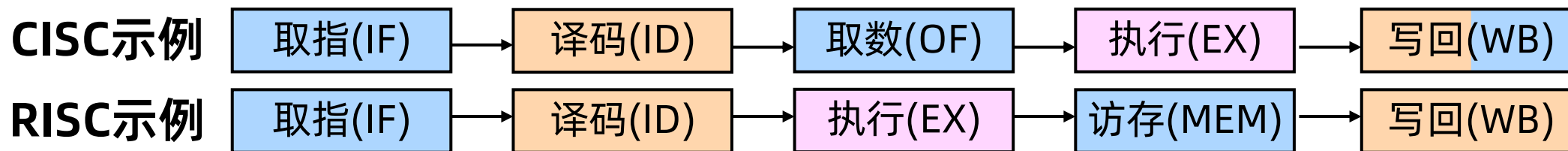
流水线的工作原理



• 基本思想

- 指令执行过程分为多个阶段 ←基础 (分割)
- 各个阶段可以同时处理不同指令的操作 ←效果 (重叠)

• 基本组成：所有功能段按顺序组成



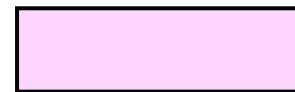
图例



内存访问



寄存器访问



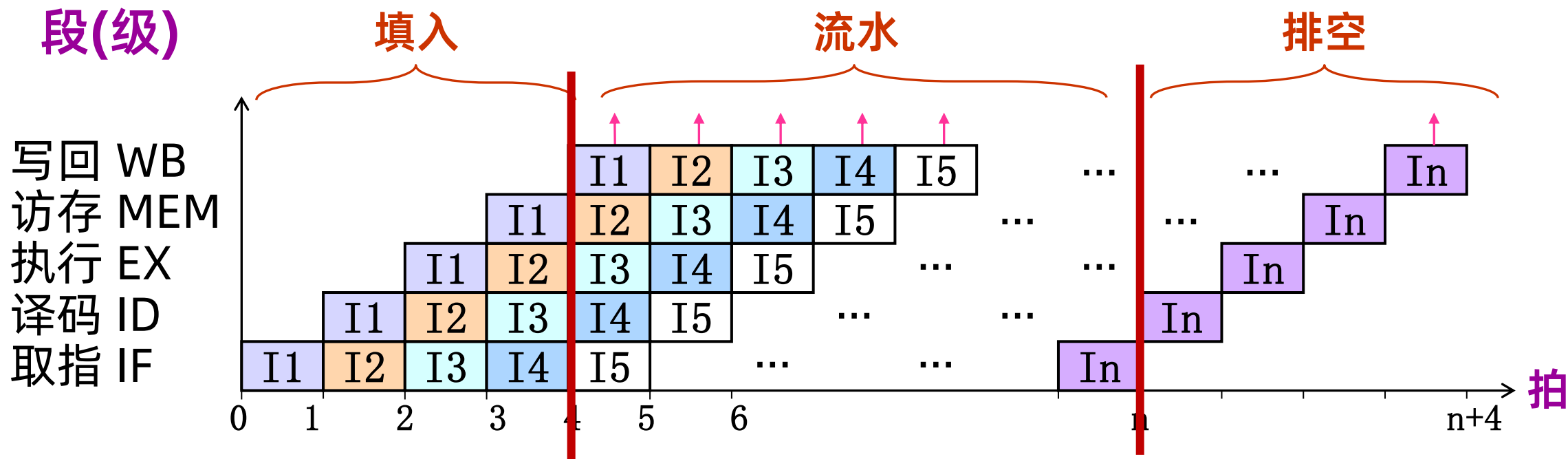
计算过程



流水线的工作原理



- **工作原理：** 每条指令按序通过各个段，不同指令执行过程重叠



- **程序执行时间：** $T_{\text{串行}} = n \cdot (m\Delta t)$, $T_{\text{流水}} = m\Delta t + (n - 1)\Delta t$





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

基本要求→分类方法→性能指标

流水线组成的基本要求

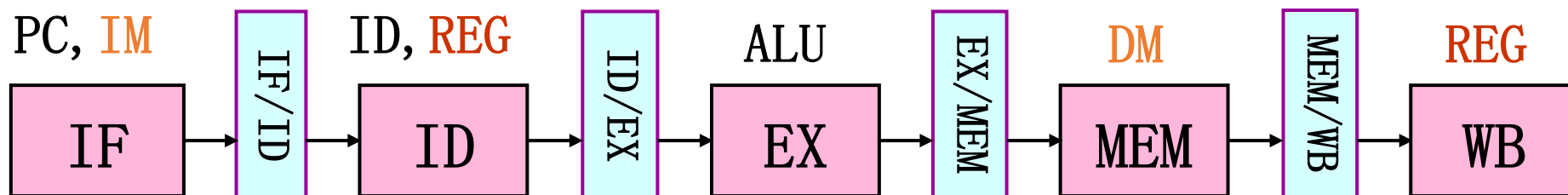
#1 段操作相互独立

#2

#3



- 要求#1: 各个段操作相互独立 ←可以重叠所需
 - 方法: 各个段的源数据来自时序部件、结果存到时序部件
 - 实现: 增设段间寄存器, 用作段的源数据/结果部件



思考: 需保存哪些信息? 各在哪些段保存? 段结束时保存如何实现?

答: 数据/地址/命令, 产生→使用间**所有段**, 采用边缘触发方式

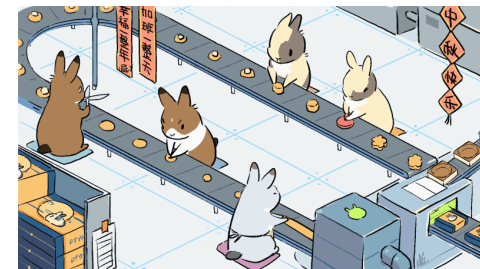
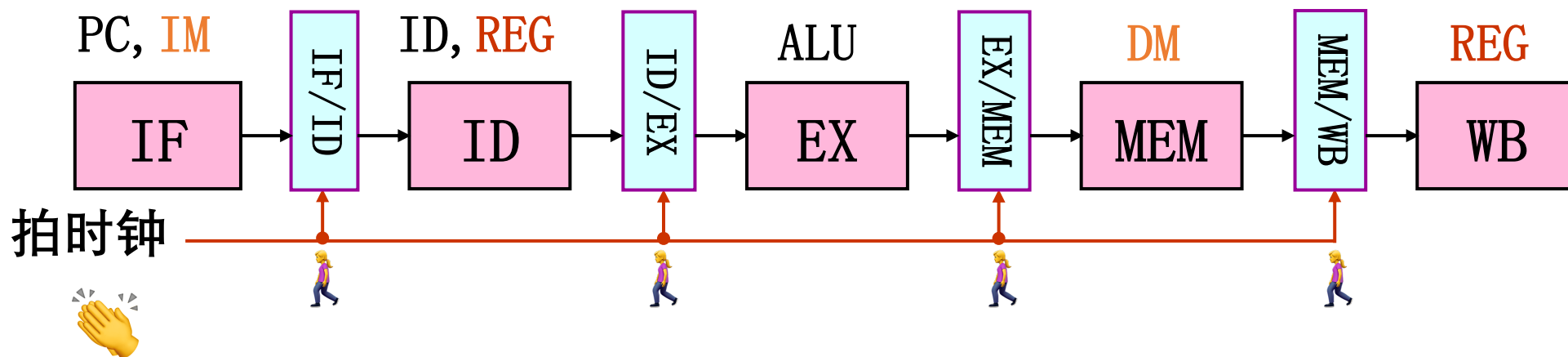


流水线组成的基本要求 (Cont'd)

- #1 段操作相互独立
- #2 各流水段的操作同步
- #3



- 要求#2: 各流水段的操作**同步** ←连续重叠所需
 - 实现: 设置**公共时钟**, 拍时钟周期 = $\max\{\text{段}i\text{时长}\}$
 - ↳ 一段间寄存器同时写($\approx$ 操作同步)



流水线组成的基本要求 (Cont'd)

#1 段操作相互独立

#2 各流水段的操作同步

#3 各流水段的操作无冲突



• 要求#3: 各流水段的操作**无冲突** ←保证结果正确

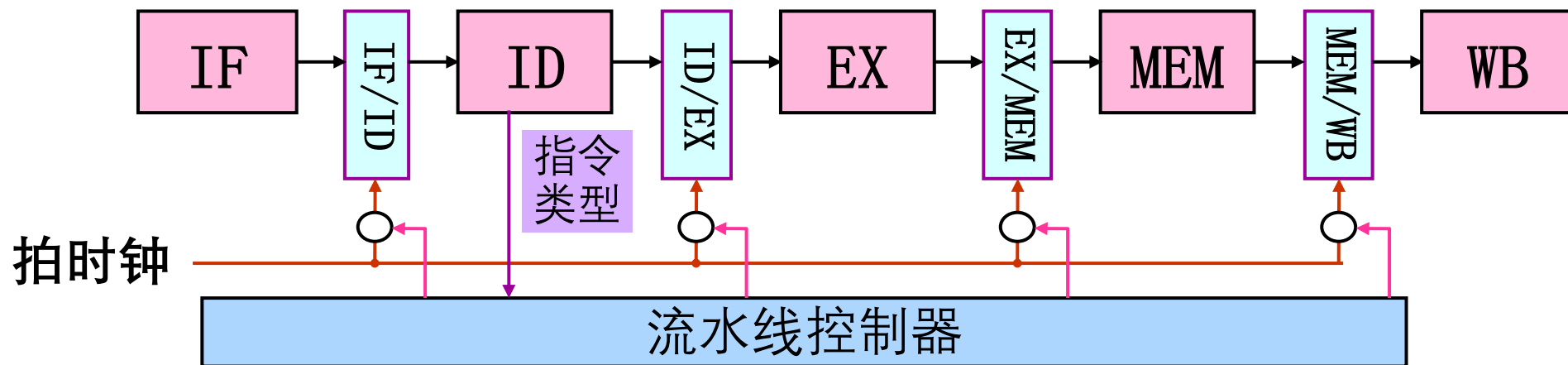
• 冒险 (Hazard): 指流水线因某些原因**无法正确执行**后续指令的现象

• 冒险类型: 结构冒险、数据冒险、控制冒险

(部件复用) (源-目OPD相关) (分支指令)

• 实现: 增设**部件**及**控制机制**, 解决冒险

```
I1: R2 ← (R1) + 5  
I2: PC ← I1 when (R2) = 6  
I3: R3 ← (R1) + 5
```





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

基本要求→**分类方法**→性能指标

#1 段操作相互独立

#2 各流水段的操作同步

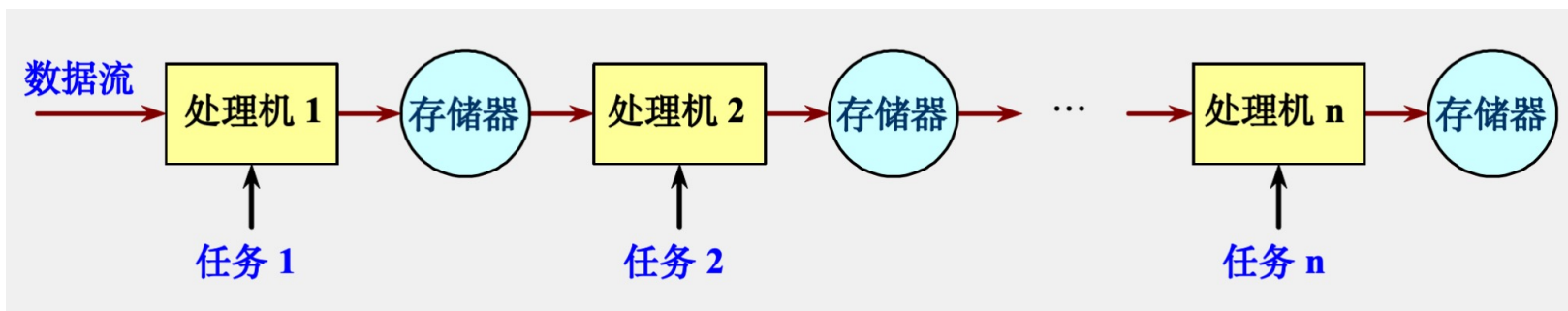
#3 各流水段的操作无冲突

流水线的分类



- 按处理级别分类

- 操作级：把处理机的算术逻辑部件分段流水
 - 浮点加：求阶差→对阶→尾数相加→规格化
- 指令级：把指令的解释执行过程分段流水
 - 四级指令流水线：取指→译码→执行→写回
- 处理机级：把多台处理机串行连接起来，对同一数据流进行处理

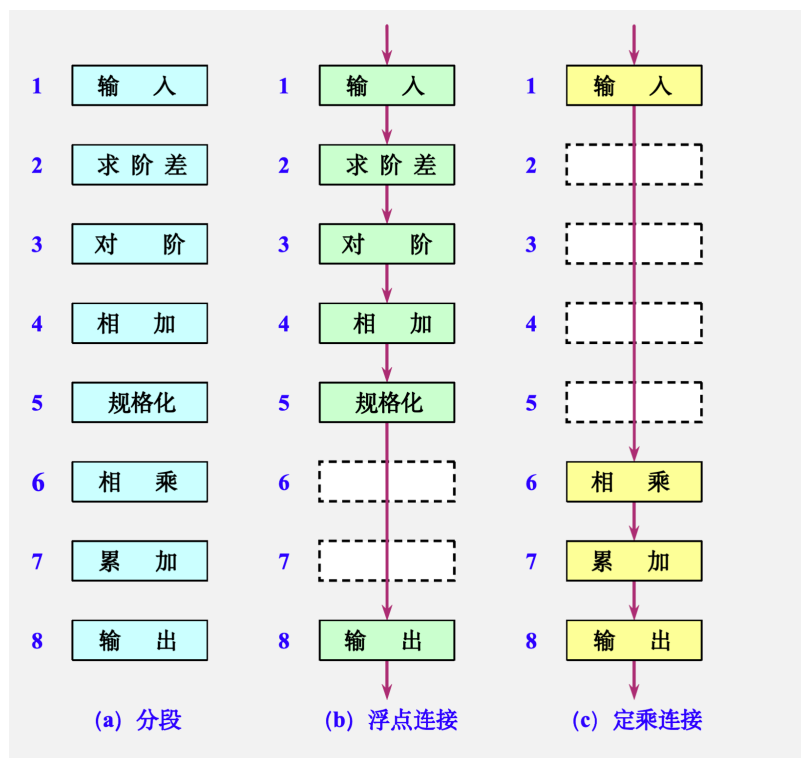
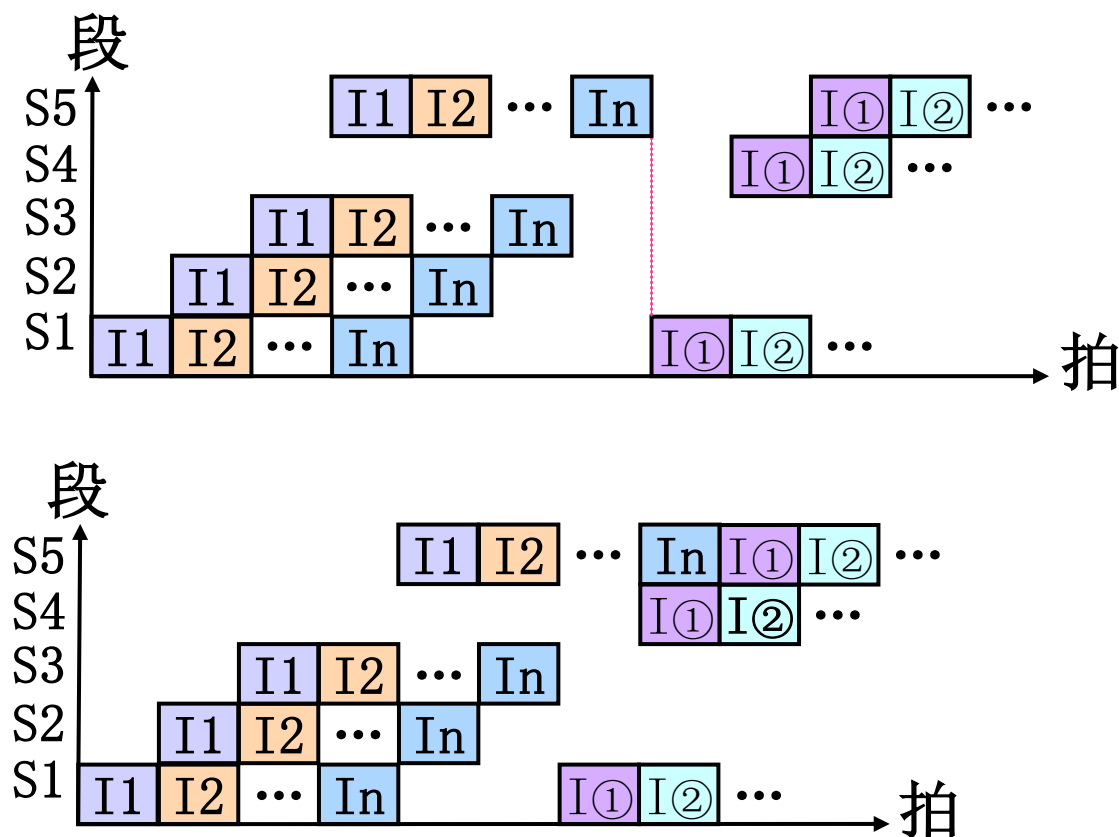


流水线的分类 (Cont'd)



- 按功能分类：单功能流水线、多功能流水线
- 按工作方式分类：静态流水线、动态流水线

动态→多功能
单功能→静态

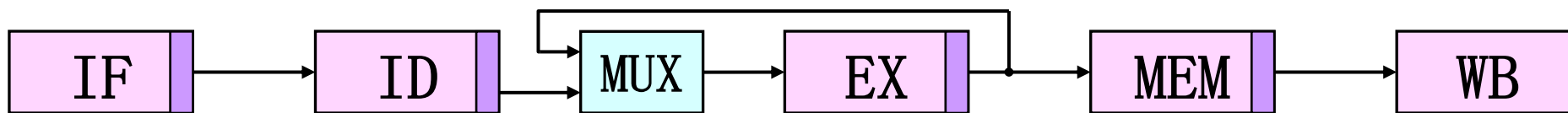


流水线的分类 (Cont'd)



- **按结构分类：**

- 线性流水线：增加部件，减少控制开销
- 非线性流水线：省部件，增加控制开销



- **按流入/流出次序分类：** 顺序流水线、乱序流水线





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

基本要求→分类方法→**性能指标**

- #1 段操作相互独立
 - 操作级、指令级、处理机级
- #2 各流水段的操作同步
 - 单功能、多功能
- #3 各流水段的操作无冲突
 - 静态、动态
 - 线性、非线性
 - 顺序、乱序

流水线的性能指标

提高流水线性能:

1. 减少 Δt
- 2.
- 3.



• 吞吐量 (Throughput) : $TP = \text{任务数}_n / \text{处理}_n \text{个任务总时间}$

• 实际吞吐率: $TP = \frac{n}{T_{\text{流水}}} = \frac{n}{m\Delta t + (n-1)\Delta t}$, $\Delta t = \max\{\Delta t_i\}$

• 最大吞吐率: 当 $n \gg m$ 时, $TP_{\max} = 1/\Delta t$, 即拍长的倒数

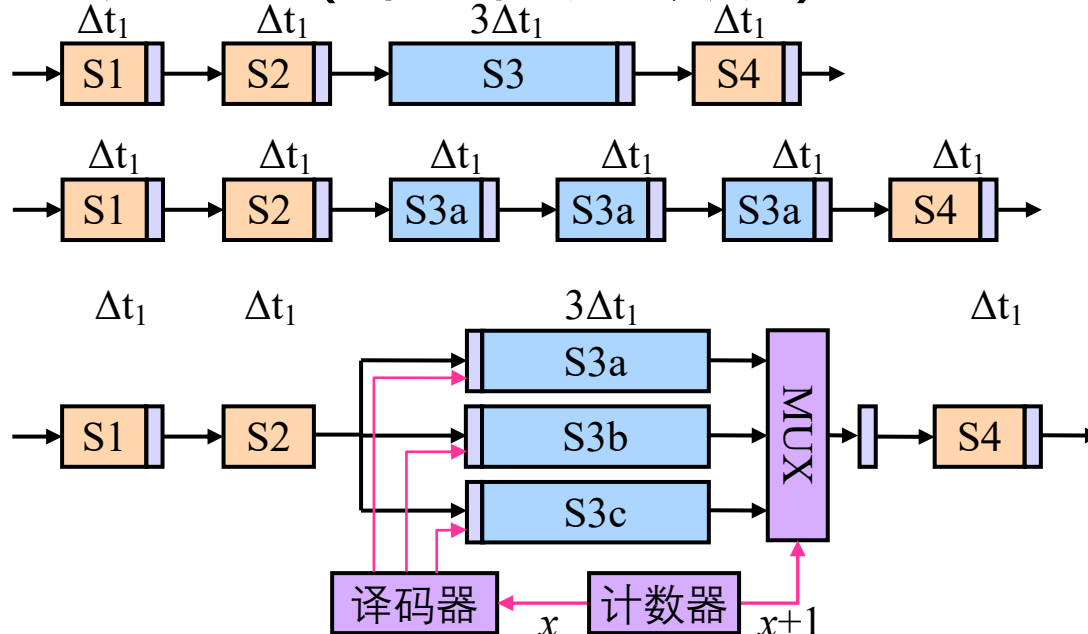
• 优化吞吐率的方法: 减少 Δt (即消除瓶颈段)

瓶颈段处理方案:

① 细分瓶颈段:

② 重复设置部件:
(轮流使用)

(控制方法)



流水线的性能指标 (Cont'd)

提高流水线性能:

1. 减少 Δt
2. 增加段数 m
- 3.



- **加速比 (Speedup)** : $S = \text{串行执行时间} / \text{流水执行时间}$

- 实际加速比:
$$S = \frac{n \cdot m \Delta t}{m \Delta t + (n-1) \Delta t} = \frac{nm}{m+n-1}$$

- 最大加速比: 当 $n \gg m$ 时, $S_{\max} = m$, 即流水线段数

- 优化加速比的方法: 增加段数 m (可减少 Δt)



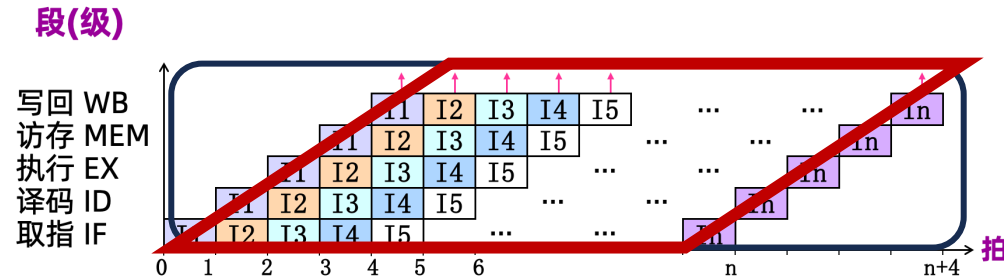
流水线的性能指标 (Cont'd)



- **效率 (Efficiency) :** 即部件利用率
 - $E = \text{部件使用时间} / \text{流水执行总时间}$
 - 各段时间相等, 不考虑冒险: $E = \frac{n}{m+n-1}$
 - 对于线性流水线, $E = S/m = TP \cdot \Delta t$
- $E = \text{任务所占时空区} / m \text{个段总时空区}$
- 最高效率: 当 $n \gg m$ 时, $E_{\max} \rightarrow 1$
- 优化效率的方法: 增加任务数量 n

$$TP = \frac{n}{T_{\text{流水}}} = \frac{n}{m\Delta t + (n-1)\Delta t}$$

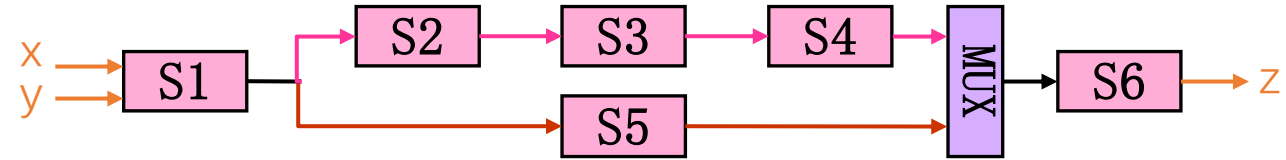
$$S = \frac{n \cdot m\Delta t}{m\Delta t + (n-1)\Delta t} = \frac{nm}{m+n-1}$$



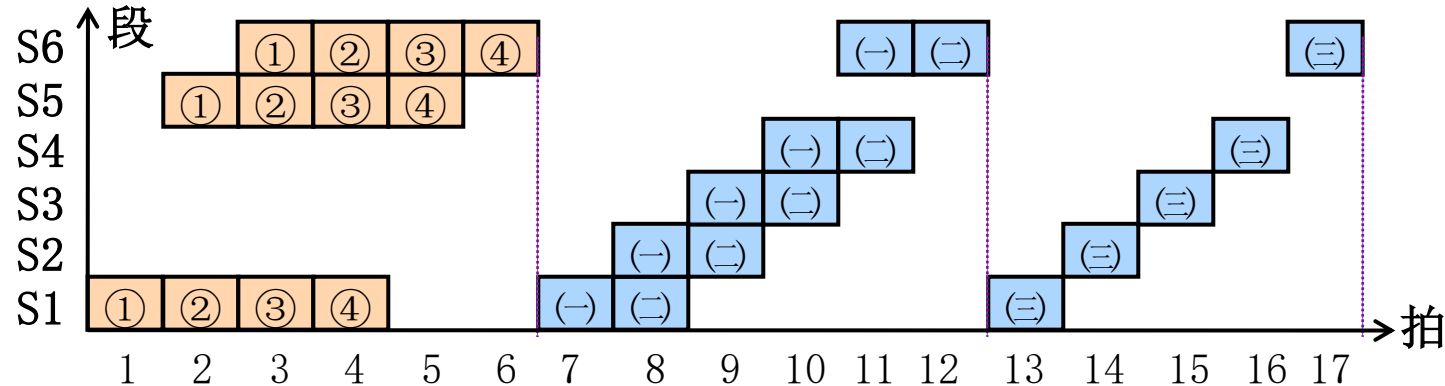
流水线的性能指标 (Cont'd)



- 例：静态、加/乘双功能流水线中，加法功能仅使用3个段，计算并分析实现 $\prod_{i=1}^4 (a_i + b_i)$ 的性能



- 算法：先完成所有加法、再做乘法（减少功能切换）



- 确定计算过程
- 画时空图
- 计算性能

- $TP = 7 / (17 \Delta t)$, $S = (3 \times 4 + 5 \times 3) / 17 \approx 1.588$, $E = 27 / (17 \times 6) = 26.5\%$
- 分析：
 - ①指令数少，填入、排空时间较多 ← 仅拍3和4流水
 - ②静态流水，排空后_后才能切换功能 ← (一)的S1
 - ③数据冒险，消除冒险_后 (如通过阻塞)才继续流水 ← (三)的S1





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

Part IV. 流水线具体设计

相关与冒险 → 数据冒险处理 → 结构冒险处理 → 控制冒险处理

指令相关性与流水线冒险



- **相关 (Dependence)**：指令间存在的依赖关系
 - 数据相关、名相关、控制相关
- **冒险 (Hazard)**：因相关等原因引起的流水异常现象
 - 结构冒险、数据冒险、控制冒险
- **相关是程序的属性，并不一定会带来冒险**
 - 反应了原始的程序顺序（将会影响最终的执行结果）
 - 代表了出现冒险的概率
 - 决定了并行的程度
- **冒险是硬件组成的属性**



数据相关 *Data dependences*



- **(真) 数据相关**
 - 指令 i 产生的数据会被指令 j 使用
 - 指令 j 与指令 k 的数据相关, 指令 k 又和指令 i 的数据相关 (传递性)
- **数据流动是经过寄存器时, 较好检测**
 - 译码阶段直接就能检测出来
- **数据流动是经过存储器时, 比较难检测**
 - 形式不同的地址其有效地址却可能相同

$c = a + b$

 $e = c + d$



名相关 *Name dependence*



- **名相关**

- 两条指令使用了同一个寄存器或内存地址（称为**名**），但并没有真正进行数据交换（也就是对存储地址的重用）

- **反相关 Anti-dependence**

- 指令 j 写的名是指令 i 读的名，并且指令 i 先执行

- **输出相关 Output dependence**

- 指令 i 和 j 写了同一个名，但是两个指令的顺序是不能更改的

$a = b + c$



$b = c + d$

Anti-dependence

$a = b + c$



$a = d + e$

Output dependence



控制相关 *Control dependence*



- **控制相关是指由分支指令引起的相关**
 - 为了保证程序应有的执行顺序，必须严格按控制相关确定的顺序执行
- **典型的程序结构是 “if-then” 结构**

```
if p1 {  
    S1;  
}  
S;  
If p2 {  
    S2;  
}
```

s1 is control dependent on p1

s2 is control dependent on p2

s is neither control dependent on p1 nor p2



相关案例



数据相关：

$I2 - I1, I3 - I2$

反相关：

$I3 - I4$

输出相关：

$I5 - I4$

控制相关：

$I8 - I7$

$I1: R0 \leftarrow [(R6) + 20]$

$I2: R2 \leftarrow (R0) + (R1)$

$I3: R3 \leftarrow (R2) + (R1)$

$I4: R1 \leftarrow (R4) + 25$

$I5: R1 \leftarrow (R0) + 20$

$I6: R0 \leftarrow (R1) - 25$

$I7: JNZ \ I9$

$I8: R1 \leftarrow (R1) + 50$

$I9:$

指令相关性与流水线冒险的关系



- 指令 i 产生的**结果**需要被后续指令 j 用到
 - 可能引起数据冒险
- 指令 i 用到后续指令 j 的**硬件资源**
 - 可能引起结构冒险
- 指令 i **控制**了哪一条指令需要被执行
 - 可能引起控制冒险





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

Part IV. 流水线具体设计

相关与冒险→**数据冒险处理**→结构冒险处理→控制冒险处理

数据相关性与数据冒险



- **RAW (read after write)相关性:** 指令j拿到了旧值

Instr i : $r3 \leftarrow (r1) \text{ op } (r2)$

Instr j : $r5 \leftarrow (r3) \text{ op } (r4)$

(Data-Dependence)

→ RAW 冒险

- **WAR (write after read)相关性:** 指令i拿到了新值

Instr i : $r3 \leftarrow (r1) \text{ op } (r2)$

Instr j : $r1 \leftarrow (r4) \text{ op } (r5)$

(Anti-Dependence)

Q: possible for a typical pipeline ?

- **WAW (write after write)相关性:** 产出错误结果

Instr i : $r3 \leftarrow (r1) \text{ op } (r2)$

Instr j : $r3 \leftarrow (r4) \text{ op } (r5)$

(Output-Dependence)

Q: possible for a typical pipeline ?

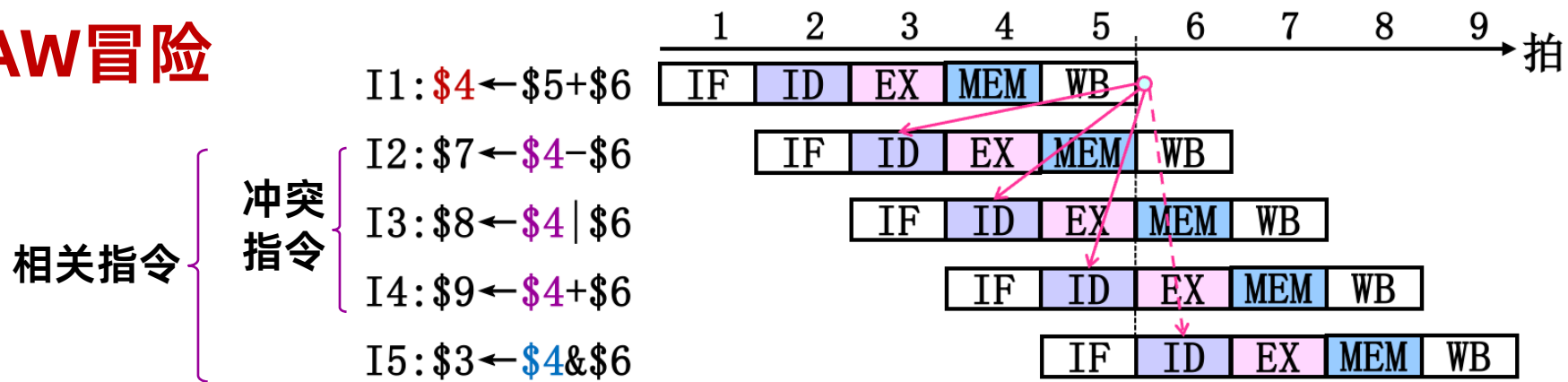


数据冒险



- 产生原因：因重叠执行，指令所需数据不可用

→RAW冒险



- 处理方法：阻塞法、转发法、乱序执行法



数据冒险的处理(1): 阻塞法

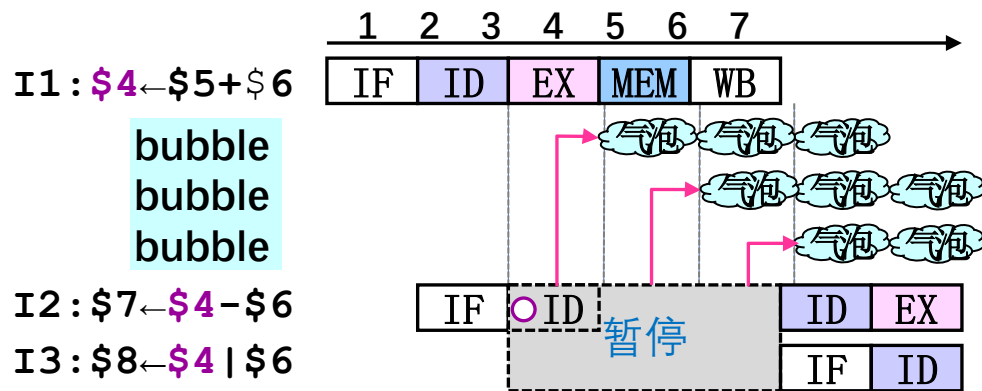
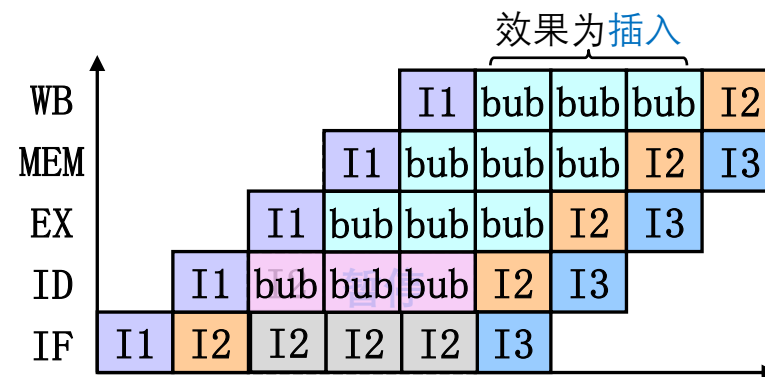


- **阻塞法**: 冲突指令及后续指令**暂停**, 直到**冒险消除**(数据就绪)
- **冒险消除时要求**: IF/ID.REG(段间REG)及PC不变, 其余段间REG为气泡
- **暂停方法**: ID段**产生**气泡(空操作uOPCmd), ←锁步机制2

后续段的控制源

IF段**暂停**(不写IF/ID及PC)

←锁步机制1



思考①: ID产生气泡, IF不暂停行吗?

RAW消除时, IF/ID.IR已不是冲突指令

思考②: 为何不在IF产生气泡?

RAW消除时, IF/ID.IR为nop指令(非冲突指令)

思考③: ID段产生气泡的方法?

用MUX选择Cmd(当前/nop指令),写ID/EX.Cmd

思考④: 如何得到停顿拍数?

ID段每拍比较(IF/ID.IR.Rs ~ 后续段间 REG.Rd), 直至匹配失败(冒险消除)

• **停顿拍数**: 数据欲读-可读的相差拍数

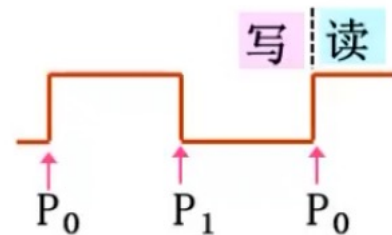
阻塞法处理案例



例1: MIPS流水线中, 写GPRs放在后半拍完成(WB的下一拍才能读出所写数据)。

现有如下MIPS指令序列:

```
I1: add $4, $5, $6      ; $4 ← $5 + $6
I2: sub $7, $4, $6      ; $7 ← $4 - $6
I3: or  $8, $4, $6      ; $8 ← $4 | $6
I4: sw  $6, 20($4)      ; M[$4 + 20] ← $6
I5: lw  $9, 20($8)      ; $9 ← M[$8 + 20]
```



问: (1)哪些指令之间存在RAW冒险?

冲突指令放在前面

(2)采用阻塞法处理RAW冒险时, 指令序列执行时间?

解: (1)RAW冒险有: I2-I1、I3-I1、I4-I1, I5-I3

I2-I1冒险需停 **3** 拍, I3-I1、I4-I1冒险各需停 **0** 拍;

I5-I3冒险需停 **2** 拍;

随I2-I1自动消除

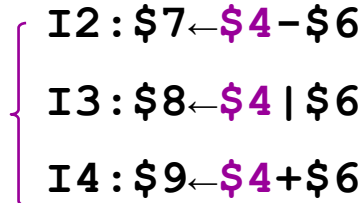
执行时间 = $T_{\text{理想流水}} + T_{\text{停顿}}$

$$= [5t + (5-1)t] + (3+2)t = 14t$$

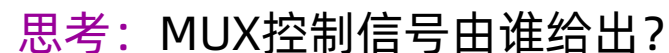
I1	IF	ID	EX	MEM	WB	
:		IF	ID	ID	ID	ID
I2			IF	IF	IF	IF
:						
I3						
:						



- ## ←可减少冲突，简化实现



- **停顿拍数：**可转发时=0拍，否则=阻塞法



流水线控制器CU

例2: 续例1, RAW冒险采用转发法处理, 写GPRs放在前半拍完成, 下列情况下, 指令序列执行时间分别是多少?



(1)有EX→EX、MEM→EX转发线路 (2)仅有EX→EX转发线路

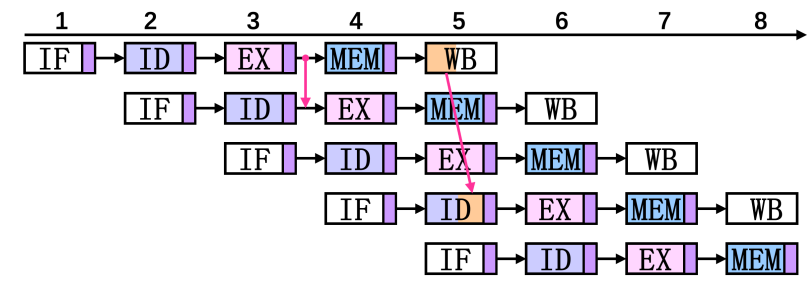
解: RAW冒险有: I2-I1、I3-I1、I4-I1, I5-I3

(1)I2-I1冒险停顿 0 拍; I3-I1冒险停顿 0 拍; (可转发)
 I4-I1冒险停顿 0 拍; I5-I3冒险停顿 0 拍; (同一拍写→读)

执行时间 = $[5t + (5-1)t] + (0+0+0+0)t = 9t$

(2)I2-I1冒险停顿 0 拍; I3-I1冒险停顿 1 拍; (ID读I1_WB结果)
 I4-I1冒险停顿 0 拍; I5-I3冒险停顿 1 拍; (同I3-I1)

执行时间 = $[5t + (5-1)t] + (0+1+0+1)t = 11t$



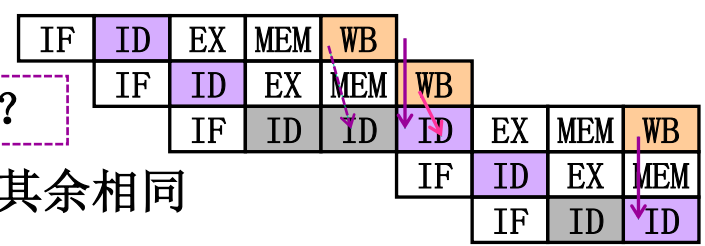
I1: \$4 ← \$5 + \$6
 I2: \$7 ← \$4 - \$6
 I3: \$8 ← \$4 | \$6
 I4: M[\$4+20] ← \$6
 I5: \$9 ← M[\$8+20]

思考①: 写GPRs放在后半拍时, 小题(2)结果如何?

I3-I1、I5-I3各停顿2拍, 其余相同

思考②: I3为\$8 ← \$4 | \$7时, 小题(2)结果如何?

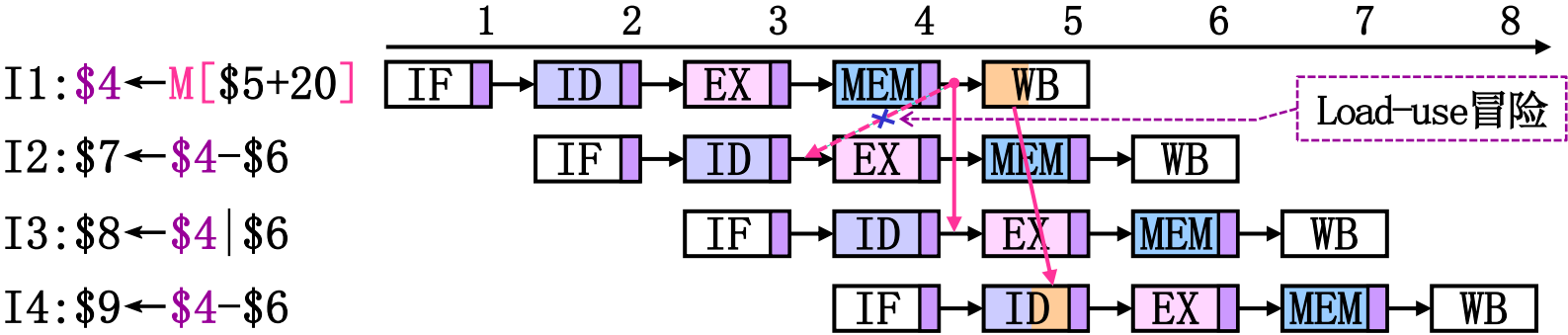
I3需同时取2个OPD(无MEM-EX转发)、停顿2拍, 其余相同



I1: \$4 ← \$5 + \$6
 I2: \$7 ← \$4 - \$6
 I3: \$8 ← \$4 | \$7
 I4: M[\$4+20] ← \$6
 I5: \$9 ← M[\$8+20]



load-use冒险：无法用转发法处理的RAW冒险



lw指令引起
仅紧邻指令[产生比使用迟1拍]

处理方法：阻塞法(插入气泡)，或软件方法(插入nop指令)

练习：MIPS流水线中，写GPRs放在后半拍实现，欲执行下列代码：

```
I1: add $4, $5, $6      ; $4 ← $5 + $6
I2: sub $7, $4, $6      ; $7 ← $4 - $6
I3: or  $8, $4, $6      ; $8 ← $4 | $6
I4: sw  $6, 20($4)      ; M[$4 + 20] ← $6
I5: lw  $6, 20($8)      ; $6 ← M[$8 + 20]
```

- (1) I2-I1、I3-I1、I4-I1, I5-I3;
- (2) I2-I1停3拍, I3-I1、I4-I1停0拍(随I2-I1消除), I5-I3停2拍; $T = [5t + (5-1)t] + (3+2)t = 14t$;
- (3) I2-I1停0拍(EX-EX线路), I3-I1停2拍, I4-I1停0拍(随I3-I1消除), I5-I3停2拍; $T = [5t + (5-1)t] + (2+2)t = 13t$;

```
I3: or  $8 ← $4 | $6
I5: lw  $6 ← M[$8 + 20]
I4: sw  M[$4 + 20] ← $6
```

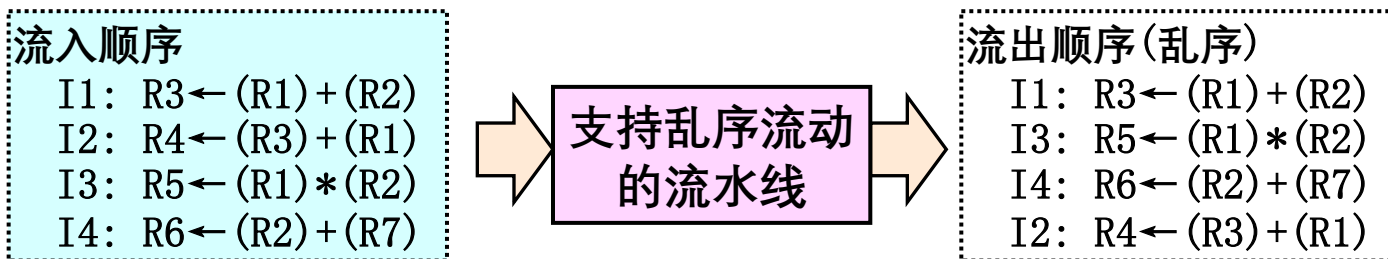
- (1) 哪些指令之间存在RAW冒险?
- (2) RAW冒险用阻塞法处理时，代码的执行时间?
- (3) RAW冒险用转发法处理(线路为EX→EX)时，代码的执行时间?
- (4) I4和I5对调后，小题(3)的结果? (4) I2-I1停0拍, I3-I1停2拍, I5-I3停0拍, I4-I5停3拍, $T = [5t + (5-1)t] + (2+3)t = 14t$ 。



数据冒险的处理(3): 乱序执行法



- **乱序执行法:** 冲突指令停顿, 后续无RAW冒险的指令可优先执行



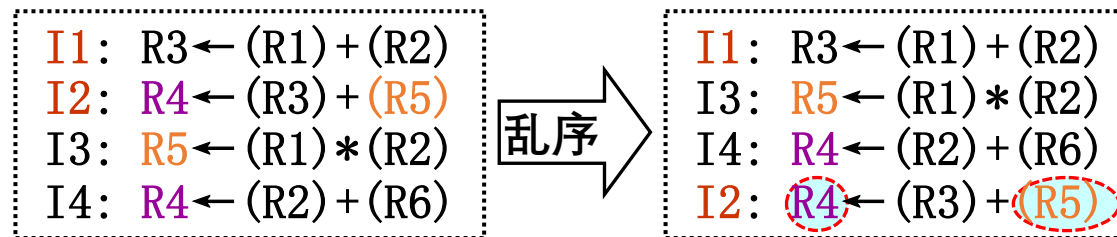
- **实现方法:** 增设指令窗口、采用动态调度方法

- **停顿拍数:** 0拍

- **新增冒险类型:**

- WAR冒险、WAW冒险

- 处理方法: 动态调度方法 (第4章)





Part I. 为什么要学习流水线?

Part II. 什么是流水线?

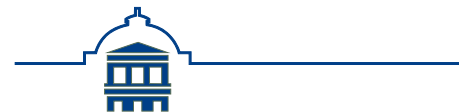
Part III. 如何设计流水线?

Part IV. 流水线具体设计

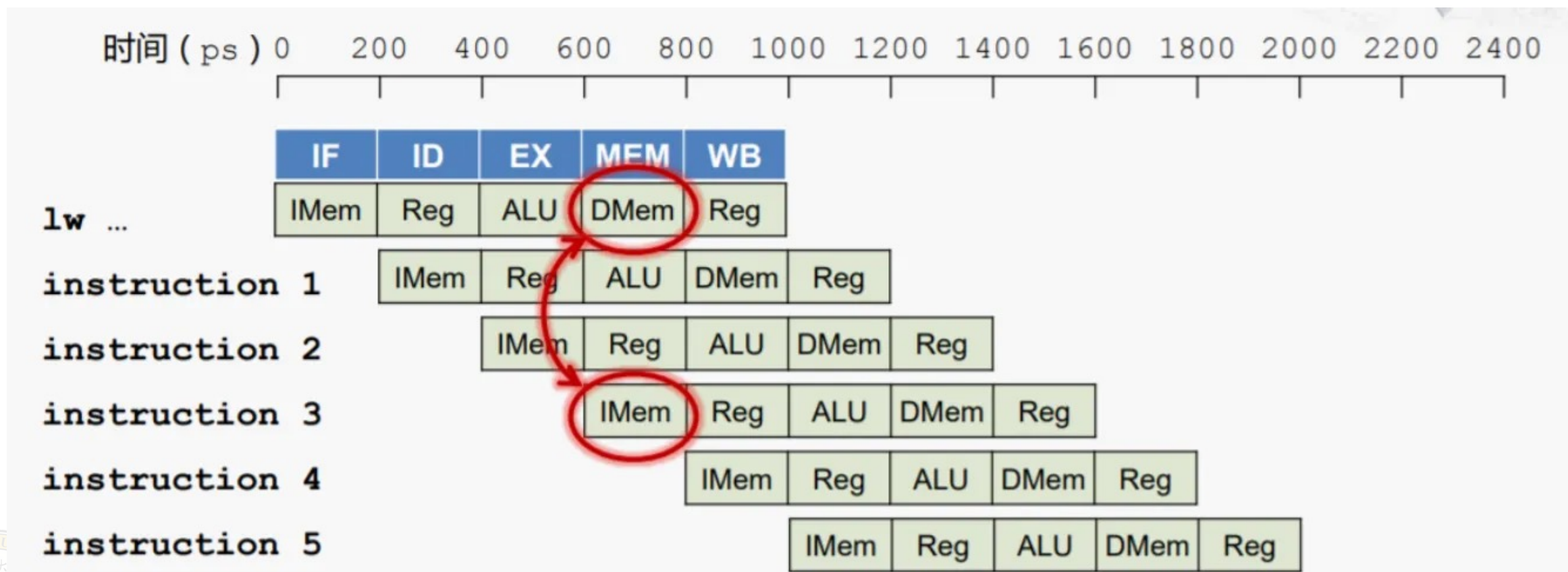
相关与冒险→数据冒险处理→**结构冒险处理**→控制冒险处理

- 阻塞法
- 转发法
- 乱序执行法*

结构冒险



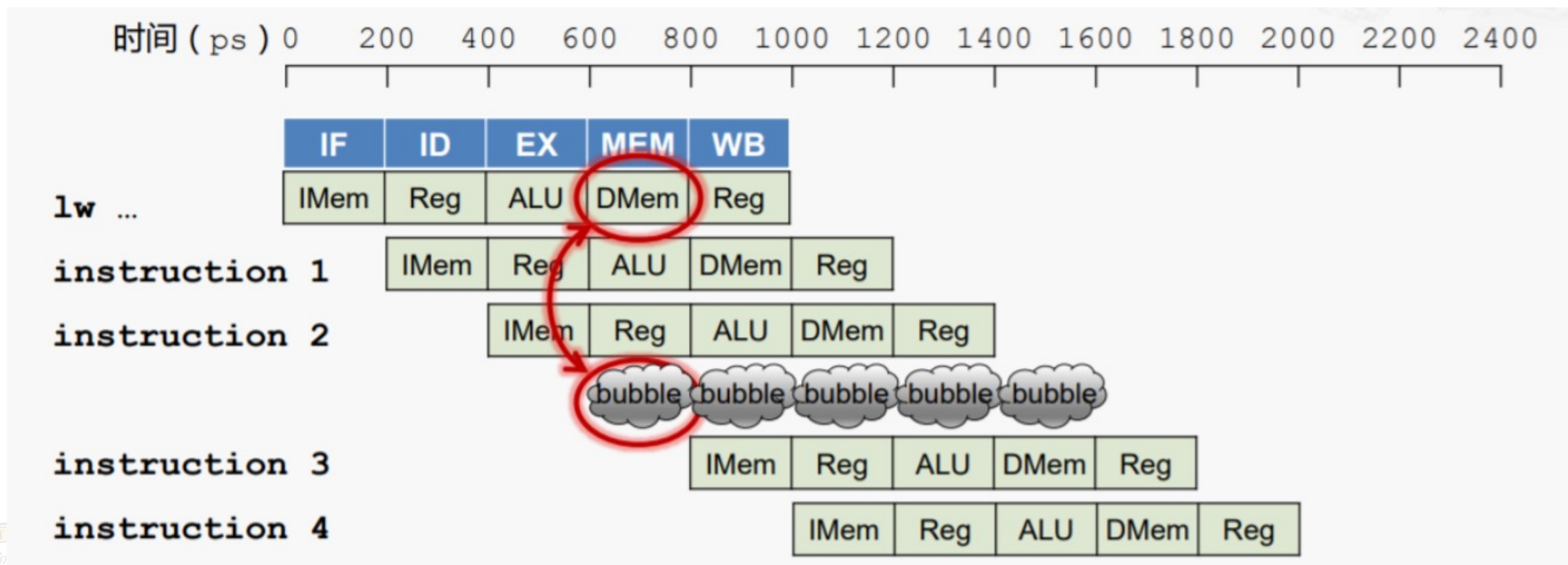
- 产生原因：争用硬件资源（部件或路径）
- 例：访存冲突



结构冒险的处理(1): 阻塞法 (分时使用资源)



- 避免资源争用, 适用于**低频率**冲突
 - 如: 冲突时使流水线停顿 (Stall), 后续指令暂停



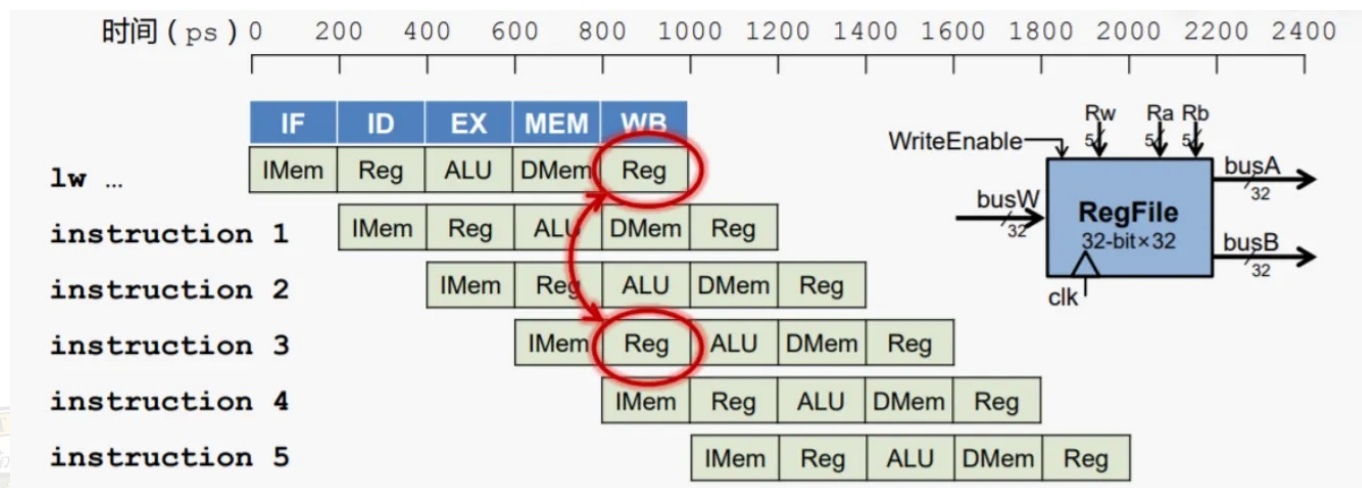
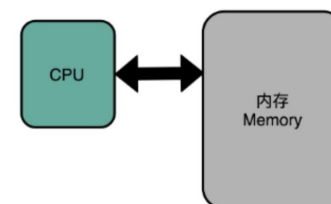
结构冒险的处理 (2): 重复设置资源



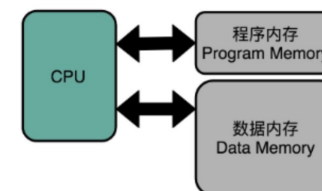
- 消除资源争用，适于**高频率冲突**
- 哈佛架构（代表性案例）：
 - 华为鲲鹏920处理器
 - 指令存储与数据存储独立编址、独立访问
- ①部件不复用、使用时间唯一
 - 如：存储采用哈佛架构、PC+4增设地址加法器



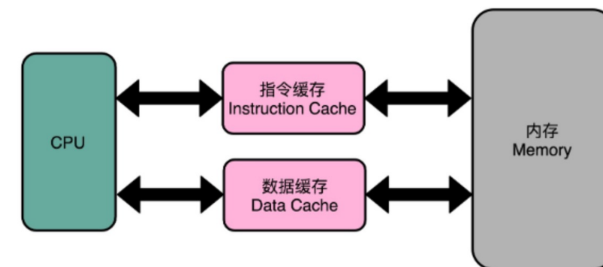
- ②部件互联采用专用结构



冯·诺伊曼架构/普林斯顿架构
Princeton Architecture



哈佛架构
Harvard Architecture



现代CPU的混合架构
Hybrid Architecture



Part I. 为什么要学习流水线?

Part II. 什么是流水线?

Part III. 如何设计流水线?

Part IV. 流水线具体设计

相关与冒险→数据冒险处理→结构冒险处理→**控制冒险处理**

- 阻塞法
- 转发法
- 乱序执行法*
- 阻塞法
- 重复设置资源

控制冒险

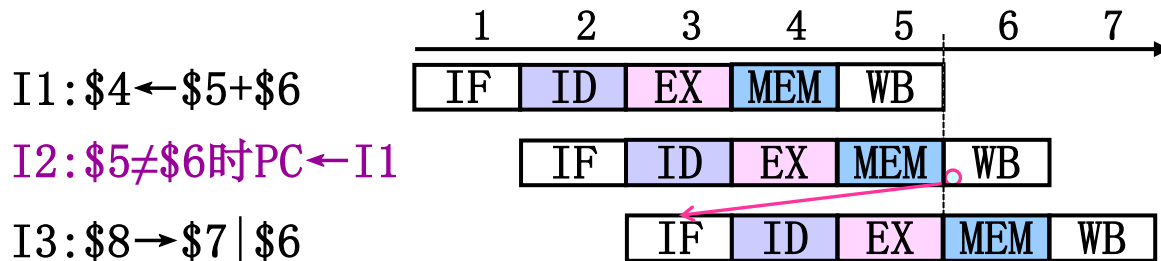


- 正常流水要求：IF段产生下条指令地址 (如PC+4)

- 冒险产生原因：因指令执行顺序改变，引起流水线停顿

↑ $NPC \neq PC+4$

↑ IF段无法给出



IF	$IR \leftarrow IM[PC], NPC \leftarrow (PC) + 4$
ID	$A \leftarrow R[rs], B \leftarrow R[rt], Imm \leftarrow ExtU(imme)$
EX	$C \leftarrow beq \cdot ((A) = (B)) ? 1 : 0, T \leftarrow (NPC) + Imm$
MEM	$PC \leftarrow C ? T : NPC$

- 处理方法：阻塞法、延迟分支法、分支预测法



控制冒险的处理(1): 阻塞法

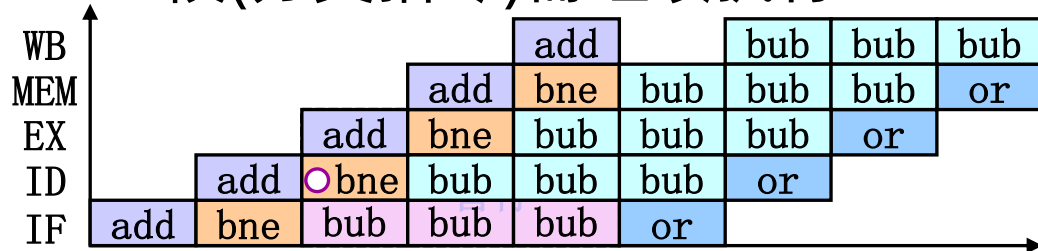


***阻塞法：**分支指令的后继指令 **暂停**，**直到**冒险消除

冒险消除时要求— PC为分支结果，各段间REG均为气泡

暂停方法— IF段**产生**气泡(nop指令)、**PC为**分支结果 ←锁步机制3

└←ID段(分支指令)需继续执行 └←最后一拍写(之前不变)



思考①：改为ID下拍产生气泡、IF暂停，可以吗？

可以，但很麻烦，且IF**末拍**需产生气泡

停顿拍数— 从ID段到指令执行完的拍数(= $T_{bne} - 1$)

优化方法：①尽早判断是否转移

②尽早计算转移目标地址

} 需同时进行



锁步机制对比



- **数据冒险-阻塞法锁步机制**

- IF段暂停(不写IF/ID及PC)
- ID段产生气泡(空操作uOPCmd)

←锁步机制1

←锁步机制2

- **控制冒险-阻塞法锁步机制**

- IF段产生气泡(nop指令)、PC为分支结果

←锁步机制3

思考②：与数据冒险之锁步机制的差别？

气泡产生时机(非ID)/方式(非uOPCmd)、IF(无需暂停)

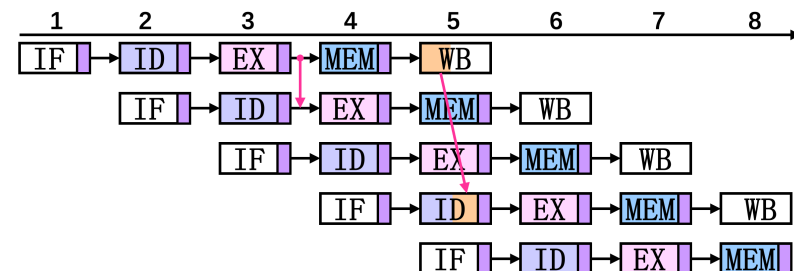


控制冒险—阻塞法案例



例3： MIPS流水线中，有EX→EX的转发线路，分支指令在MEM段写PC。现有如下指令序列：

	addi \$4, \$5, 100	; I1: \$4←\$5+100
L1:	add \$8, \$6, \$7	; I2: \$8←\$6+\$7
	sw \$8, 20(\$6)	; I3: M[\$6+20]←\$8
	addi \$5, \$5, 1	; I4: \$5←\$5+1
	bne \$5, \$4, L1	; I5: \$5≠\$4时PC←L1
	addi \$9, \$9, 10	; I6: \$9←\$9+10



- (1) 哪些指令之间存在RAW冒险？
- (2) 控制冒险采用阻塞法处理时，指令序列的执行时间？

解： (1) RAW冒险有：I3-I2、I5-I4 （I5-I1有相关、无冒险）

(2) I3-I2冒险停顿 **0** 拍，I5-I4冒险停顿 **0** 拍；

控制冒险用阻塞法处理时，I5每次使流水线停 **3** 拍，

$$\text{执行时间} = [5t + (402-1)t] + [0t + (0t + 3t) * 100] = 706t$$



-
- Diagram illustrating a 5-stage pipeline (IF, ID, EX, MEM, WB) with a branch instruction (bne) and its delay slots. The branch instruction is in the ID stage, and its delay slots are the EX, MEM, and WB stages. The delay slots are filled with 'I6' (Instruction 6). The total delay is 7 cycles.
- Delay slots calculation:
- $$T_{\text{延迟分支}} = T_{\text{分支}} + S_{\text{延迟槽}} = 7 \text{ 拍}$$



$$\begin{aligned} T_{\text{延迟分支}} &= T_{\text{分支}} + S_{\text{延迟槽}} \\ &= 7 \text{拍} \end{aligned}$$

- 延迟槽中均为nop指令时 = 阻塞法
- 延迟槽中含有用指令时 < 阻塞法

难以找到足够的无关指令

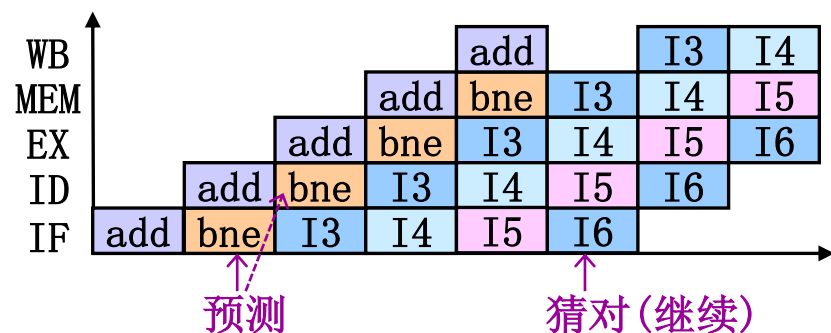
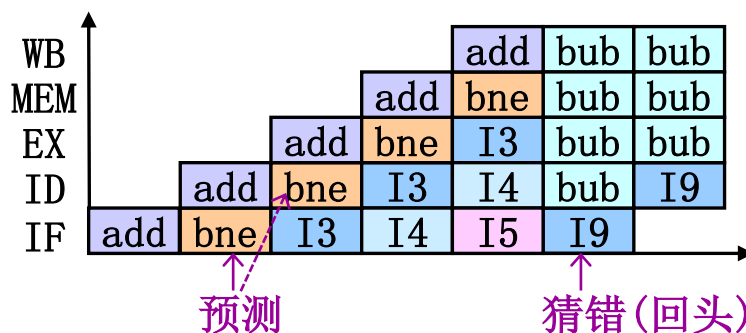
顺序指令 I1~I5 中 与 I6 有相关的指令
转移指令 I6
延迟槽 (I1~I5 中 与 I6 无相关的指令 或 nop 指令)
顺序指令 I7~I9



控制冒险的处理(3): 预测法



- **预测法**: **预测**转移方向(转/不转), 并**执行**该方向的指令
 - 猜对时继续执行后续指令, 猜错时回头执行反方向指令



实现机制:

猜对时**不写**PC,

猜错时**重写**PC、**清空流水线**(产生气泡)

- **预测策略**: 静态预测_[盲猜]/动态预测_[基于转移历史]
 - 静态预测-不转移: **无**先决条件(PC+1已知), 可在IF段预测
 - 静态预测-转移: 需先**计算**目标地址, 最早在ID段预测
 - 动态预测: **已保存**转移历史&目标地址, 可在IF段预测
- **停顿拍数**: 猜对时0/1拍, 猜错时 = 阻塞法(到指令执行完为止)
(ID预测时 = 1) └← 无论是IF预测或ID预测 ←┐

一般方式:

动态预测+静态预测
(首次执行时)

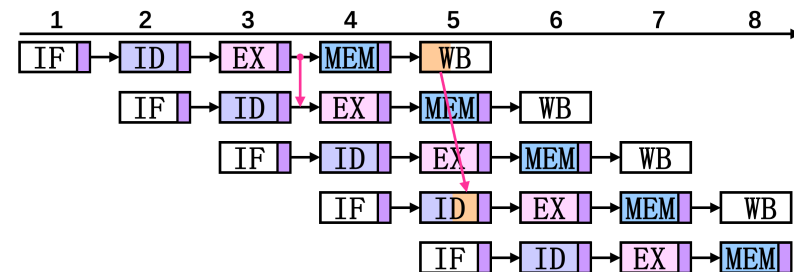
第4章讨论





例4： MIPS流水线有EX段→EX段转发线路， bne指令在ID段计算分支目标地址(专用部件)、在MEM段写PC。现有如下指令序列：

L1: add \$8, \$6, \$7 ; I2: \$8 ← \$6 + \$7, 假设\$4 = 100、\$5 = 0
sw \$8, 20(\$6) ; I3: M[\$6 + 20] ← \$8
addi \$5, \$5, 1 ; I4: \$5 ← \$5 + 1
bne \$5, \$4, L1 ; I5: \$5 ≠ \$4时PC ← L1
addi \$9, \$9, 10 ; I6: \$9 ← \$9 + 10



- (1)控制冒险用阻塞法处理时，代码执行时间？
- (2)控制冒险在IF段预测、预测不转移，代码执行时间？
- (3)若在ID段预测、预测转移，则预测的停顿时间？

解： RAW冒险有I1-I2、I5-I4，冒险需停顿0拍(可转发)；

(1)每次控制冒险暂停 3 拍； $T = [5t + (400 + 1 - 1)t] + 3t * 100 = 705t$

(2)I5预测正确 1 次(停 0 拍/次)、预测错误 99 次(停 3 拍/次)；

$$T = [5t + (400 + 1 - 1)t] + 3t * 99 = 702t$$

(3)I5预测正确时停 1 拍/次、错误时停 3 拍/次 (分支执行完即可回头)





恭喜大家通关流水线设计！



我们指令集并行再见！



- **复习**

- 计算机组成原理第5章关于CPU的内容（尤其是单周期与多周期数据通路）

- **书面作业**

- P91： 5、 6
- P92： 11 （1） （2） 、 12

