

计算机系统结构

Computer Architecture

主讲人：陈俭喜

武汉光电国家研究中心

计算机学院

chenjx@hust.edu.cn



第1章 内容回顾

(1) 基本概念：

计算机系统的多级层次模型，计算机系统结构的广义定义与狭义定义，与组织、实现的关系，计算机系统结构的分类

(2) 计算机系统设计的定量原理

- ◆ 加快经常性事件原理
- ◆ Amdahl定律（加速比，定量描述）
- ◆ CPU性能公式：执行时间公式
 - 平均时钟周期数CPI
 - 每秒百万指令数MIPS

一个原理
一个定律
一个公式

(3) 冯·诺依曼结构的特点

(4) 并行性的等级与技术途径

FAST

Throughput easy, latency hard



IO⁵⁰⁰

#1	BOF	INSTITUTION	SYSTEM	STORAGE VENDOR	TYPE	CLIENT NODES	TOTAL CLIENT PROC.	SCORE †	BW (GIB/S)	MD (KIOP/S)
1	ISC23	JNIST and HUST PDSL	Cheelo-1 with OceanStor Pacific	Huawei	OceanFS2	10	9,600	137,100.02	2,439.37	7,705,448.04
2	ISC23	Pengcheng Laboratory	Pengcheng Cloudbrain-II on Atlas 900	Pengcheng Laboratory and Tsinghua University	SuperFS	10	1,200	11,516.36	263.97	502,435.85
3	SC22	Sugon Cloud Storage Laboratory	ParaStor	Sugon	ParaStor	10	2,560	8,726.42	718.11	106,042.93
4	SC22	SuPro Storteck	StarStor	SuPro Storteck	StarStor	10	2,560	6,751.75	515.15	88,491.65
5	SC22	Tsinghua Storage Research Group	SuperStore	Tsinghua Storage Research Group	SuperFS	10	1,200	5,517.73	179.60	169,515.95
6	SC23	Argonne National Laboratory	Aurora	Intel	DAOS	10	2,080	3,748.85	934.00	15,046.98
7	ISC22	National Supercomputing Center in Jinan	Shanhe	PDSL	flashfs	10	2,560	3,534.42	207.79	60,119.50
8	SC21	Huawei HPDA Lab	Athena	Huawei	OceanFS	10	1,720	2,395.03	314.56	18,235.71
9	SC21	Olympus Lab	OceanStor Pacific	Huawei	OceanFS	10	1,720	2,298.69	317.07	16,664.88
10	ISC21	Intel	Endeavour	Intel	DAOS	10	1,440	1,859.56	398.77	8,671.65



Throughput is easy

Latency is hard

Throughput is an engineering problem, latency is a physics problem!

性能公式

$$\text{CPU时间} = \text{指令条数IC} * \text{CPI} * \text{周期时间}$$

• 复杂指令系统CISC

- 减少指令条数IC，使用复杂的指令
- 易编程（汇编语言级），程序代码量少
- 商业上很成功（Intel、AMD）

• 精简指令系统RISC 降低CPI，开发指令级并行

- 减少CPI，使用大量单周期指令
- 增加了指令条数，但并不太多
- 可能减少时钟周期时间
- 技术上胜利者，应用越来越广（ARM，RISC-V）

和讯网 · 1周前

阿里达摩院：RISC-V崛起 未来占有率超25%

【RISC-V 在芯片设计领域崛起，发展前景备受瞩目】在芯片设计领域，指令集是底层技术标准，不...

1分钟读完



ITBEAR科技资讯 · 1周前

国产RISC-V芯片崛起，能否撼动ARM与X86的霸主地位？

在芯片架构的浩瀚星空中，两大巨头ARM与X86犹如... 瞩目。ARM以其在手机芯片领域的绝...

1分钟读完



和讯网 · 1周前

旋极信息：RISC-V芯片发展迅猛 2027年将爆发

【3月1日讯：旋极信息股价创新高，RISC-V 产业发展受关注】2月28日，旋极信息股价盘中触及...

1分钟读完



财联社 · 1周前

“算力门槛下降近20倍”！DeepSeek爆火有望带动RISC-V发展 多家上市...

《科创板日报》2月28日讯（记者 黄心怡）由达摩院举办的2025玄铁RISC-V生态大会上，《科创板日...

5分钟读完



第3章 流水线技术

1. 什么是流水线?
2. 如何评价流水线?
3. 怎样才能使流水线的效率最高?
4. 如何处理流水线的资源争用?



3.1 流水线的基本概念

洗衣为例

Ann, Brian, Cathy, Dave

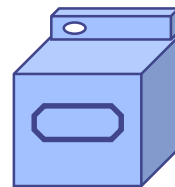
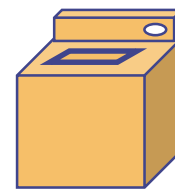
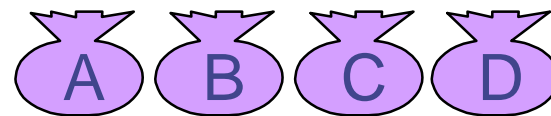
每人进行洗衣的动作

wash, dry, and fold

Washer需要 30 minutes

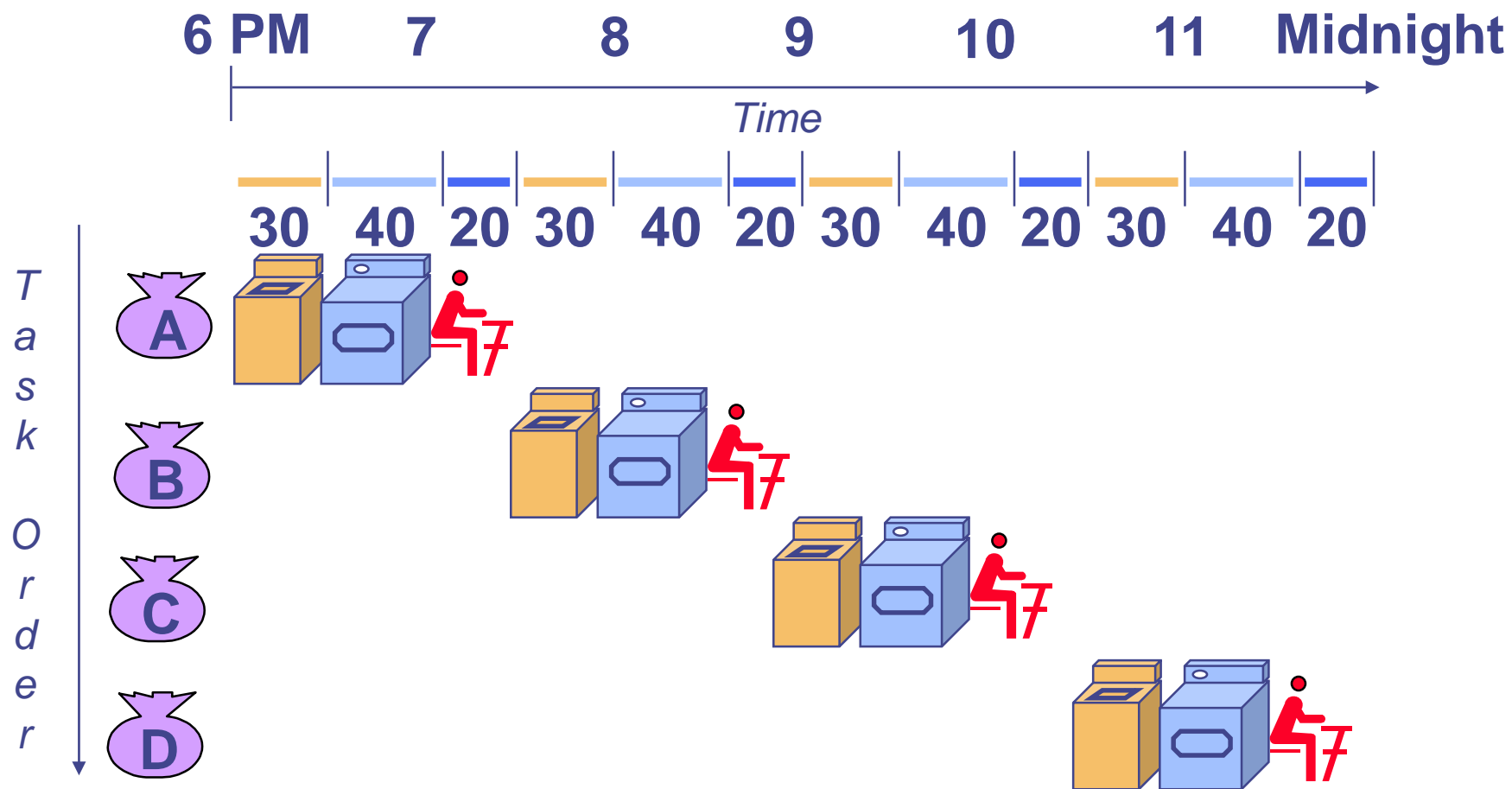
Dryer 需要 40 minutes

Folder 需要 20 minutes



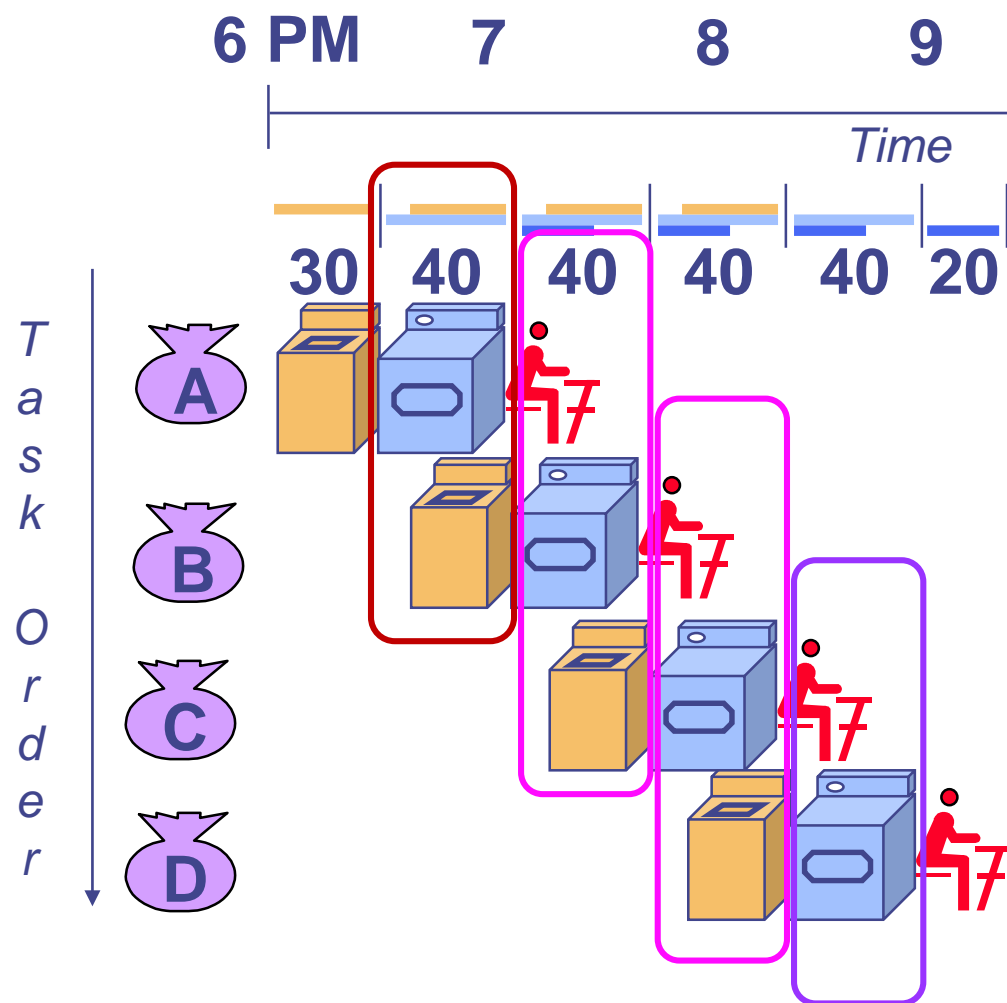
如果让你来管理洗衣店，你会如何安排？

Sequential Laundry (串行方式)



1. 串行方式下，4批衣服需要花费6小时 ($4 \times (30 + 40 + 20) = 360$ 分钟)
2. N批衣服，需花费的时间为 $N \times (30 + 40 + 20) = 90N$
3. 如果用流水线方式洗衣服，则花多少时间呢?

Pipelined Laundry: (Start work ASAP)



串行方式为6小时，N批则为90N分钟

流水方式：

只需 $30+4*40+20=210$ 分 (3.5小时)

如果有N批衣服呢？

所花时间为： $30+N*40+20$ 分钟

假定每一步时间均衡，则比串行方式提高约3倍！

流水方式下，所花时间主要与最长阶段时间有关！

回忆：Load指令的5个阶段

阶段1	阶段 2	阶段 3	阶段 4	阶段5
Ifetch	Reg/Dec	Exec	Mem	Wr

- **Ifetch (取指IF) :** 从指令存储器取指令并计算PC+4
- **Reg/Dec (取数和译码ID) :** 寄存器取数，同时对指令进行译码
- **Exec (执行EX) :** 计算内存单元
- **Mem (读存储器MEM) :** 存储器中读
- **Wr(写寄存器WB):** 将数据写到寄存器中

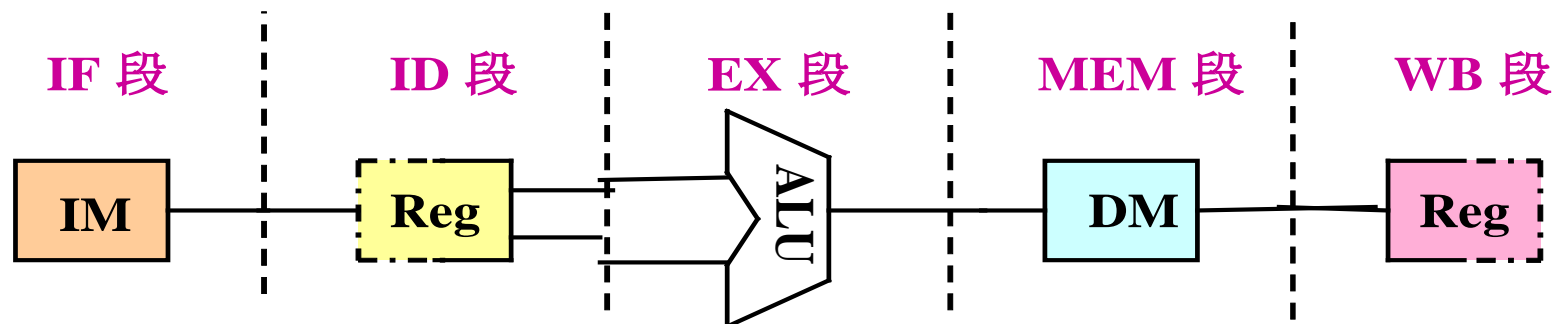
指令的执行过程是否和“洗衣”过程类似？是否可以采用类似方式来执行指令呢？如果可以，要怎么操作？

3.1 流水线的基本概念

2. 流水线技术

- 把一个重复的过程分解为若干个子过程，每个子过程由专门的功能部件来实现。
- 把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其它子过程并行进行。

3. 流水线中的每个子过程及其功能部件称为流水线的级或段，段与段相互连接形成流水线。流水线的段数称为流水线的深度。

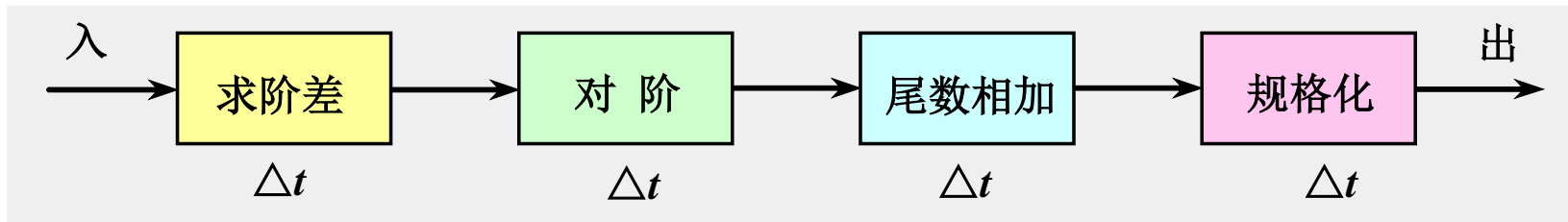


3.1 流水线的基本概念

4. 浮点加法流水线示例

- 把流水线技术应用于运算的执行过程，就形成了
运算操作流水线，也称为部件级流水线。
- 把浮点加法的全过程分解为求阶差、对阶、尾数相加、规格化四个子过程。

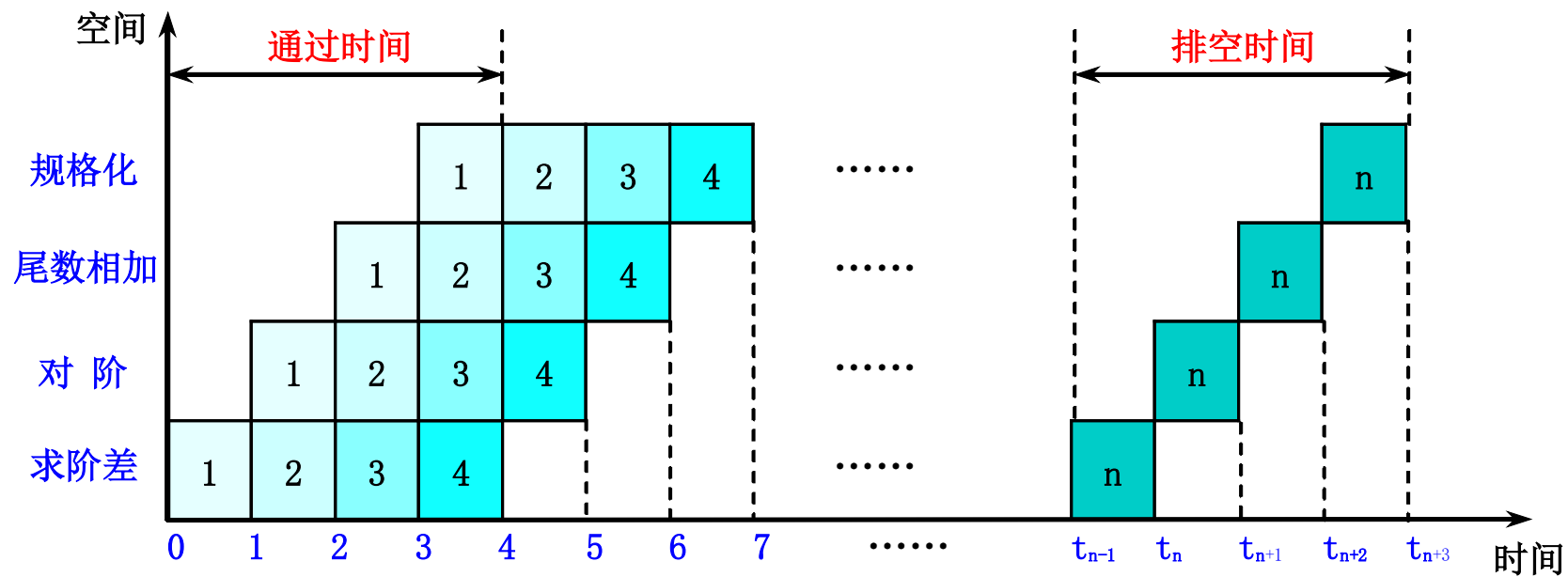
理想情况：速度提高3倍



3.1 流水线的基本概念

5. 时一空图

- 时一空图从时间和空间两个方面描述了流水线的工作过程
- 时一空图中，横坐标代表时间，纵坐标代表流水线的各个段
- 浮点加法流水线的时空图



3.1 流水线的基本概念

6. 流水技术的特点

- 流水线把一个**处理过程分解**为若干个子过程（段），每个子过程由一个专门的功能部件来实现
- 流水线中**各段**的时间应**尽可能相等**，否则将引起流水线堵塞、断流

时间最长的段将成为流水线的瓶颈

- 流水线每一个段的后面都要有一个缓冲寄存器（锁存器），称为**流水寄存器**
 - **作用：**在相邻的两段之间传送数据，以提供后面要用到的信息，并把各段的处理工作相互隔离

3.1 流水线的基本概念

6. 流水技术的特点（续）

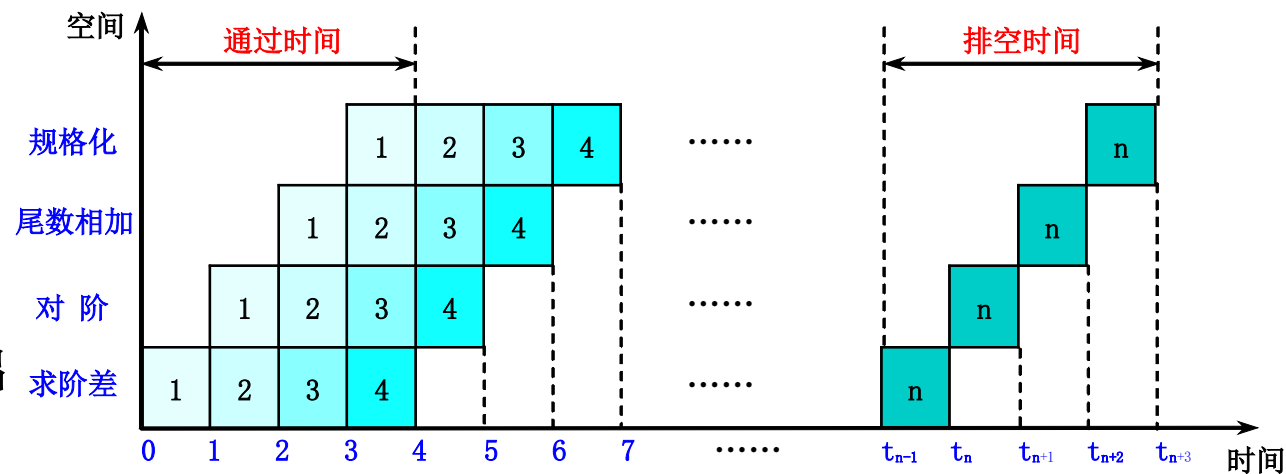
- 流水技术适合于大量重复的时序过程，只有在输入端不断地提供任务，才能充分发挥流水线的效率
- 流水线需要有通过时间和排空时间

□ 通过时间：

第一个任务从进入流水线到流出结果所需的时间

□ 排空时间：

最后一个任务从进入流水线到流出结果所需的时间



3.1 流水线的基本概念

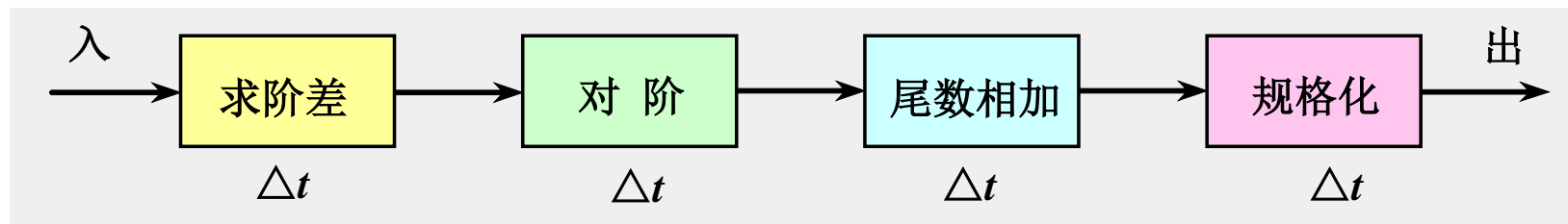
3.1.2 流水线的分类

从不同的角度和观点，把流水线分成多种不同的种类。

1. 部件级、处理机级及处理机间流水线

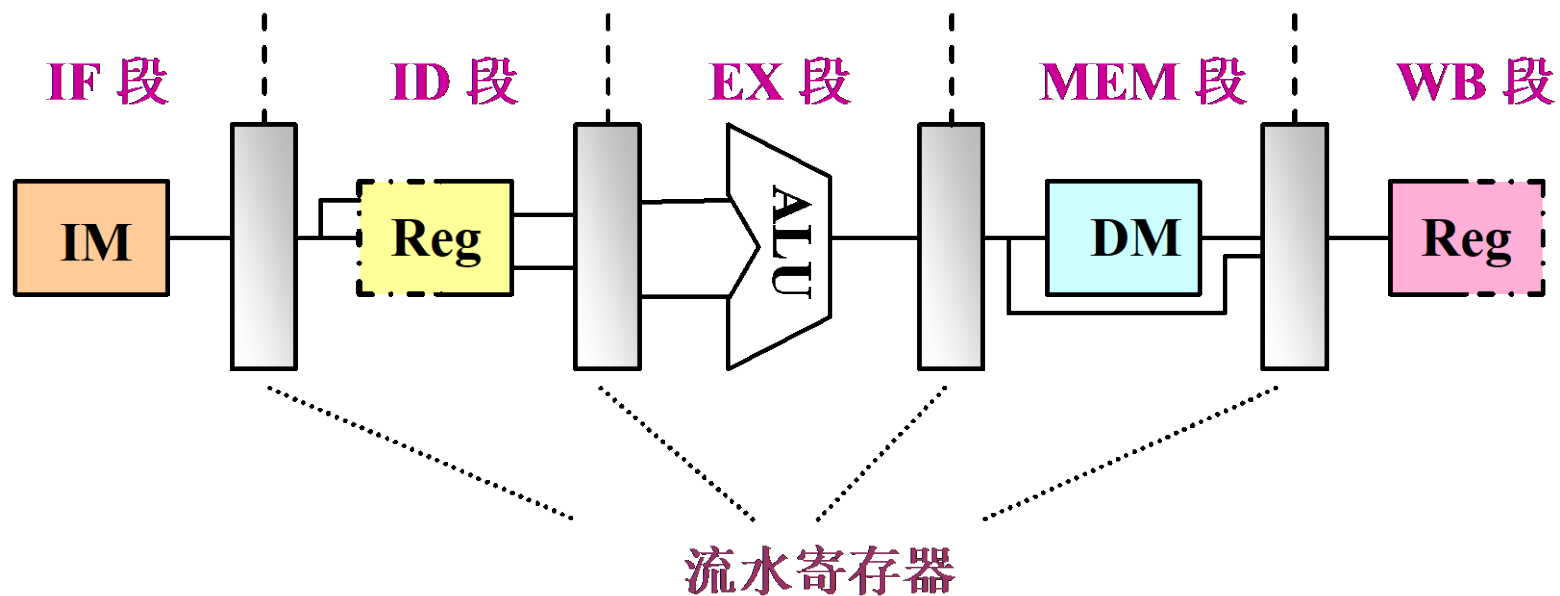
（按照流水技术用于计算机系统的等级不同）

- **部件级流水线**（运算操作流水线）：把处理机中的部件分段，再把这些分段相互连接起来，使得各种类型的运算操作能够按流水方式进行



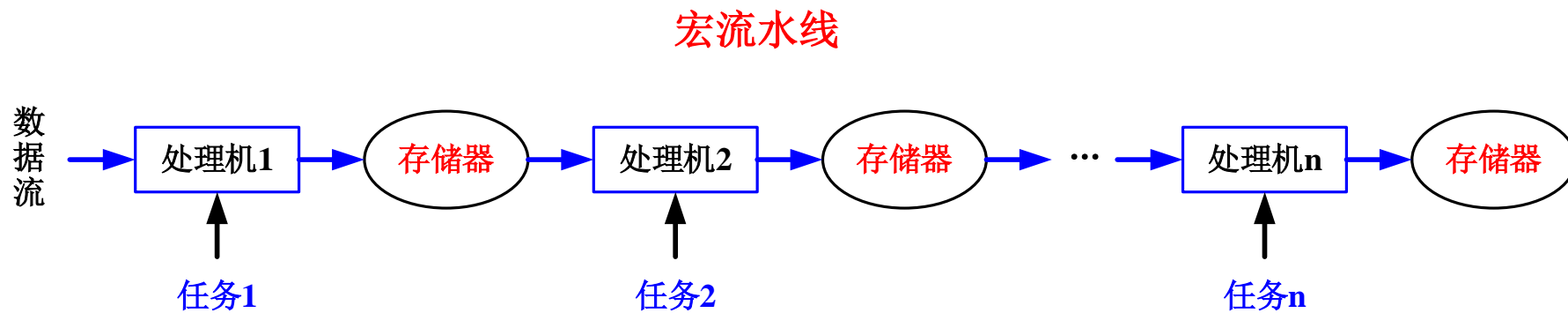
3.1 流水线的基本概念

- **处理机级流水线（指令流水线）**：把指令的执行过程按照流水方式处理。把一条指令的执行过程分解为若干个子过程，每个子过程在独立的功能部件中执行



3.1 流水线的基本概念

- **系统级流水线（宏流水线）**：把多台处理机串行连接起来，对同一数据流进行处理，每个处理机完成整个任务中的一部分。



3.1 流水线的基本概念

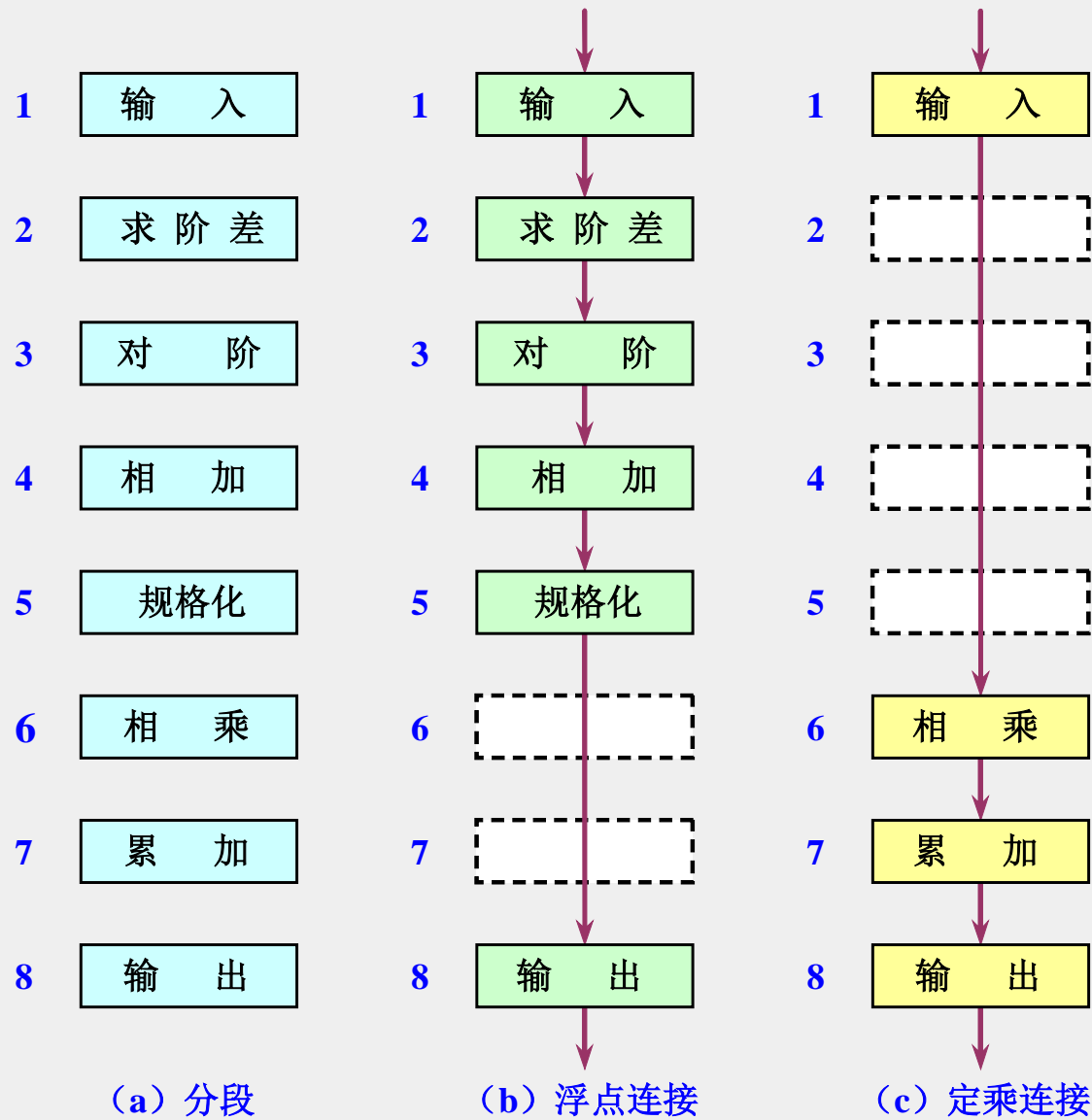
3.1.2 流水线的分类

2. 单功能流水线与多功能流水线

（按照流水线所完成的功能来分类）

- **单功能流水线：**只能完成一种固定功能的流水线
- **多功能流水线：**流水线的各段可以进行不同的连接，
以实现不同的功能

例： [ASC的多功能流水线](#)（图3.3）



3.1 流水线的基本概念

3.1.2 流水线的分类

3. 静态流水线与动态流水线

（按照同一时间内各段之间的连接方式对多功能流水线作进一步的分类）

➤ **静态流水线：**在同一时间内，多功能流水线中的各段只能按同一种功能的连接方式工作。

□ 对于静态流水线来说，只有当输入的是一串相同的运算任务时，流水的效率才能得到充分的发挥。

例如：ASC的8段流水线（固定单一任务，大公司）

3.1 流水线的基本概念

- **动态流水线：**在同一时间内，多功能流水线中的各段可以按照不同的方式连接，同时执行多种功能。

- **优点**

- 灵活，能够提高流水线各段的使用率，从而提高处理速度

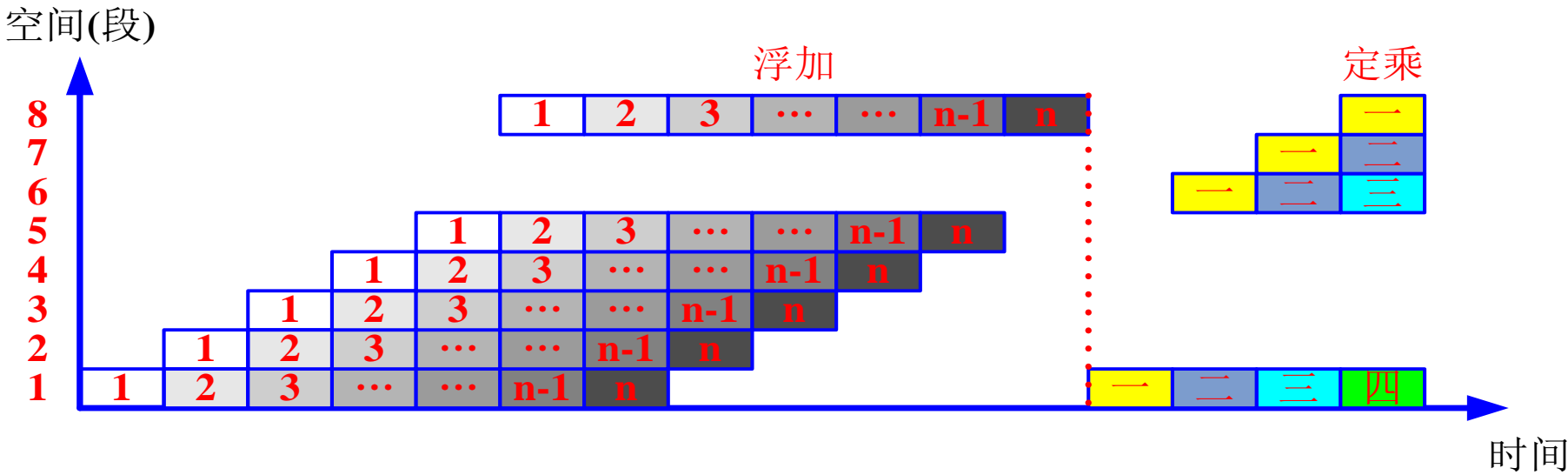
- **缺点**

- 控制复杂

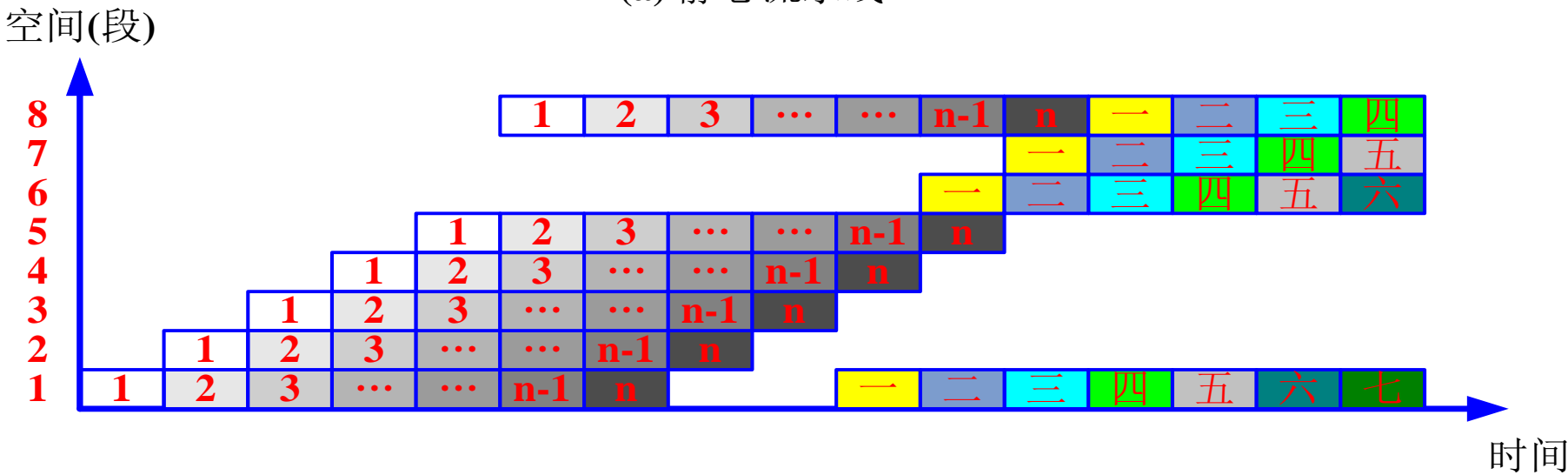
- 静、动态流水线时空图的对比

静、动态流水线的时空图

假设该流水线要先做几个浮点加法，然后再做一批定点乘法。



(a) 静态流水线



(b) 动态流水线

3.1 流水线的基本概念

3.1.2 流水线的分类

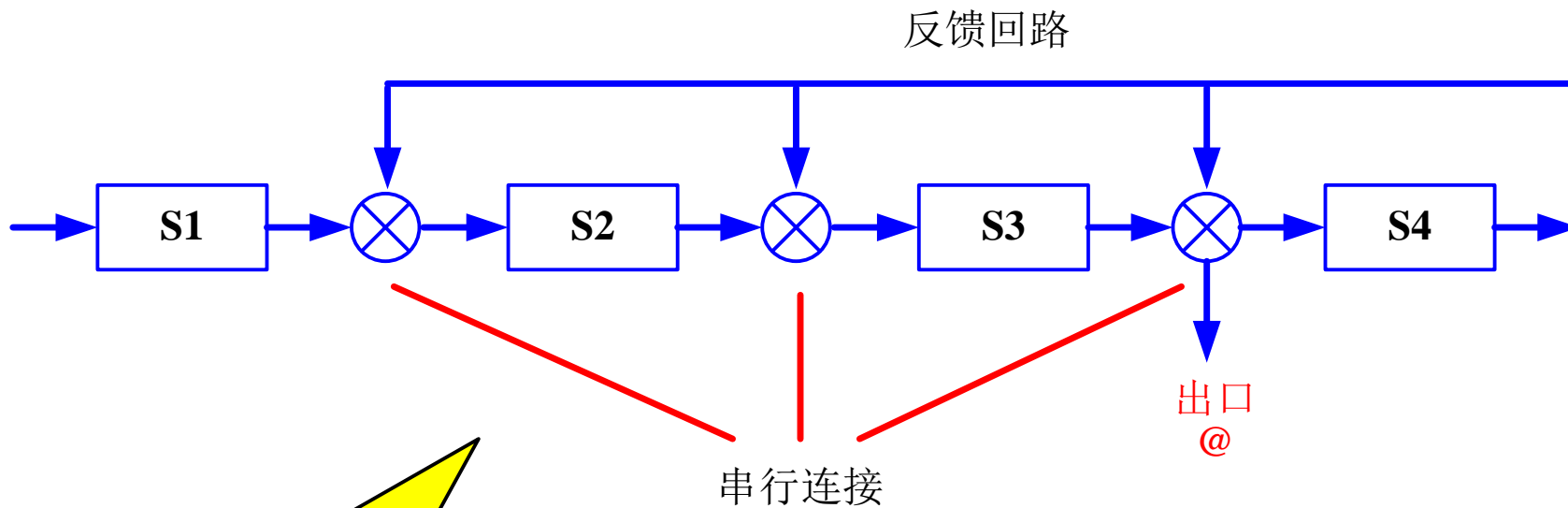
4. 线性流水线与非线性流水线

(按照流水线中是否有反馈回路来进行分类)

- **线性流水线**：流水线的各段串行连接，没有反馈回路。数据通过流水线中的各段时，每一个段最多只流过一次。
- **非线性流水线**：流水线中除了有串行的连接外，还有反馈回路。
- 非线性流水线的调度问题
 - 确定什么时候向流水线引进新的任务，才能使该任务不会与先前进入流水线的任务发生冲突——争用流水段

3.1 流水线的基本概念

非线性流水线（举例）



虽然流水线仅由四段构成，
但有些段可能要重复通过。

例如任务 @：

→S1→S2→S3→S4→S2→S3→S4→S3→

3.1 流水线的基本概念

3.1.2 流水线的分类

5. 顺序流水线与乱序流水线

（根据任务流入和流出的顺序是否相同来进行分类）

➤ **顺序流水线：**

流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。

每一个任务在流水线的各段中是一个跟着一个顺序流动的。

➤ **乱序流水线：**

流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，

允许后进入流水线的任务先完成（从输出端流出）。

也称为无序流水线、错序流水线、异步流水线

3.2 流水线的性能指标

3.2.1 吞吐量

吞吐量：在单位时间内流水线所完成的任务数量或输出结果的数量

$$TP = \frac{n}{T_k}$$

n ：任务数

T_k ：处理完成 n 个任务所用的时间

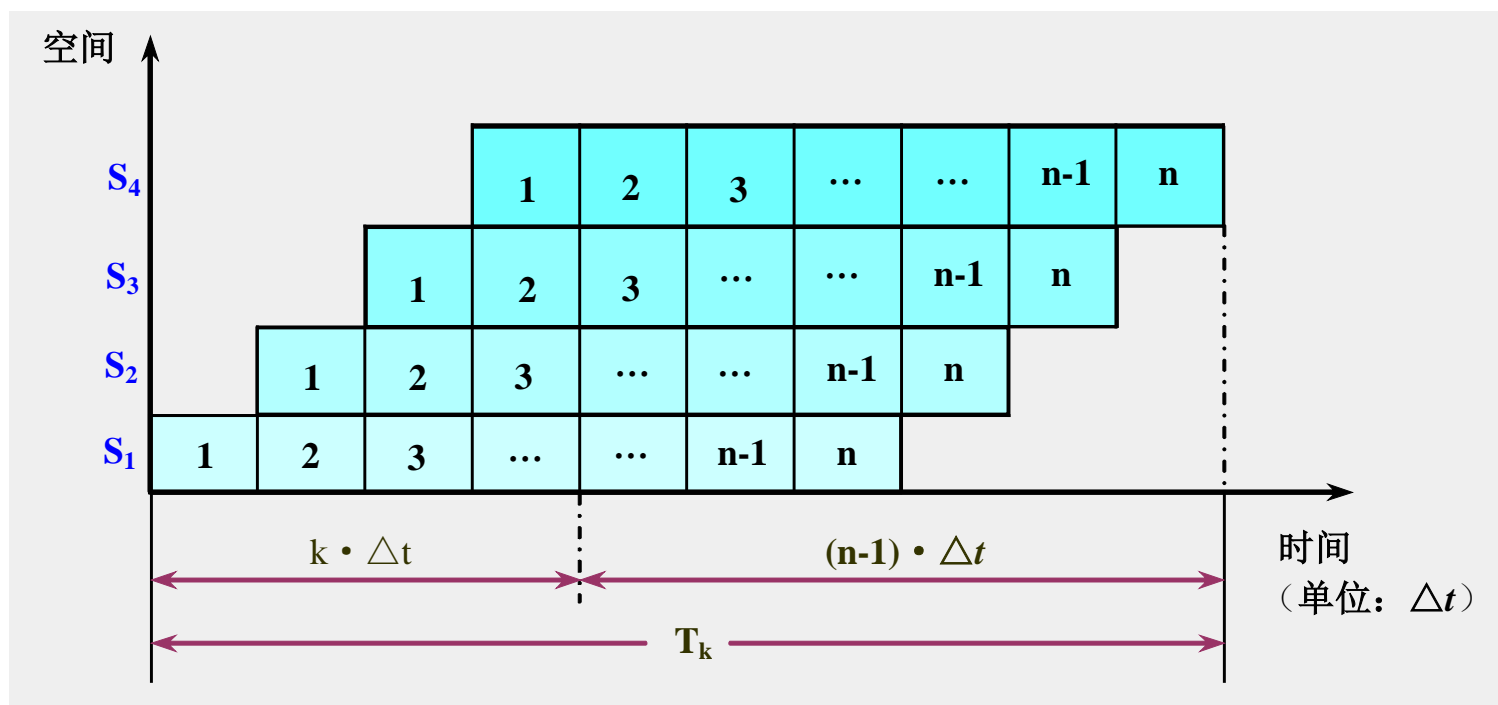


OPS(Operates Per Second), **IOPS**(IOs Per Second),
TPS(Transactions Per Second), **QPS**(Queries Per Second)

3.2 流水线的性能指标

1. 各段时间均相等的流水线

➤ 各段时间均相等的流水线时空图



3.2 流水线的性能指标

- 流水线完成 n 个连续任务所需要的总时间为：

（假设一条 k 段线性流水线）

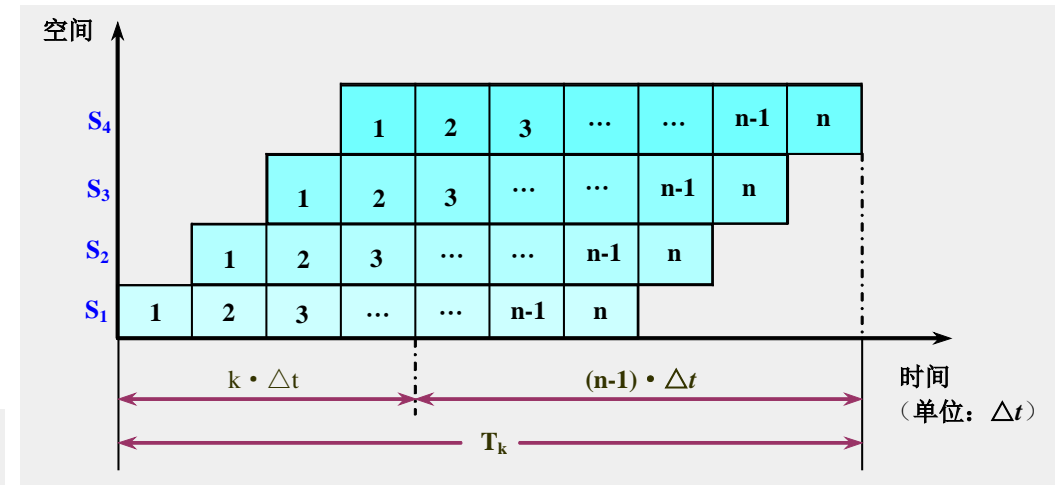
$$T_k = k \Delta t + (n-1) \Delta t = (n+k-1) \Delta t$$

- 流水线的实际吞吐率

$$TP = \frac{n}{(n+k-1)\Delta t}$$

- 最大吞吐率

$$TP_{\max} = \lim_{n \rightarrow \infty} \frac{n}{(n+k-1)\Delta t} = \frac{1}{\Delta t}$$



3.2 流水线的性能指标

➤ 最大吞吐率与实际吞吐率的关系

$$TP = \frac{n}{n + k - 1} TP_{\max}$$

- 流水线的实际吞吐率小于最大吞吐率，它除了与每个段的时间有关外，还与流水线的段数 k 以及输入到流水线中的任务数 n 等有关。
- 只有当 $n \gg k$ 时，才有 $TP \approx TP_{\max}$

3.2 流水线的性能指标

2. 各段时间不完全相等的流水线

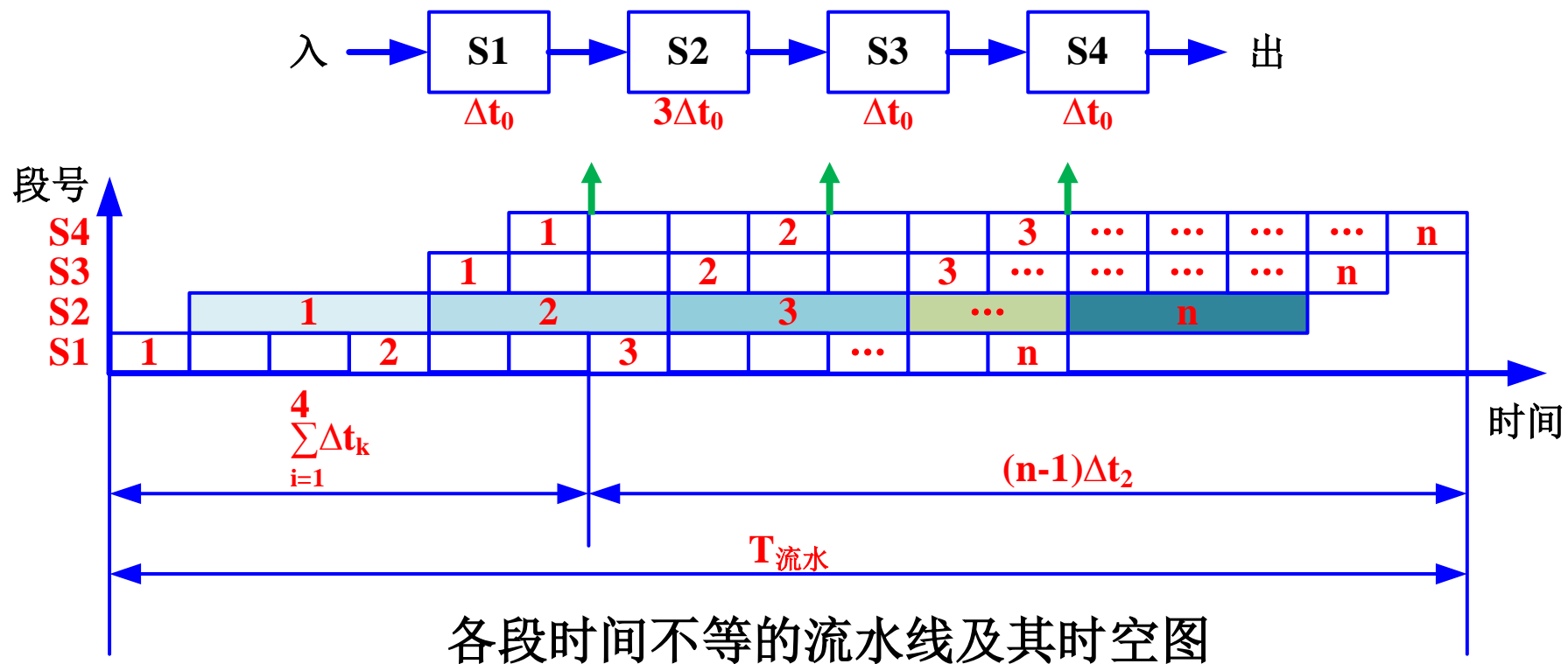
➤ 各段时间不等的流水线及其时空图

举例1 (时空图)

- 一条4段的流水线
- S1, S3, S4各段的时间: Δt
- S2的时间: $3\Delta t$ (瓶颈段)

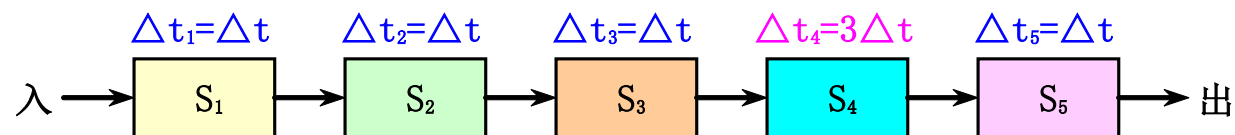
流水线中这种时间最长的段称为流水线的**瓶颈段**

3.2 流水线的性能指标

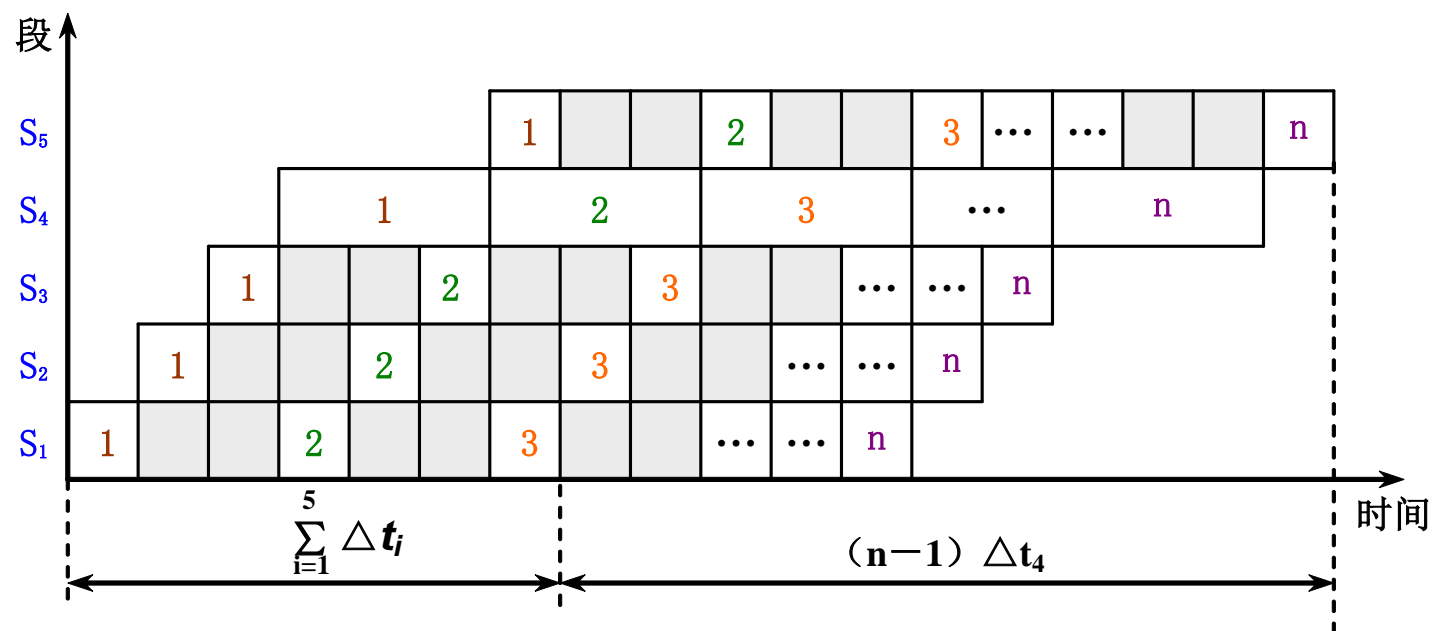


举例2：一条5段的流水线

- S_1, S_2, S_3, S_5 各段的时间: Δt
- S_4 的时间: $3\Delta t$ (瓶颈段)



(a) 流水线



(b) 时空图

3.2 流水线的性能指标

- 各段时间不等的流水线的实际吞吐率为：

（ Δt_i 为第 i 段的时间，共有 k 个段 ）

$$TP = \frac{n}{\sum_{i=1}^k \Delta t_i + (n-1) \max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$

- 流水线的最大吞吐率为：

$$TP_{\max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$

3.2 流水线的性能指标

对前面举例2中的5段流水线

最大吞吐率为：

$$TP_{\max} = \frac{1}{3\Delta t}$$

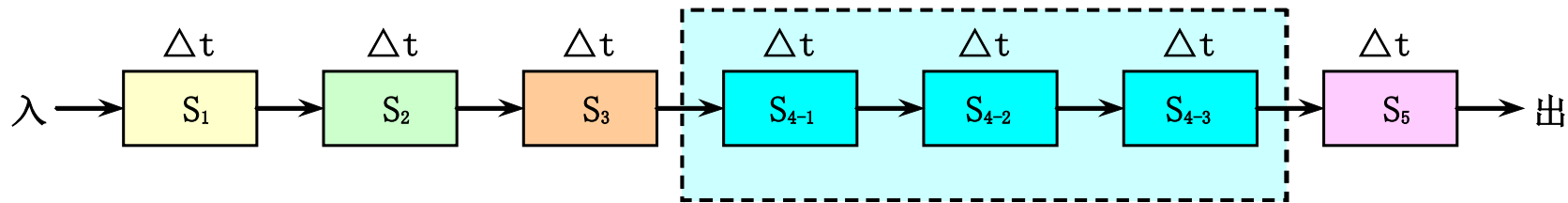
3.2 流水线的性能指标

3. 解决流水线瓶颈问题的常用方法

➤ 细分瓶颈段

例如：对前面的5段流水线

把瓶颈段 S_4 细分为3个子流水线段： S_{4-1} ， S_{4-2} ， S_{4-3}



改进后的流水线的吞吐率：

$$TP_{\max} = \frac{1}{\Delta t}$$

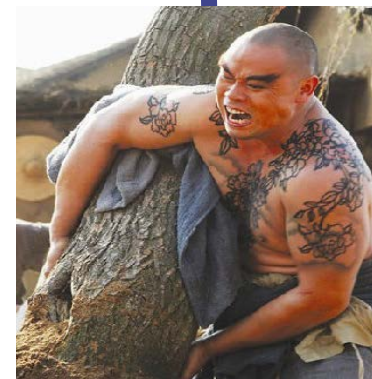
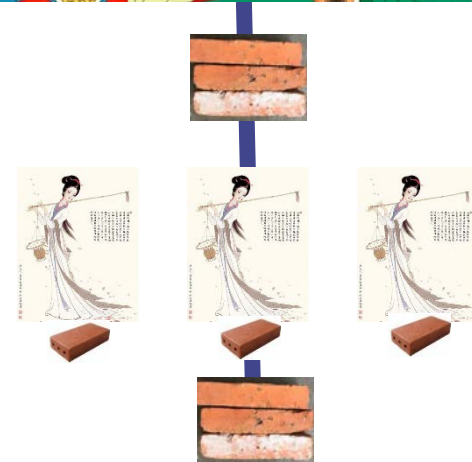
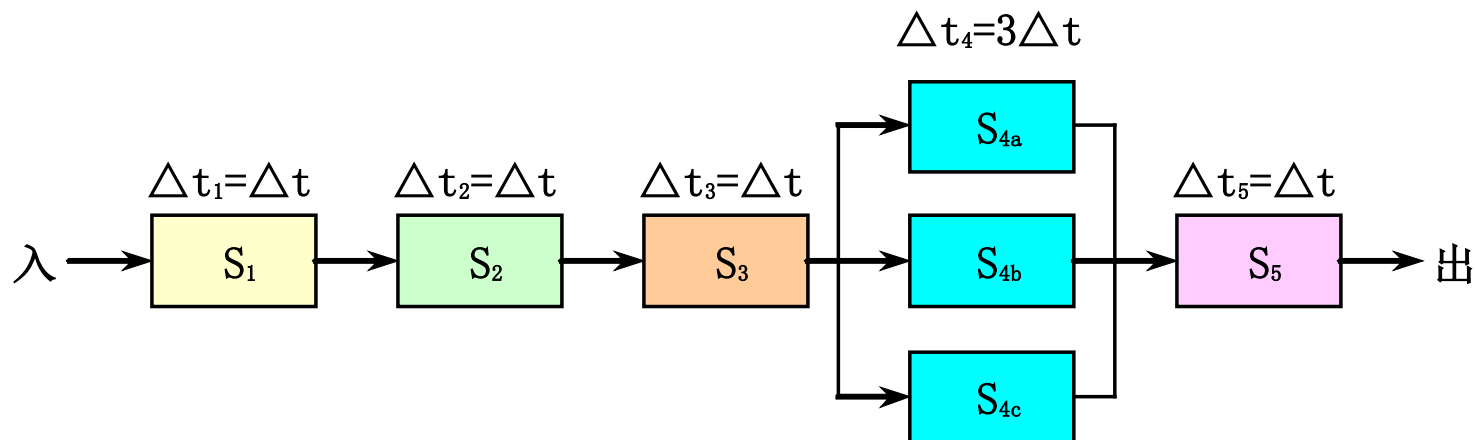
3.2 流水线的性能指标

➤ 重复设置瓶颈段

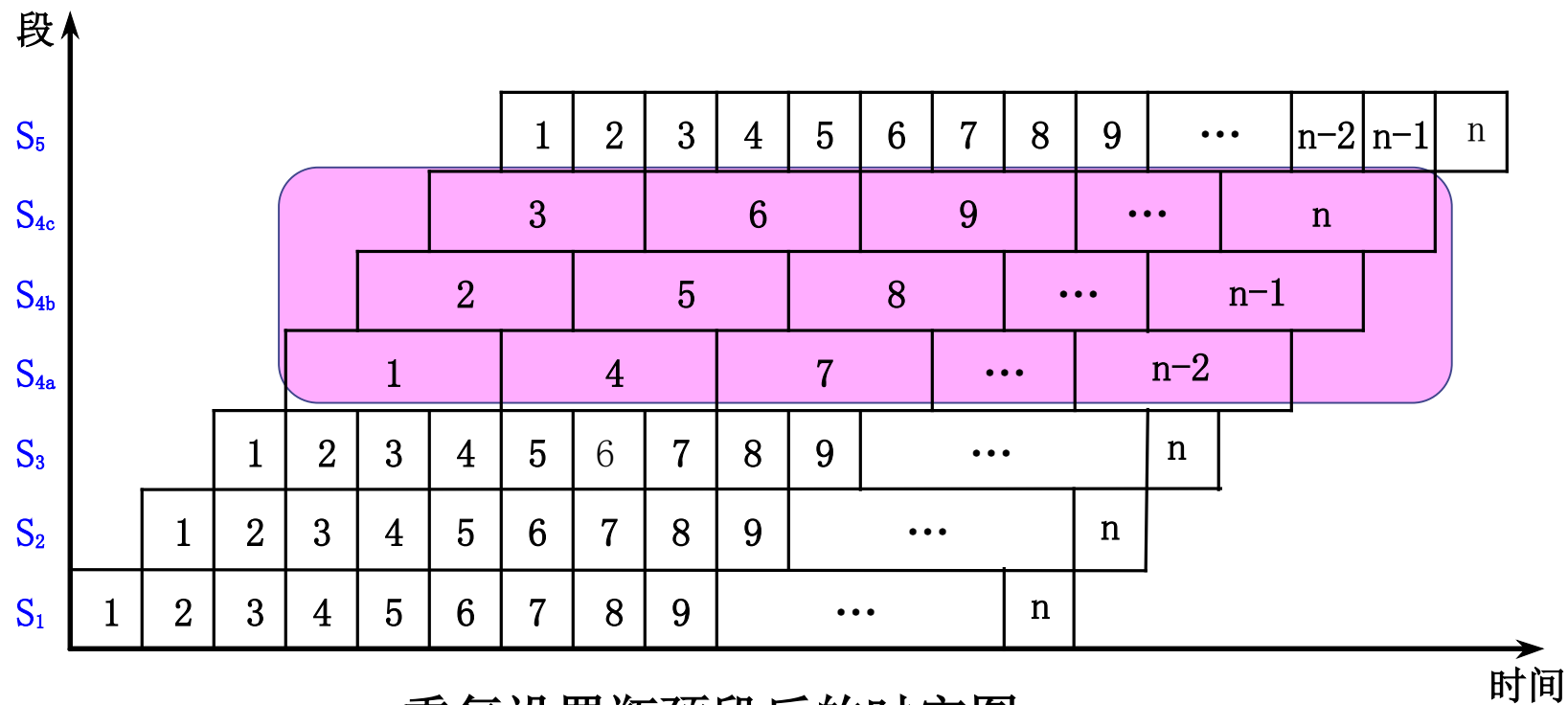
- 举例：时-空图
- 缺点：控制逻辑比较复杂，所需的硬件增加了。

例如：对前面的5段流水线

重复设置瓶颈段 S_4 ： S_{4a} , S_{4b} , S_{4c}



3.2 流水线的性能指标



重复设置瓶颈段后的时空图

改进后的流水线的吞吐率：

$$TP_{\max} = \frac{1}{\Delta t}$$

3.2 流水线的性能指标

3.2.2 流水线的加速比

加速比：完成同样一批任务，不使用流水线所用的时间与使用流水线所用的时间之比。

假设：不使用流水线（即顺序执行）所用的时间为 T_s ，使用流水线后所用的时间为 T_k ，则该流水线的加速比为：

$$S = \frac{T_s}{T_k}$$

3.2 流水线的性能指标

1. 流水线各段时间相等（都是 Δt ）

- 一条 k 段流水线完成 n 个连续任务

所需要的时间为：

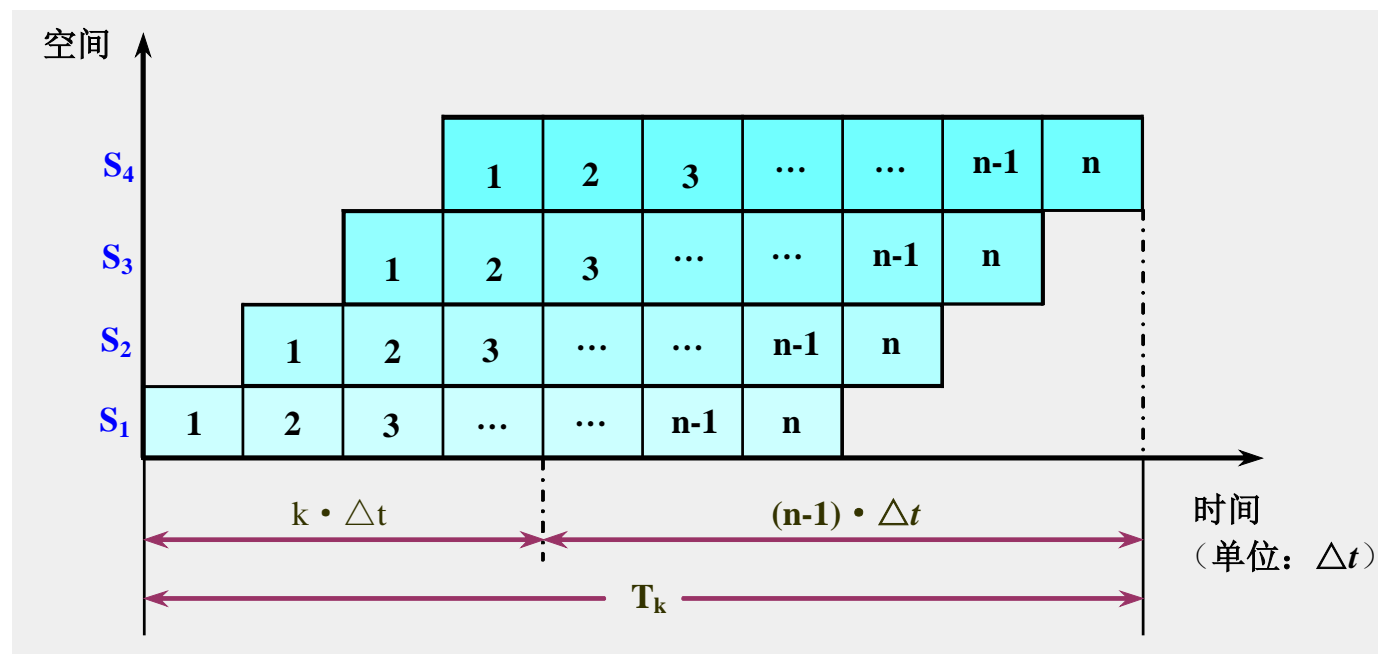
$$T_k = (k + n - 1)\Delta t$$

- 顺序执行 n 个任务

所需要的时间： $T_s = nk\Delta t$

- 流水线的实际加速比为：

$$S = \frac{nk}{k + n - 1}$$



3.2 流水线的性能指标

➤ 最大加速比

$$S_{\max} = \lim_{n \rightarrow \infty} \frac{nk}{k + n - 1} = k$$

当 $n \gg k$ 时, $S \approx k$

思考：流水线的段数愈多愈好？

3.2 流水线的性能指标

2. 流水线的各段时间不完全相等时

- 一条 k 段流水线完成 n 个连续任务的实际加速比为：

$$S = \frac{n \sum_{i=1}^k \Delta t_i}{\sum_{i=1}^k \Delta t_i + (n-1) \max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$

3.2 流水线的性能指标

3.2.3 流水线的效率

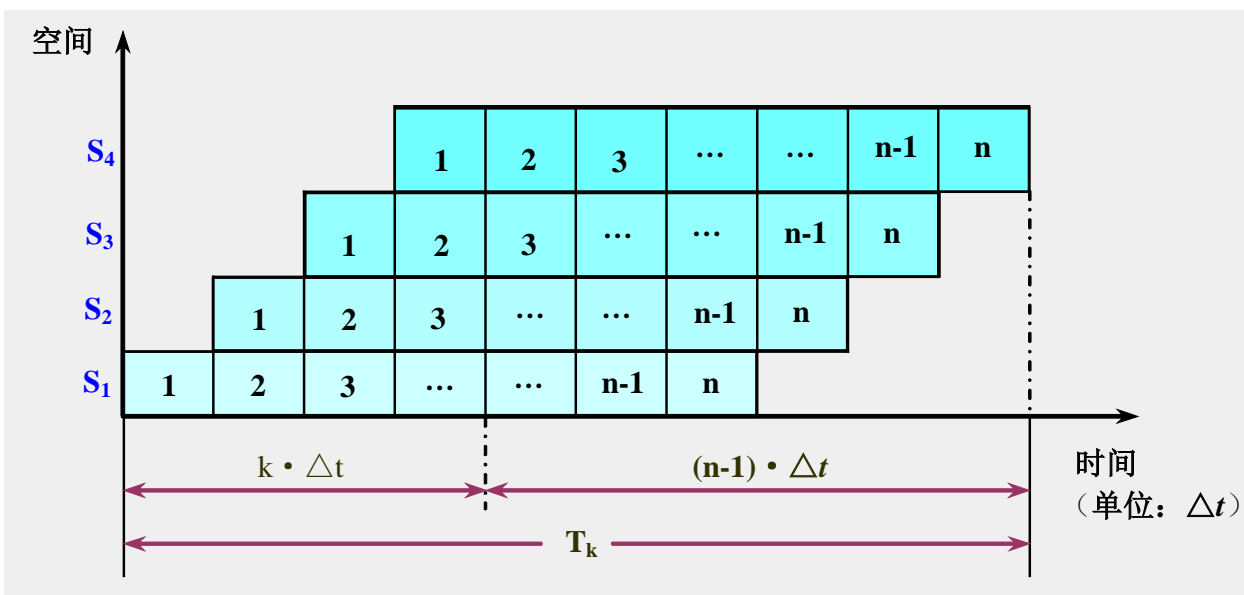
流水线的效率：流水线中的设备实际使用时间与整个运行时间的比值，即流水线设备的利用率。

由于流水线有通过时间和排空时间，所以在连续完成 n 个任务的时间内，各段并不是满负荷地工作。

1. 各段时间相等

➤ 各段的效率 e_i 相同

$$e_1 = e_2 = \cdots = e_k = \frac{n\Delta t}{T_k} = \frac{n}{k + n - 1}$$



- 整条流水线的效率为：

$$E = \frac{e_1 + e_2 + \cdots + e_k}{k} = \frac{ke_1}{k} = \frac{kn\Delta t}{kT_k}$$

- 可以写成：

$$E = \frac{n}{k + n - 1}$$

- 最高效率为：

$$E_{\max} = \lim_{n \rightarrow \infty} \frac{n}{k + n - 1} = 1$$

当 $n \gg k$ 时， $E \approx 1$ 。

3.2 流水线的性能指标

- 当流水线各段时间相等时，流水线的效率与吞吐率成正比。

$$E = TP \cdot \Delta t$$

$$TP = \frac{n}{(k + n - 1)\Delta t}$$

- 流水线的效率是流水线的实际加速比 S 与它的最大加速比 k 的比值。

$$E = \frac{S}{k}$$

$$S = \frac{nk}{k + n - 1}$$

当 $E=1$ 时， $S=k$ ，实际加速比达到最大！

$$E = \frac{n}{k + n - 1}$$

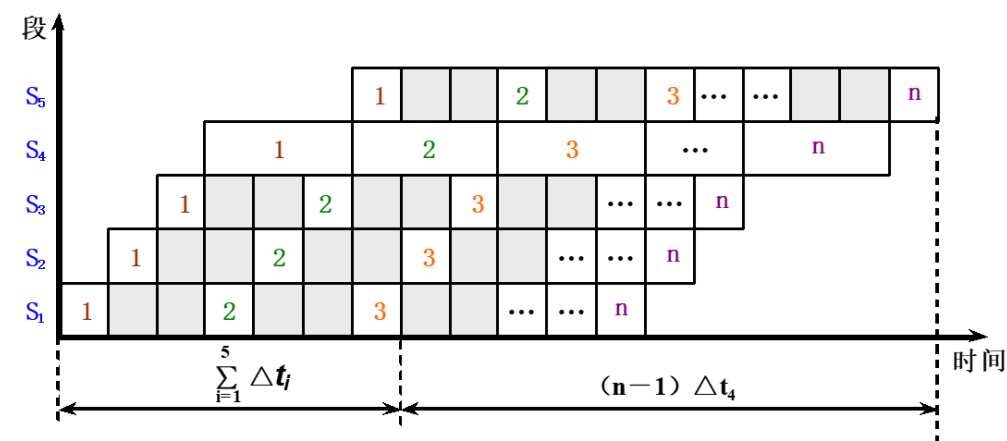
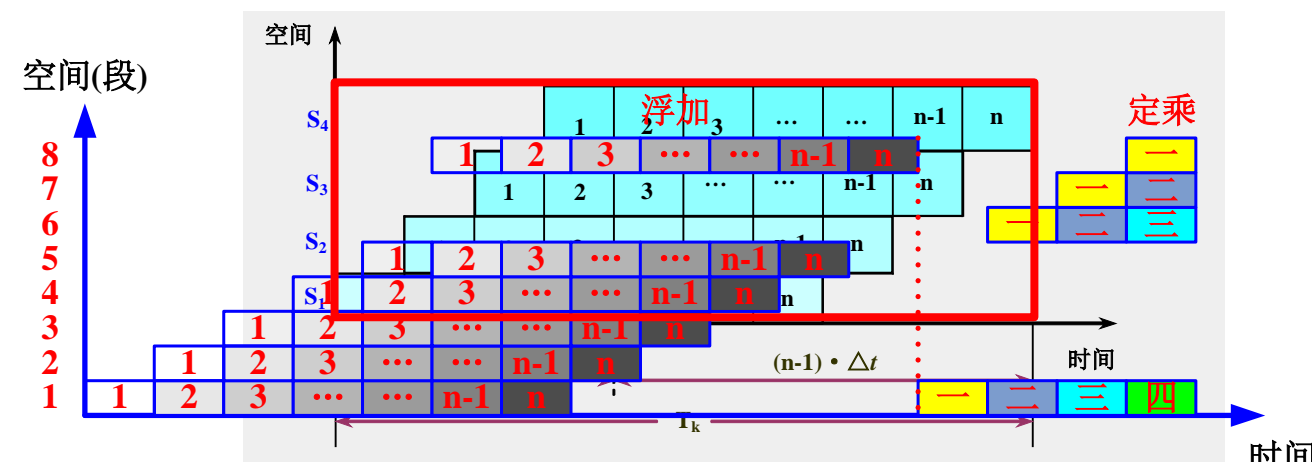
3. 从时空图上看，效率就是n个任务占用的时空面积和

k个段总的时空面积之比（举例）

$$E = \frac{n \text{ 个任务实际占用的时空区}}{k \text{ 个段总的时空区}}$$

当各段时间不相等时：

$$E = \frac{n \cdot \sum_{i=1}^k \Delta t_i}{k \left[\sum_{i=1}^k \Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \dots, \Delta t_k) \right]}$$



例3.3

已知：一条流水线， $CYCLE_{\text{顺序}}=10\text{ns}$ ， $CYCLE_{\text{流水}}=11\text{ns}$ ，各指令的时钟周期数见右表。求最大加速比。

指令类型	在程序中所占比例	时钟周期数
ALU 指令	40%	4
分支指令	20%	4
访存指令	40%	5

解：

题目未给出指令总数 n ，表示 $n=\infty$ ，这时 TP 达到 TP_{max}

定义每条指令延迟时间 $TPI=CPI \times CYCLE$

最大加速比公式 $S_{\text{max}}=TP_{\text{max流水}}/TP_{\text{max顺序}}=TPI_{\text{max顺序}}/TPI_{\text{max流水}}$

$TPI_{\text{顺序}}=CPI_{\text{顺序}} \times CYCLE_{\text{顺序}}=(4 \times 40\%+4 \times 20\%+5 \times 40\%) \times 10\text{ns}=44\text{ns}$

$TPI_{\text{流水}}=\text{MAX}\{\Delta t_i\}=CYCLE_{\text{流水}}=11\text{ns}$ （从公式3.7转）

代入公式得：最大加速比 $S_{\text{max}}=44\text{ns}/11\text{ns}=4$ 倍

例3.4

已知：一条RISC流水线，各段时间如右表（单位ns），求最大加速比。

	Δt_1	Δt_2	Δt_3	Δt_4	Δt_5
顺序方式下	10	8	10	10	7
流水方式下	11	9	11	11	8

解：

题目同样未给出指令总数n，同样使用最大加速比公式：

$$S_{\max} = TPI_{\max\text{顺序}} / TPI_{\max\text{流水}}$$

$$TPI_{\text{顺序}} = (10 + 8 + 10 + 10 + 7) \text{ ns} = 45 \text{ ns}$$

$$TPI_{\text{流水}} = \text{MAX} \{ \Delta t_i \} = 11 \text{ ns} \quad (\text{从公式3.7转})$$

$$\text{代入公式得：最大加速比 } S_{\max} = 45 \text{ ns} / 11 \text{ ns} \approx 4.1 \text{ 倍}$$

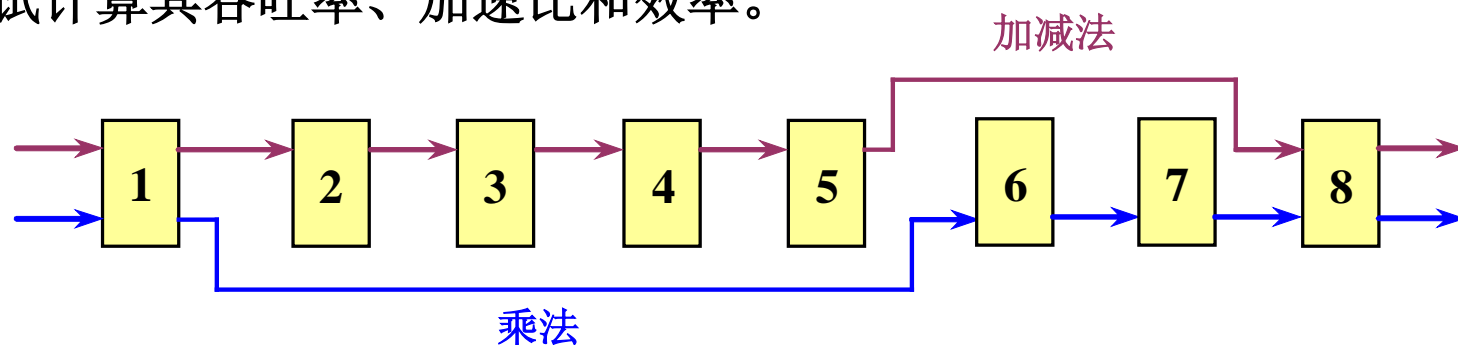
3.2 流水线的性能指标

3.2.4 流水线的性能分析举例

例3.1 设在下图所示的静态流水线上计算：

$$\prod_{i=1}^4 (A_i + B_i)$$

流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中，试计算其吞吐率、加速比和效率。



(每段的时间都为 Δt)

3.2 流水线的性能指标

解：（1）选择适合于流水线工作的算法

- 先计算 A_1+B_1 、 A_2+B_2 、 A_3+B_3 和 A_4+B_4 ；
- 再计算 $(A_1+B_1) \times (A_2+B_2)$ 和 $(A_3+B_3) \times (A_4+B_4)$ ；
- 然后求总的乘积结果。

（2）画出时空图

3.2 流水线的性能指标

(3) 计算性能

- 在18个 Δt 时间中，给出了7个结果。吞吐率为：

$$TP = \frac{7}{18\Delta t}$$

- 不用流水线，由于一次求和需 $6\Delta t$ ，一次求积需 $4\Delta t$ ，则产生上述7个结果共需 $(4 \times 6 + 3 \times 4) \Delta t = 36\Delta t$

加速比为：

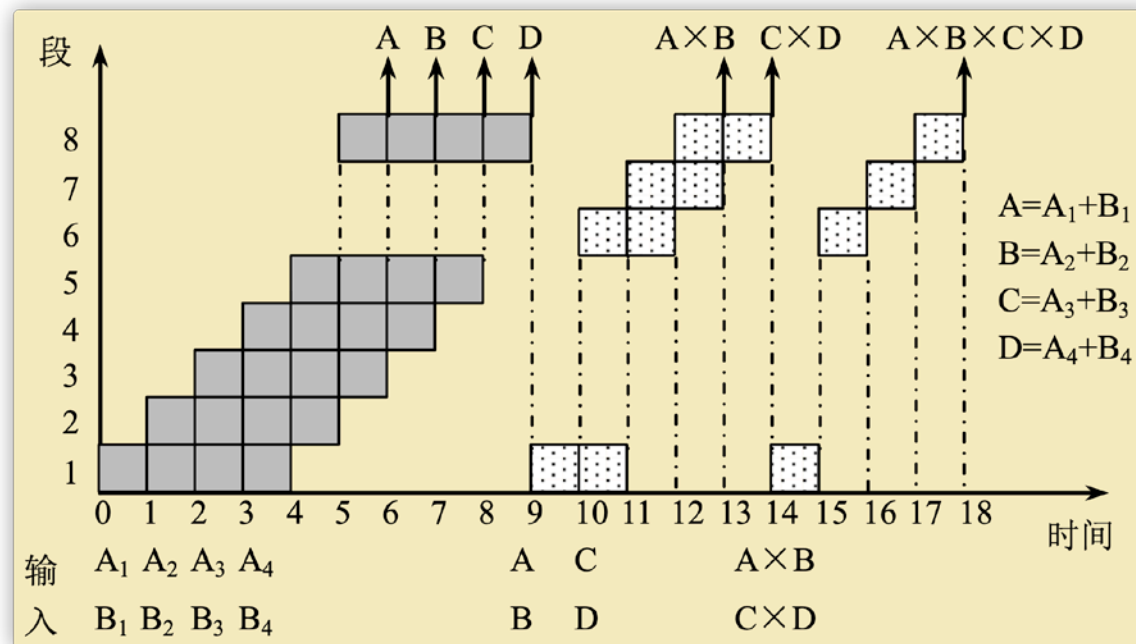
$$S = \frac{36\Delta t}{18\Delta t} = 2$$

3.2 流水线的性能指标

流水线的效率

$$E = \frac{4 \times 6 + 3 \times 4}{8 \times 18} = 0.25$$

可以看出，在求解此问题时，该流水线的效率不高。



上讲回顾

1. 流水线的概念

- 大量重复的过程/任务，分段，每段由单独的功能部件执行，形成多任务间并行
- 各段时间尽可能相等

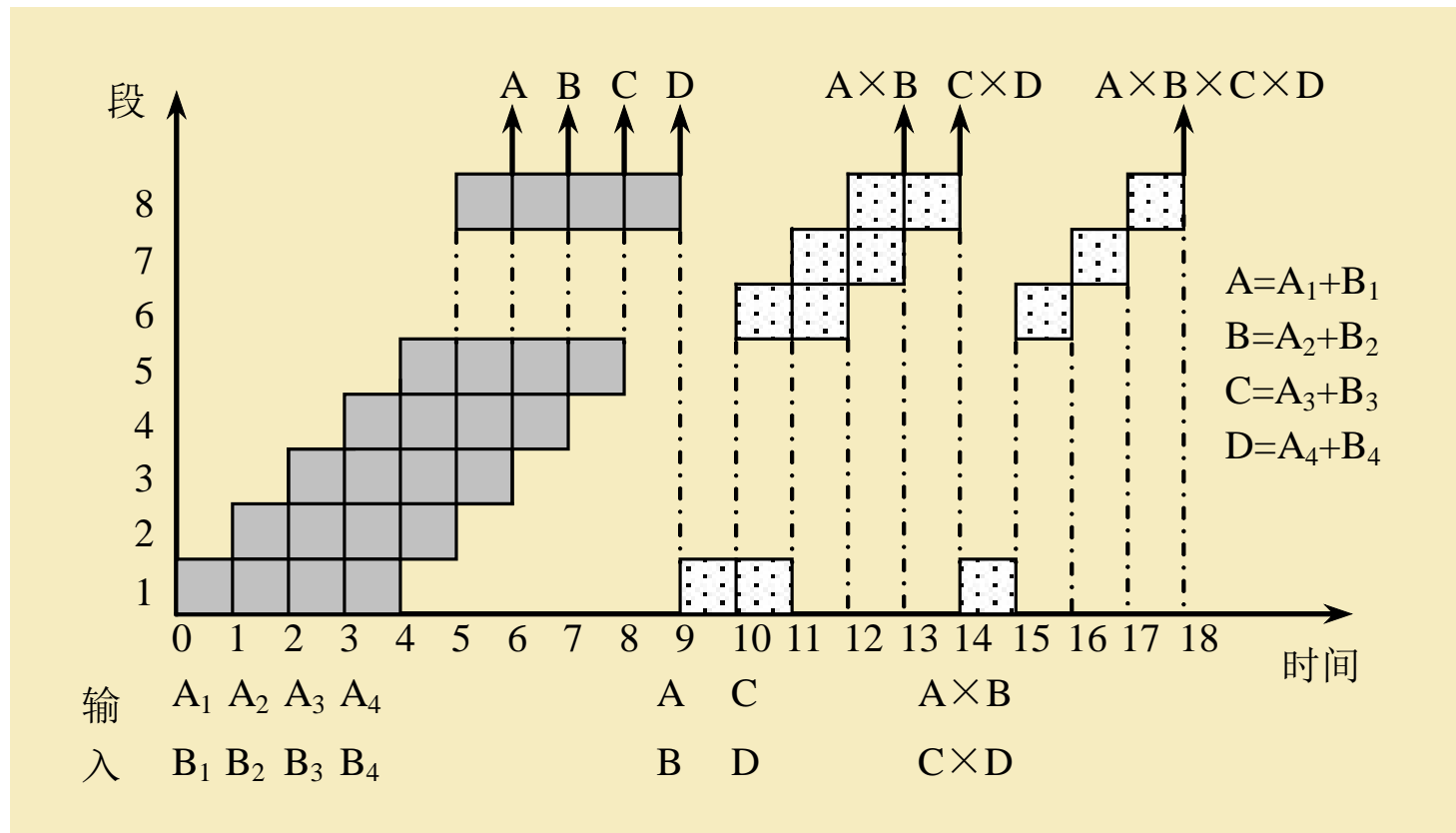
2. 流水线性能评价指标

- 吞吐率
- 加速比
- 效率

$$TP = \frac{n}{(n + k - 1)\Delta t}$$

$$S = \frac{nk}{k + n - 1}$$

$$E = \frac{n}{k + n - 1}$$



$$TP = \frac{7}{18\Delta t}$$

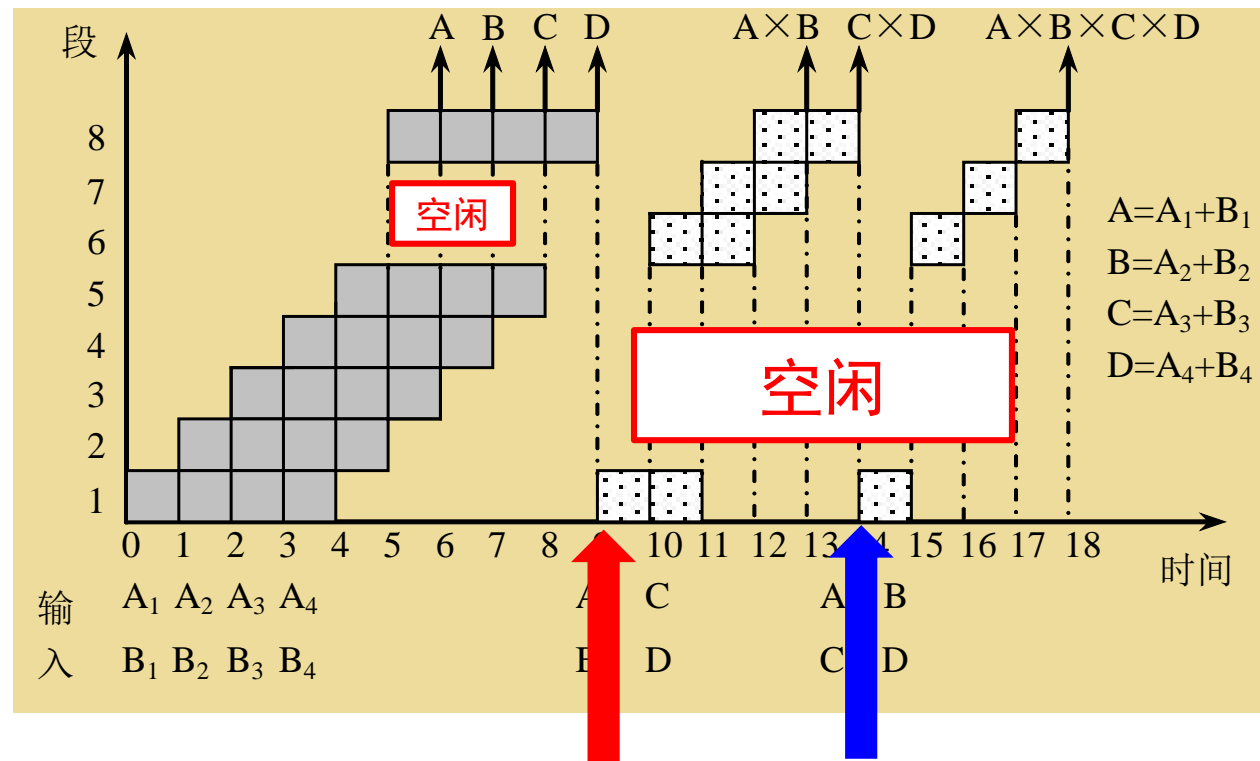
$$S = \frac{36\Delta t}{18\Delta t} = 2$$

$$E = \frac{4 \times 6 + 3 \times 4}{8 \times 18} = 0.25$$

3.2 流水线的性能指标

主要原因

- 多功能流水线在做某一种运算时，总**有一些段是空闲**的；
- 静态流水线在进行**功能切换**时，要等前一种运算全部流出流水线后才能进行后面的运算
- 运算之间存在**关联**，后面有些运算要用到前面运算的结果；
- 流水线的工作过程有**建立**与**排空**部分

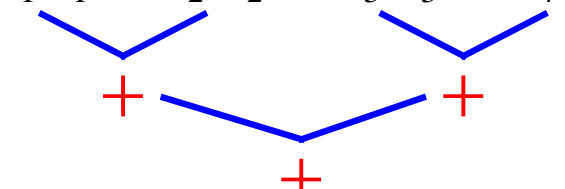


3.2 流水线的性能指标

下面我们再看一个例子：

例 在静态流水线上计算：

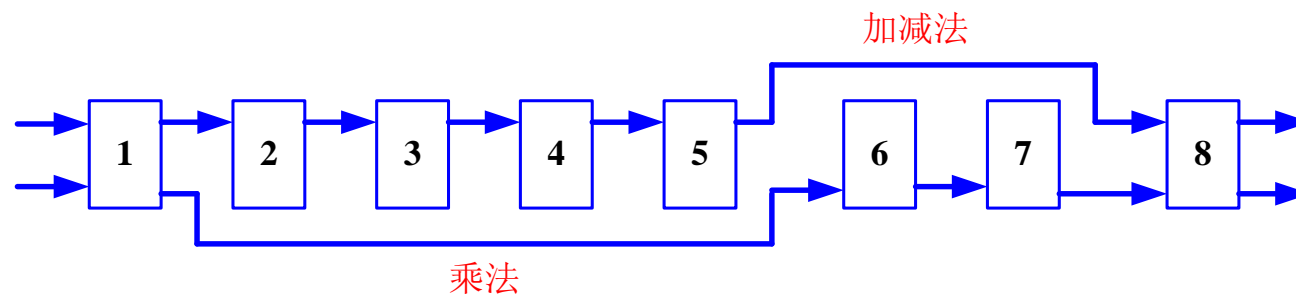
$$\sum_{i=1}^4 (A_i \times B_i)$$

$$\sum_{i=1}^4 A_i B_i = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4$$


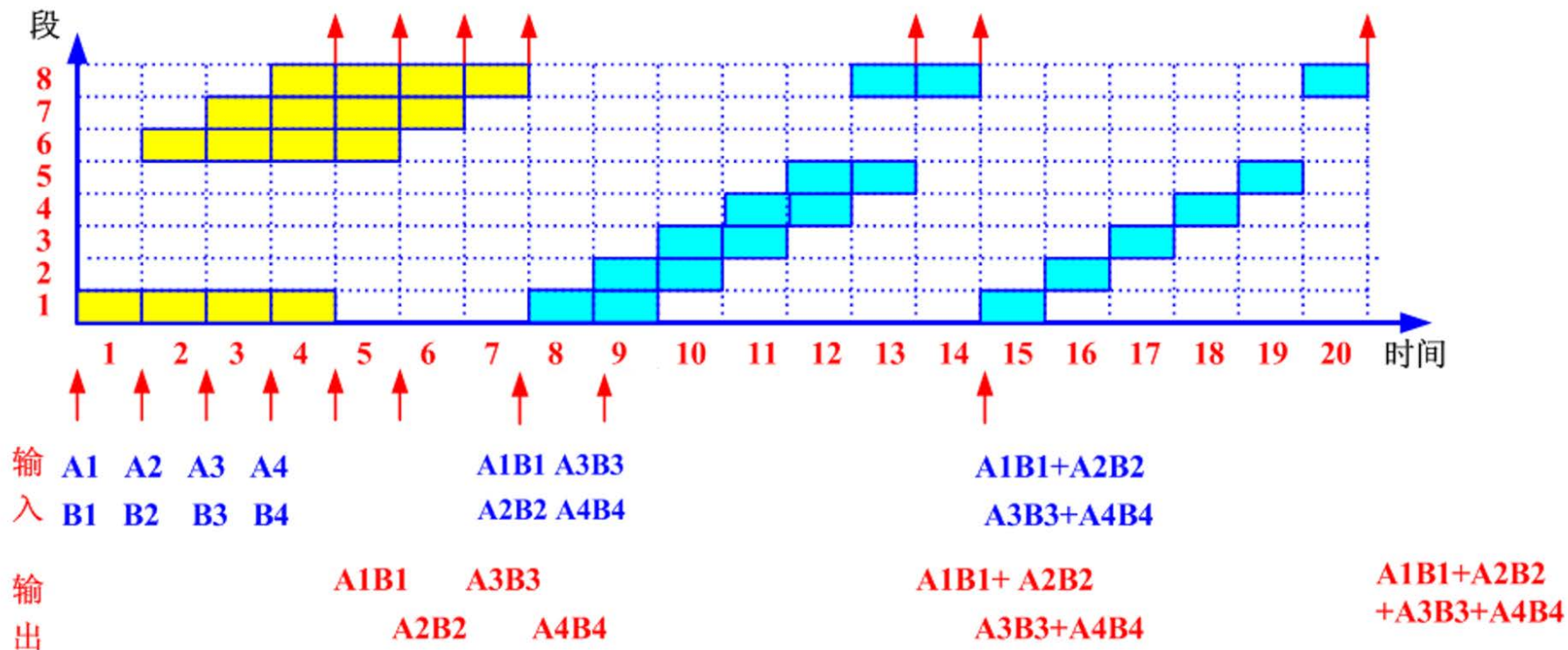
求：吞吐率，加速比，效率。

解： (1) 确定适合于流水处理的计算过程

(2) 画时空图



静态流水线时空图



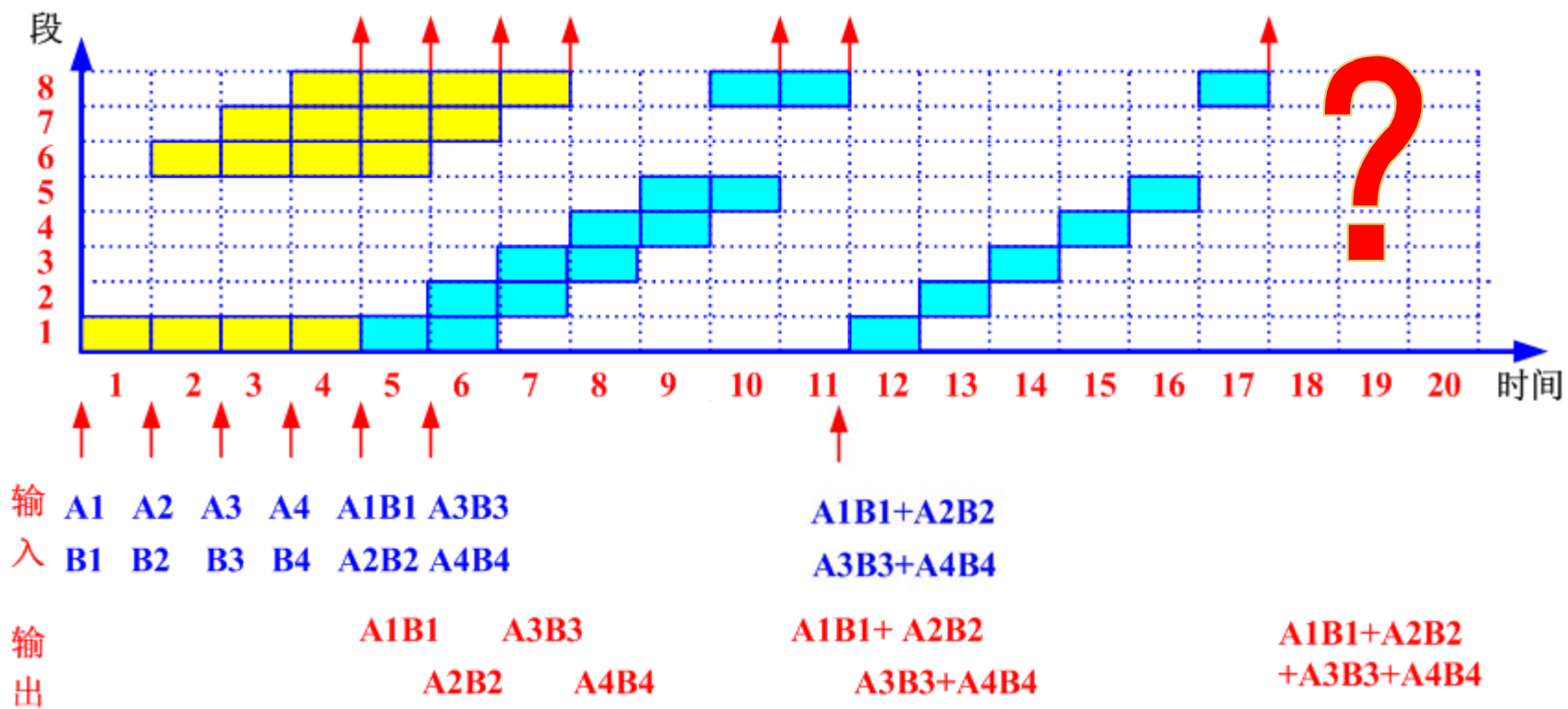
(3) 计算性能

吞吐率 $TP = 7 / (20 \Delta t)$

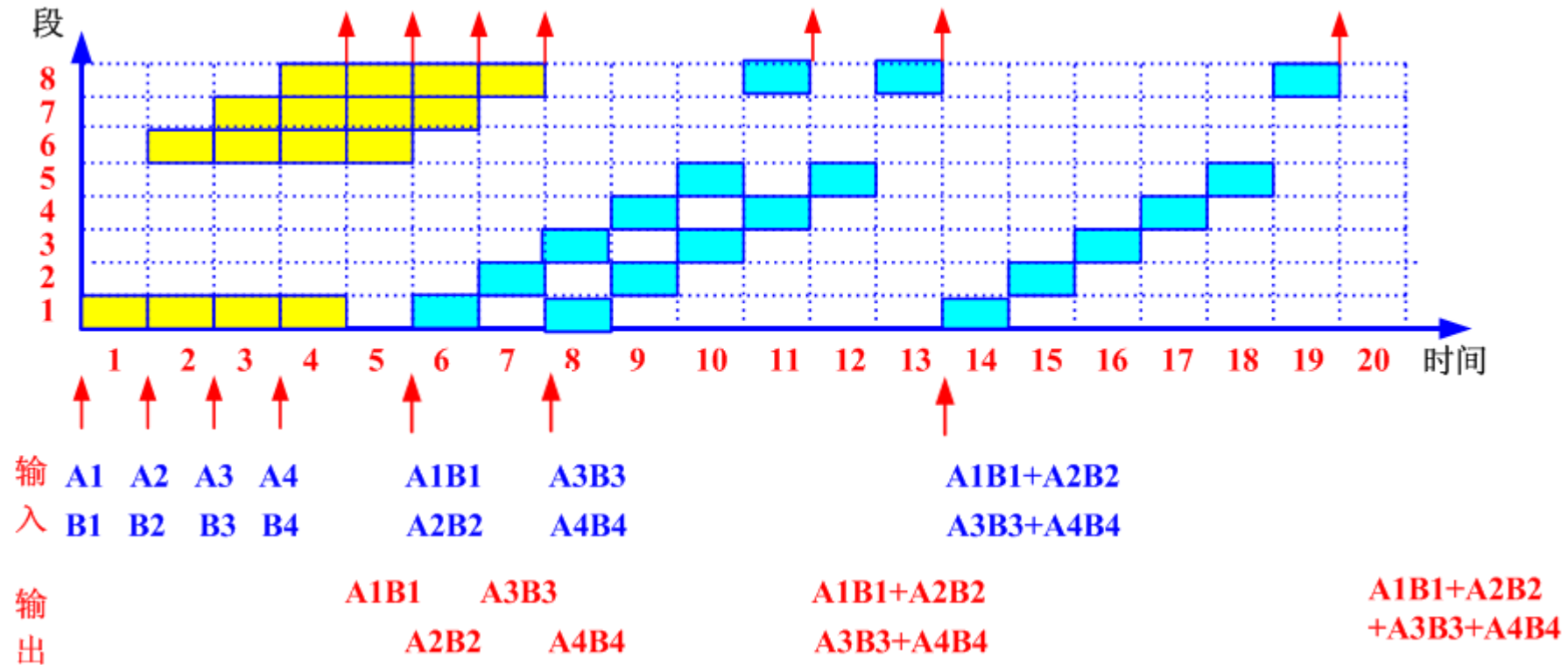
加速比 $S = (34 \Delta t) / (20 \Delta t) = 1.7$

效率 $E = (4 \times 4 + 3 \times 6) / (8 \times 20) = 0.21$

动态流水线时空图-1



动态流水线时空图-2



$$T_p = 7 / (19 \Delta t)$$

$$S = 34 / 19$$

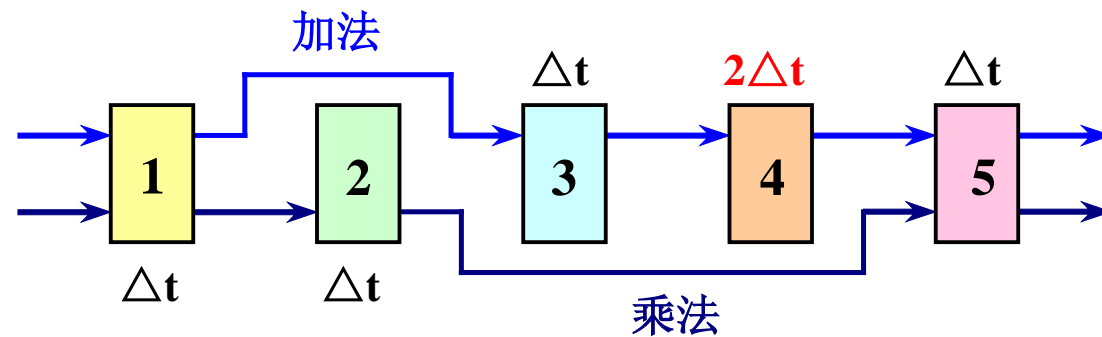
$$E = (4 * 4 + 3 * 6) / (8 * 19)$$

3.2 流水线的性能指标

例： 有一条**动态**多功能流水线由5段组成，加法用1、3、4、5段，乘法用1、2、5段，第4段的时间为 $2\Delta t$ ，其余各段时间均为 Δt ，而且流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中。若在该流水线上计算：

$$\sum_{i=1}^4 (A_i \times B_i)$$

试计算其吞吐率、加速比和效率。



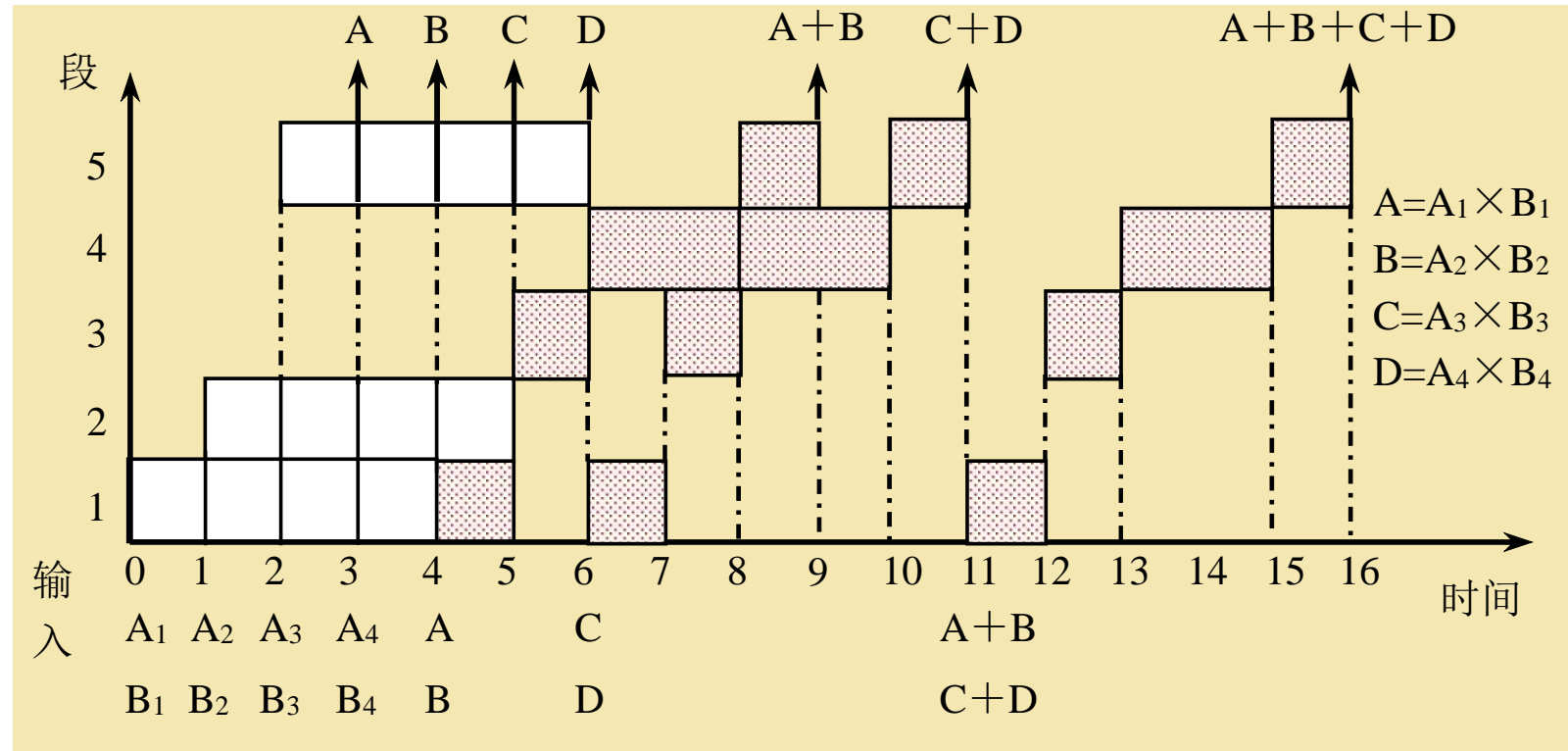
3.2 流水线的性能指标

解：(1) 选择适合于流水线工作的算法

- 应先计算 $A_1 \times B_1$ 、 $A_2 \times B_2$ 、 $A_3 \times B_3$ 和 $A_4 \times B_4$ ；
- 再计算 $(A_1 \times B_1) + (A_2 \times B_2)$
 $(A_3 \times B_3) + (A_4 \times B_4)$ ；
- 然后求总的累加结果。

(2) 画出时空图

(3) 计算性能



$$TP = \frac{7}{16\Delta t}$$

$$S = \frac{27\Delta t}{16\Delta t} \approx 1.69$$

$$E = \frac{4 \times 3 + 3 \times 5}{5 \times 16} \approx 0.338$$

3.2 流水线的性能指标

3.2.5 流水线设计中的若干问题

1. 瓶颈问题

- 理想情况下，流水线在工作时，其中的任务是同步地**每一个时钟周期往前流动一段**。
- 当流水线各段不均匀时，**机器的时钟周期取决于瓶颈段的延迟时间**。
- 在设计流水线时，要尽可能使各段时间相等。

2. 流水线的额外开销

- 流水寄存器延迟：**建立时间，传输延迟**
- 时钟偏移开销：时钟到达各流水寄存器的最大差值时间

3.2 流水线的性能指标

➤ 明确几个问题

- ❑ 流水线并不能减少（而且一般是增加）单条指令的执行时间，但却能提高吞吐率。
- ❑ 适当增加流水线的深度（段数）可以提高流水线的性能。
- ❑ 流水线的深度受限于流水线的额外开销。
- ❑ 当时钟周期小到与额外开销相同时，流水已没意义。因为这时在每一个时钟周期中已没有时间来做有用的工作。

第3章 流水线技术

1. 什么是流水线?
2. 如何评价流水线?
3. 怎样才能使流水线的效率最高?
4. 如何处理流水线的资源争用?

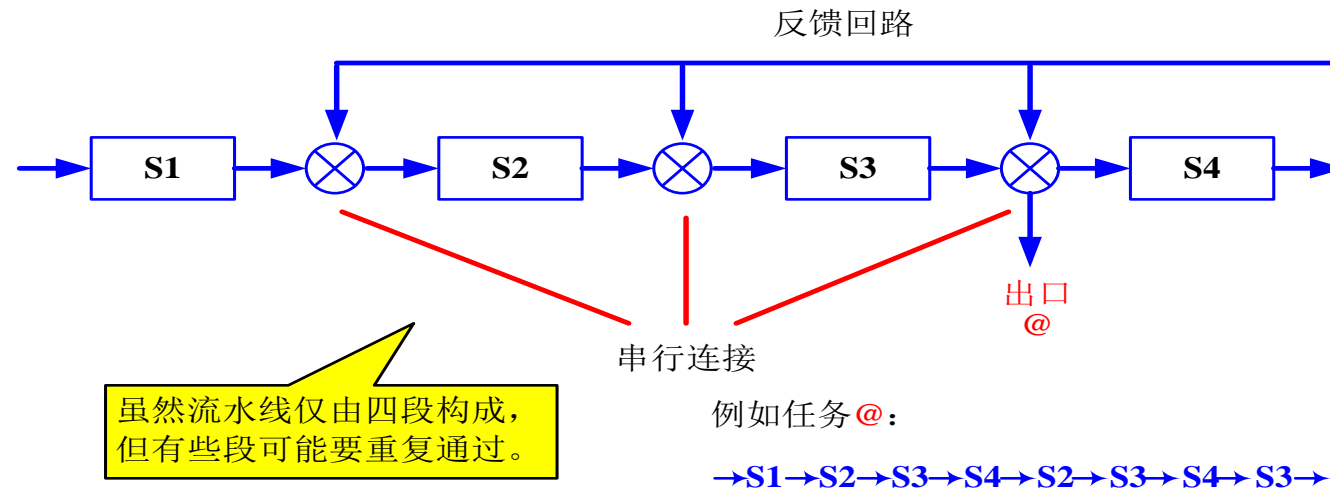


3.3 非线性流水线的调度

冲突问题

- 在非线性流水线中，存在反馈回路，当一个任务在流水线中流过时，可能要多次经过某些段。
- 流水线调度要解决的问题：

应按什么样的时间间隔向流水线输入新任务，才能**既**不发生功能段使用冲突，**又**能使流水线有较高的吞吐率和效率



3.3 非线性流水线的调度

3.3.1 单功能非线性流水线的最优调度

- 向一条非线性流水线的输入端连续输入两个任务之间的时间间隔称为非线性流水线的**启动距离**。
- 会引起非线性流水线功能段使用冲突的启动距离则称为**禁用启动距离**。
- 启动距离和禁用启动距离一般都用时钟周期数来表示，是一个正整数。
- **预约表**
 - 横向（向右）：时间（一般用时钟周期表示）
 - 纵向（向下）：流水线的段

3.3 非线性流水线的调度

预约表

- 横向：时间
- 纵向：功能段

例：一个5功能段非线性流水线预约表

时间 功能段	1	2	3	4	5	6	7	8	9
S1	√								√
S2		√	√					√	
S3				√					
S4					√	√			
S5							√	√	

- 如果在第n个时钟周期使用第k段，则在第k行和第n列的交叉处的格子里有一个√。
- 如果在第k行和第n列的交叉处的格子里有一个√，则表示在第n个时钟周期要使用第k段。

3.3 非线性流水线的调度

1. 根据预约表写出禁止表F

- **禁止表F**：一个由禁用启动距离构成的集合。
- 具体方法

对于预约表的每一行的任何一对√，用它们所在的列号相减（大的减小的），列出各种可能的差值，然后删除相同的，剩下的就是禁止表的元素。

➤ 在上例中

- ❑ 第一行的差值只有一个：8；
 - ❑ 第二行的差值有3个：1, 5, 6；
 - ❑ 第3行只有一个√，没有差值；
 - ❑ 第4和第5行的差值都只有一个：1；
- 其禁止表是：F = { 1, 5, 6, 8 }

时间 功能段	1	2	3	4	5	6	7	8	9
S1	√								√
S2		√	√					√	
S3				√					
S4					√	√			
S5							√	√	

3.3 非线性流水线的调度

2. 根据禁止表F写出初始冲突向量 C_0

(进行从一个集合到一个二进制位串的转变)

➤ **冲突向量C**：一个N位的二进制位串。

➤ 设 $C_0 = (c_N c_{N-1} \dots c_i \dots c_2 c_1)$ ，则：

$$c_i = \begin{cases} 1 & i \in F \\ 0 & i \notin F \end{cases}$$

□ $c_i=0$ ：允许间隔*i*个时钟周期后送入后续任务

□ $c_i=1$ ：不允许间隔*i*个时钟周期后送入后续任务

➤ 对于上面的例子

$$F = \{ 1, 5, 6, 8 \}$$

$$C_0 = (10110001)$$

3.3 非线性流水线的调度

3. 根据初始冲突向量 C_0 画出状态转换图

- 当第一个任务流入流水线后，初始冲突向量 C_0 决定了下一个任务需间隔多少个时钟周期才可以流入。
- 在第二个任务流入后，新的冲突向量是怎样的呢？

- 假设第二个任务是在与第一个任务间隔 j 个时钟周期流入，这时，由于第一个任务已经在流水线中前进了 j 个时钟周期，其相应的禁止表中各元素的值都应该减去 j ，并丢弃小于等于0的值。
- 对冲突向量来说，就是逻辑右移 j 位（左边补0）。
- 在冲突向量上，就是对它们的冲突向量进行“或”运算。

$$SHR^{(j)}(C_0) \vee C_0$$

其中： $SHR^{(j)}$ 表示逻辑右移 j 位

根据当前冲突向量
可以决定后续何时
可流入新的任务

3.3 非线性流水线的调度

➤ 推广到更一般的情况

假设: C_k : 当前的冲突向量

j : 允许的时间间隔

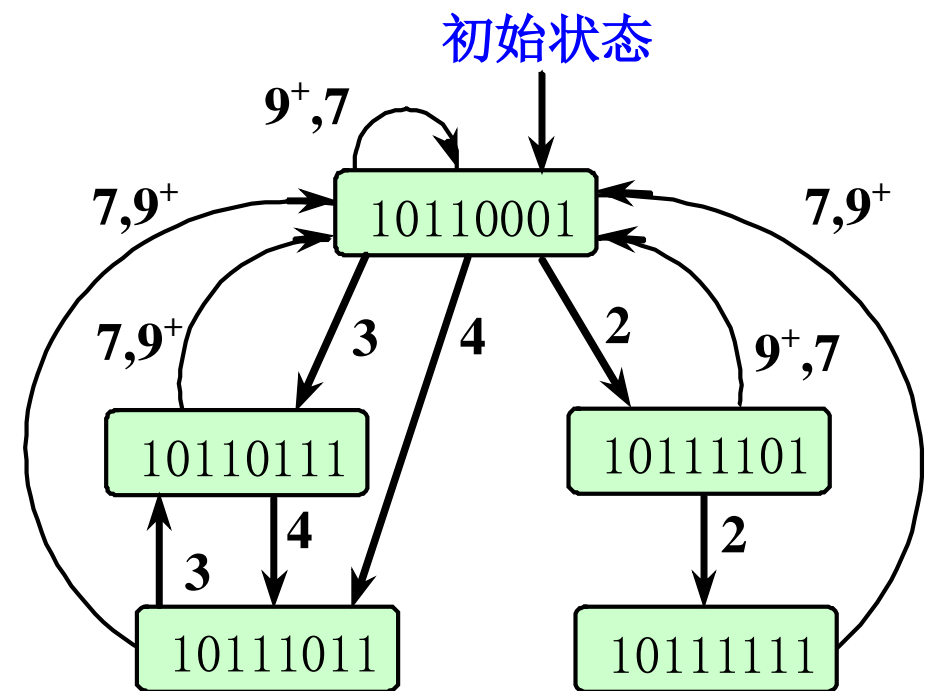
则新的冲突向量为:

$$SHR^{(j)}(C_k) \vee C_0$$

- 对于所有允许的时间间隔都按上述步骤求出其新的冲突向量, 并且把新的冲突向量作为当前冲突向量, 反复使用上述步骤, 直到不再产生新的冲突向量为止。

3.3 非线性流水线的调度

- 从初始冲突向量 C_0 出发，反复应用上述步骤，可以求得所有的冲突向量以及产生这些向量所对应的时间间隔。由此可以画出用冲突向量表示的**流水线状态转移图**。
 - **有向弧**：表示状态转移的方向
 - **弧上的数字**：表示引入后续任务（从而产生新的冲突向量）所用的时间间隔（时钟周期数）



3.3 非线性流水线的调度

对于上面的例子

(1) $C_0 = (10110001)$

引入后续任务可用的时间间隔为：2、3、4、7个时钟周期

如果采用2，则新的冲突向量为：

$$(00101100) \vee (10110001) = (10111101)$$

如果采用3，则新的冲突向量为：

$$(00010110) \vee (10110001) = (10110111)$$

如果采用4，则新的冲突向量为：

$$(00001011) \vee (10110001) = (10111011)$$

如果采用7，则新的冲突向量为：

$$(00000001) \vee (10110001) = (10110001) \quad (\text{初始冲突向量})$$

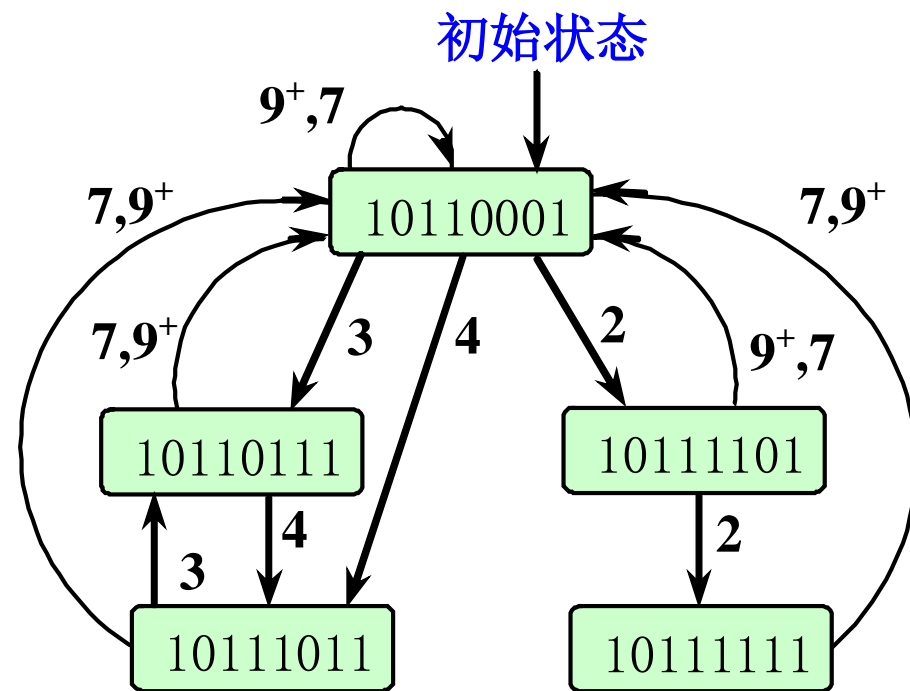
(2) 对于新向量 (1011101)，其可用的时间间隔为2个和7个时钟

周期。用类似上面的方法，可以求出其后续的冲突向量分别为

(1011111)。

(3) 对于其他新向量，也照此处理。

(4) 在此基础上，画出状态转移示意图。



3.3 非线性流水线的调度

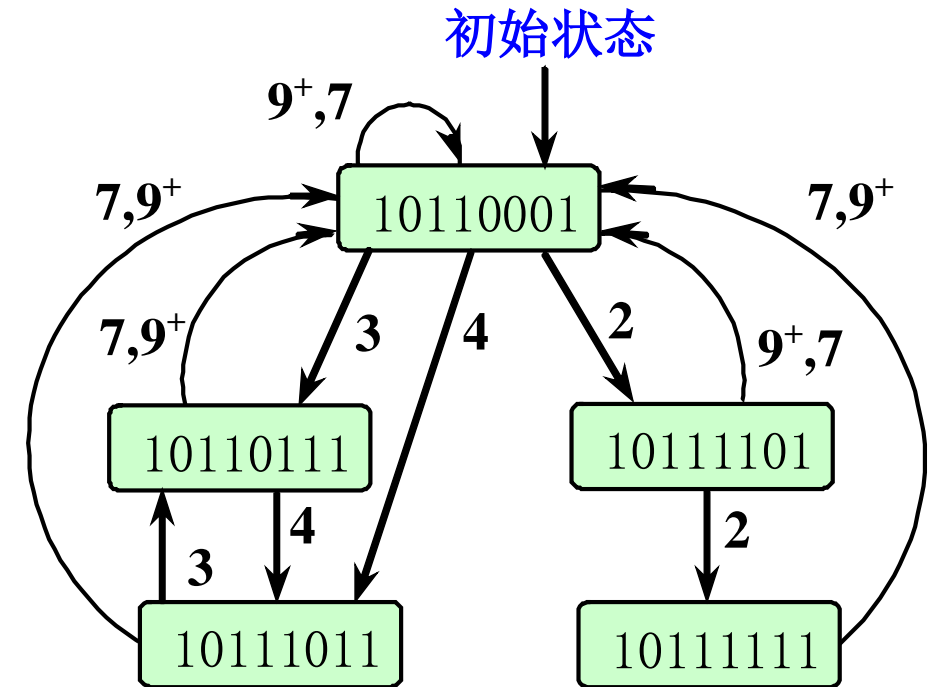
4. 根据状态转换图写出最优调度方案

- 根据流水线状态图，由初始状态出发，任何一个闭合回路即为一种调度方案。
- 列出所有可能的调度方案，计算出每种方案的平均时间间隔，从中找出其最小者即为最优调度方案。
- 上例中，各种调度方案及其平均间隔时间。
 - **最佳方案：(3, 4)**
平均间隔时间：**3.5**个时钟周期（吞吐率最高）

各种调度策略及平均延迟拍数

调度策略	平均延迟拍数
(2,7)	4.5
(2,2,7)	3.67
(3,7)	5
(3,4)	3.5
(3,4,3,7)	4.25
(3,4,7)	4.67
(4,3,7)	4.67
(4,7)	5.5
(7)	7

关于算法



3.4 流水线的相关与冲突

3.4.1 一条经典的5段流水线

- 介绍一条经典的5段RISC流水线
- 首先讨论在**非流水**情况下是如何实现的

1. 一条指令的执行过程分为以下5个周期：

- 取指令周期（IF）
 - 以程序计数器PC中的内容作为地址，从存储器中取出指令并放入指令寄存器IR；
 - 同时PC值加4（假设每条指令占4个字节），指向顺序的下一条指令。

3.4 流水线的相关与冲突

➤ 指令译码/读寄存器周期（ID）

对指令进行译码，并用IR中的寄存器地址去访问通用寄存器组，读出所需的操作数。

➤ 执行/有效地址计算周期（EX）

不同指令所进行的操作不同：

- **load和store指令**：ALU把指令中所指定的寄存器的内容与偏移量相加，形成访存有效地址。
- **寄存器—寄存器ALU指令**：ALU按照操作码指定的操作对从通用寄存器组中读出的数据进行运算。

3.4 流水线的相关与冲突

- **寄存器—立即数ALU指令**：ALU按照操作码指定的操作对从通用寄存器组中读出的操作数和指令中给出的立即数进行运算。
- **分支指令**：ALU把指令中给出的偏移量与PC值相加，形成转移目标的地址。同时，对在前一个周期读出的操作数进行判断，确定分支是否成功。
- **存储器访问 / 分支完成周期（MEM）**

该周期处理的指令只有load、store和分支指令。
其它类型的指令在此周期不做任何操作。

3.4 流水线的相关与冲突

- load和store指令

load指令：用上一个周期计算出的有效地址从存储器中读出相应的数据；

store指令：把指定的数据写入这个有效地址所指出的存储器单元。（完成）

- 分支指令

分支“成功”，就把转移目标地址送入PC。

分支指令执行完成。（完成）

3.4 流水线的相关与冲突

➤ 写回周期（WB）

ALU运算指令和load指令在这个周期把结果数据写入通用寄存器组。

ALU运算指令：结果数据来自ALU。

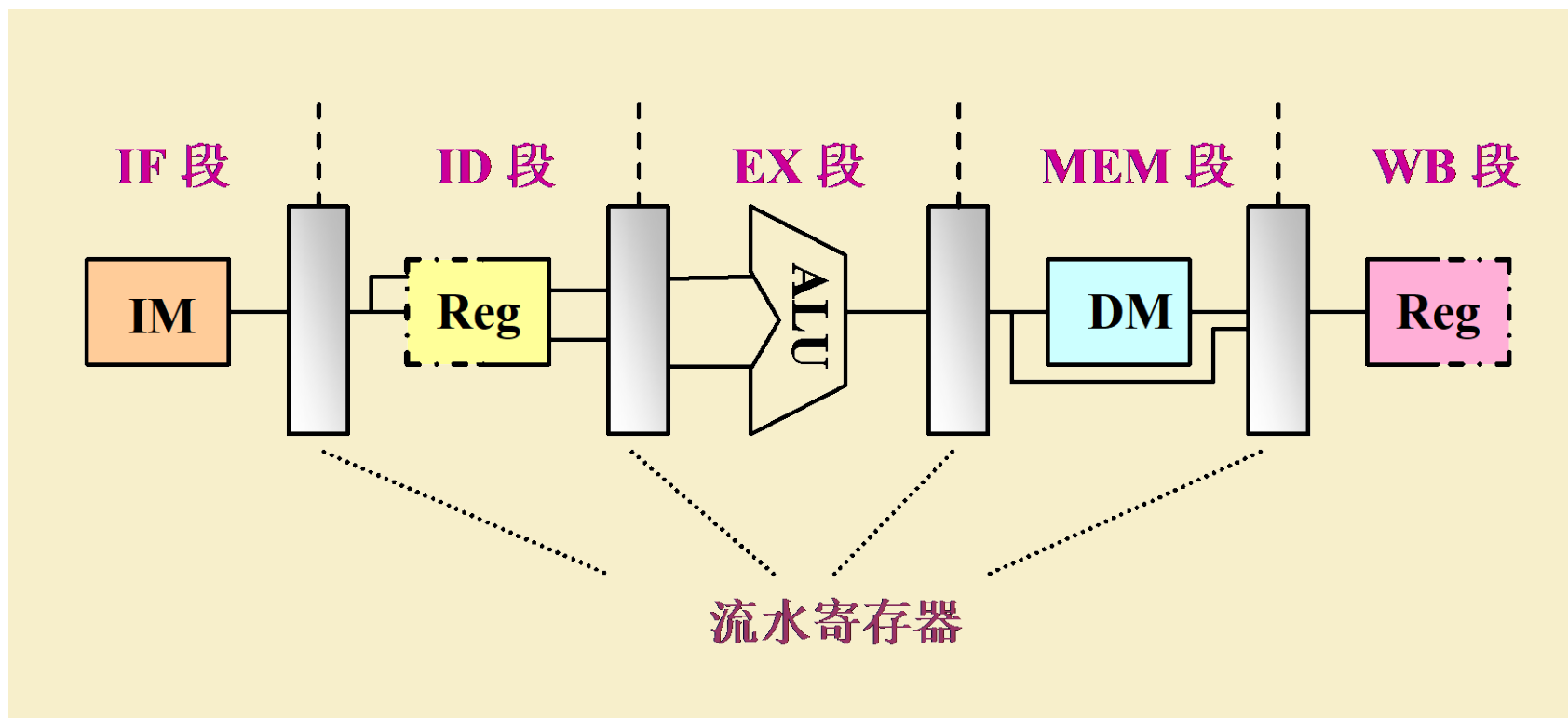
load指令：结果数据来自存储器。

在这个实现方案中：

- 分支指令需要4个时钟周期；
- store指令需要4个周期；
- 其它指令需要5个周期才能完成。

将上述实现方案修改为流水线实现

- 一条经典的5段流水线
 - 每一个周期作为一个流水段；
 - 在各段之间加上锁存器（流水寄存器）。



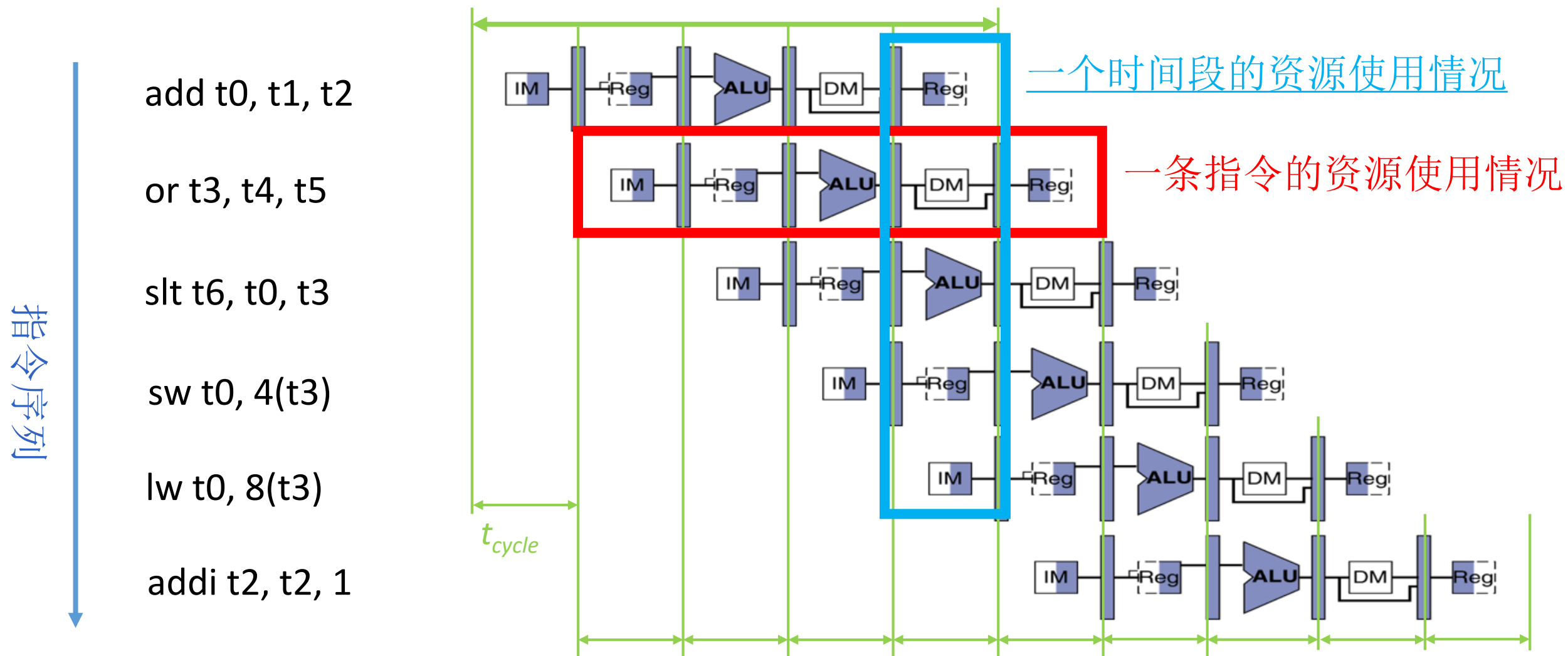
3.4 流水线的相关与冲突

5段流水线的两种描述方式

➤ 第一种描述（类似于时空图）

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

➤ 第二种描述（按时间错开的数据通路序列）



3. 采用流水线方式实现时，应解决好以下几个问题：

- 要保证不会在**同一时钟周期**要求同一个功能段**做两件不同的工作**
 例如：不能要求ALU同时做有效地址计算和算术运算。
- 避免**IF**段的访存（取指令）与**MEM**段的**访存**（读/写数据）发生**冲突**
 - 可以采用分离的指令存储器和数据存储器；
 - 一般采用分离的指令**Cache**和数据**Cache**。
- **ID**段和**WB**段都要访问同一寄存器文件（**访问寄存器冲突**）

ID段：读 WB段：写

 - 把写操作安排在时钟周期的前半拍完成，把读操作安排在后半拍完成

3.4 流水线的相关与冲突

➤ 考虑PC的问题

- 流水线为了能够每个时钟周期启动一条新的指令，就必须在每个时钟周期进行PC值的加4操作，并保留新的PC值。这种操作**必须在IF段完成**，以便为取下一条指令做好准备。

（需设置一个专门的加法器）

- 但分支指令也可能改变PC的值，而且是在MEM段进行，这会导致冲突

如何处理分支指令？

3.4 流水线的相关与冲突

3.4.2 相关与流水线冲突

- **相关：** 两条指令之间存在某种依赖关系。

如果两条指令相关，则它们就有可能不能在流水线中重叠执行或者只能部分重叠执行。

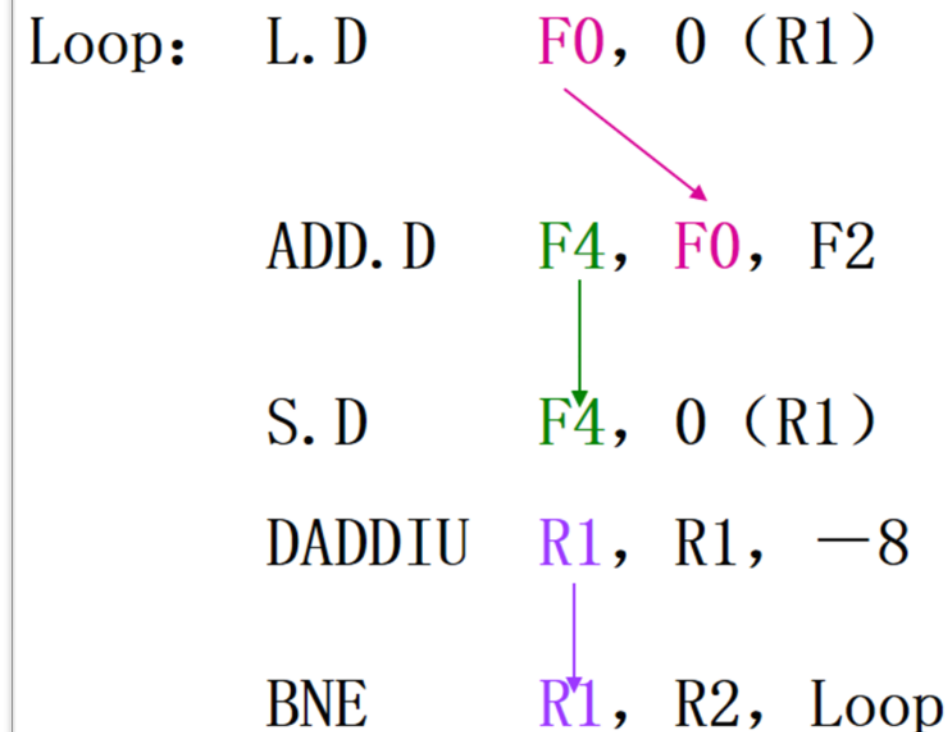
- 相关有3种类型
 - ❑ 数据相关（也称真数据相关）
 - ❑ 名相关
 - ❑ 控制相关

3.4 流水线的相关与冲突

1. 数据相关

- 对于两条指令*i* (在前, 下同) 和*j* (在后, 下同), 如果下述条件之一成立, 则称指令*j*与指令*i*数据相关。
 - 指令*j*使用指令*i*产生的结果;
 - 指令*j*与指令*k*数据相关, 而指令*k*又与指令*i*数据相关。
- 数据相关具有传递性。

**数据相关反映了数据的流动关系, 即如何
从其产生者流动到其消费者**



3.4 流水线的相关与冲突

数据相关识别：

- 当数据的流动是经过寄存器时，相关的检测比较直观和容易。
- 当数据的流动是经过存储器时，检测比较复杂。
 - 相同形式的地址其有效地址未必相同；
 - 形式不同的地址其有效地址却可能相同。

3.4 流水线的相关与冲突

2. 名相关

- **名**：指令所访问的寄存器或存储器单元的名称
- 如果两条指令使用相同的名，但是它们之间并没有数据流动，则称这两条指令存在**名相关**
- 指令j与指令i之间的名相关有两种：
 - **反相关**：如果指令j写的名与指令i读的名相同，则称指令i和j发生了反相关
$$\text{指令j写的名} = \text{指令i读的名}$$
 - **输出相关**：如果指令j和指令i写相同的名，则称指令i和j发生了输出相关

$$\text{指令j写的名} = \text{指令i写的名}$$

3.4 流水线的相关与冲突

名相关的特点：

- 名相关的两条指令之间并没有数据的传送
 - 如果一条指令中的名改变了，并不影响另外一条指令的执行
 - 换名技术
 - **换名技术**：通过改变指令中操作数的名来消除名相关
 - 对于寄存器操作数进行换名称为**寄存器换名**
- 既可以用编译器静态实现，也可以用硬件动态完成

3.4 流水线的相关与冲突

例如：考虑下述代码：

- | | | | |
|---|--------|--------------|------------|
| ① | DIV. D | F2, F8, F4 | ①和②是否存在相关？ |
| ② | ADD. D | F8, F0, F12 | ②与③之间呢？ |
| ③ | SUB. D | F10, F8, F14 | |

进行寄存器换名（F8换成S）后，变成：

DIV. D	F2, F8, F4
ADD. D	S, F0, F12
SUB. D	F10, S, F14

3.4 流水线的相关与冲突

3. 控制相关

- **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。

- 请看一个示例：

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

3.4 流水线的相关与冲突

➤ 控制相关带来了以下两个限制：

- 与一条分支指令控制相关的指令不能被移到该分支之前。否则这些指令就不受该分支控制了。

对于上述的例子，**then** 部分中的指令不能移到**if**语句之前。

- 如果一条指令与某分支指令不存在控制相关，就不能把该指令移到该分支之后。

对于上述的例子，不能把**S**移到**if**语句的**then** 部分中

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

3.4 流水线的相关与冲突

流水线冲突

流水线冲突是指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行

流水线冲突有3种类型：

- **结构冲突**：因硬件资源满足不了指令重叠执行的要求而发生的冲突
- **数据冲突**：当指令在流水线中重叠执行时，因需要用到前面指令的执行结果而发生的冲突
- **控制冲突**：流水线遇到分支指令和其它会改变PC值的指令所引起的冲突

3.4 流水线的相关与冲突

冲突产生的问题:

- 导致错误的执行结果
- 流水线可能会出现停顿, 从而降低流水线的效率和实际的加速比
- 我们约定

当一条指令被暂停时, 在该暂停指令之后流出的所有指令都要被暂停, 而在该暂停指令之前流出的指令则继续进行 (否则就永远无法消除冲突)



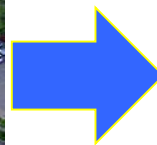
3.4 流水线的相关与冲突

1. 结构冲突

- 在流水线处理机中，为了能够使各种组合的指令都能顺利地重叠执行，需要对功能部件进行流水或重复设置资源。
- 如果某种指令组合因为资源冲突而不能正常执行，则称该处理机有结构冲突。
- 常见的导致结构冲突的原因：
 - 功能部件不是完全流水
 - 资源份数不够



一停二看三通过



3.4 流水线的相关与冲突

➤ 结构冲突举例：访存冲突

有些流水线处理机只有一个存储器，将数据和指令放在一起，访存指令会导致**访存冲突**。

□ 解决方法 I:

设置相互独立的指令存储器和数据存储器
或设置相互独立的指令Cache和数据Cache。

□ 解决办法 II : 插入暂停周期

(“流水线气泡”或“气泡”)

处理器 | 缓存 | 主板 | 内存 | SPD | 显卡 | 测试分数 | 关于 |

处理器

名字	Intel Core i7 10700				
代号	Comet Lake	TDP	65.0 W		
插槽	Socket 1200 LGA				
工艺	14 纳米	核心电压	0.752 V		
规格	Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz				
系列	6	型号	5	步进	5
扩展系列	6	扩展型号	A5	修订	Q0/G1
指令集	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3				

时钟 (核心 #0)

核心速度	4589.88 MHz
倍频	x 46.0 (8 - 47)
总线速度	99.78 MHz
额定 FSB	

缓存

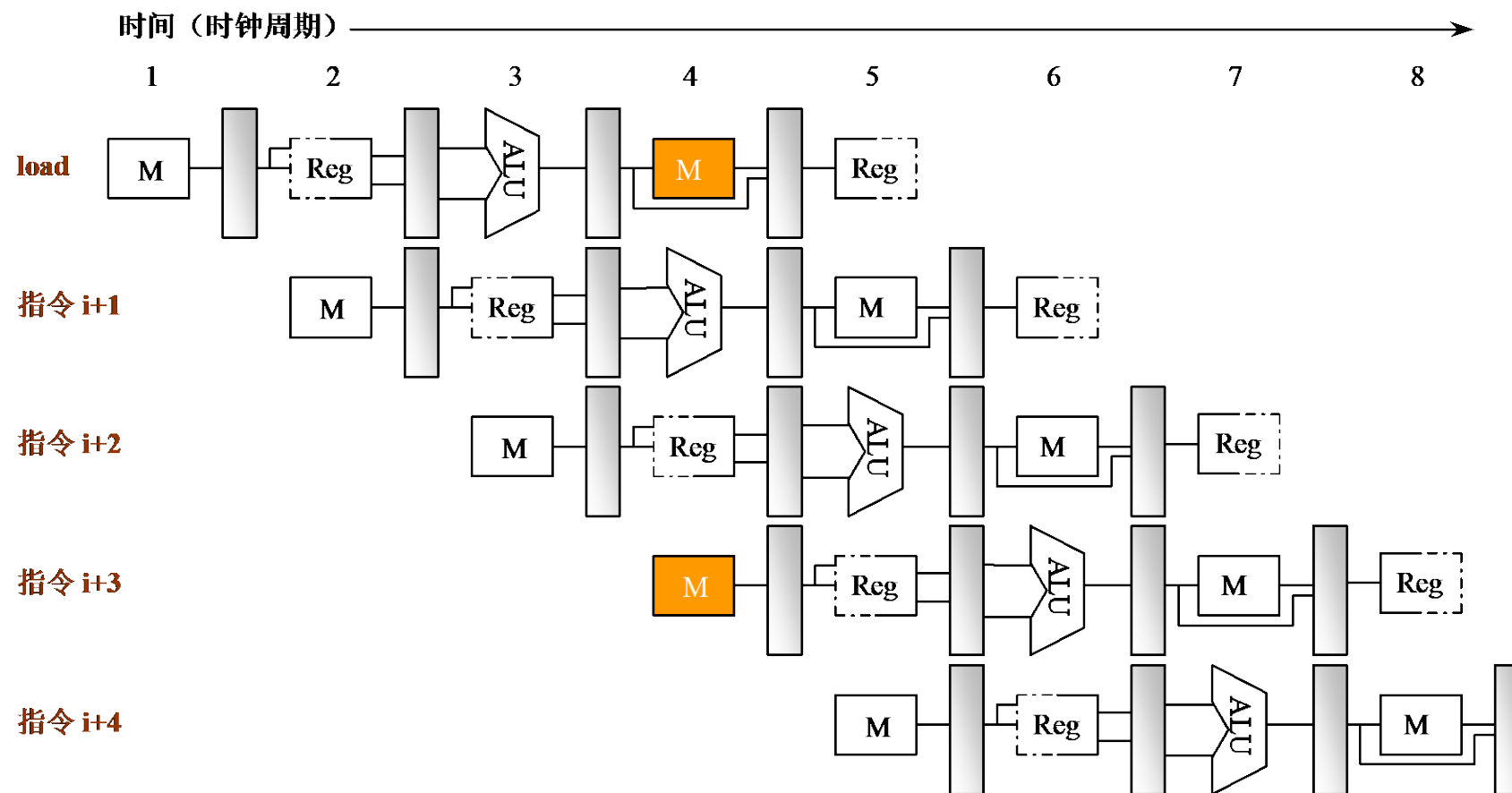
一级 数据	8 x 32 KBytes	8-way
一级 指令	8 x 32 KBytes	8-way
二级	8 x 256 KBytes	4-way
三级	16 MBytes	16-way

已选择 处理器 #1

核心数 8

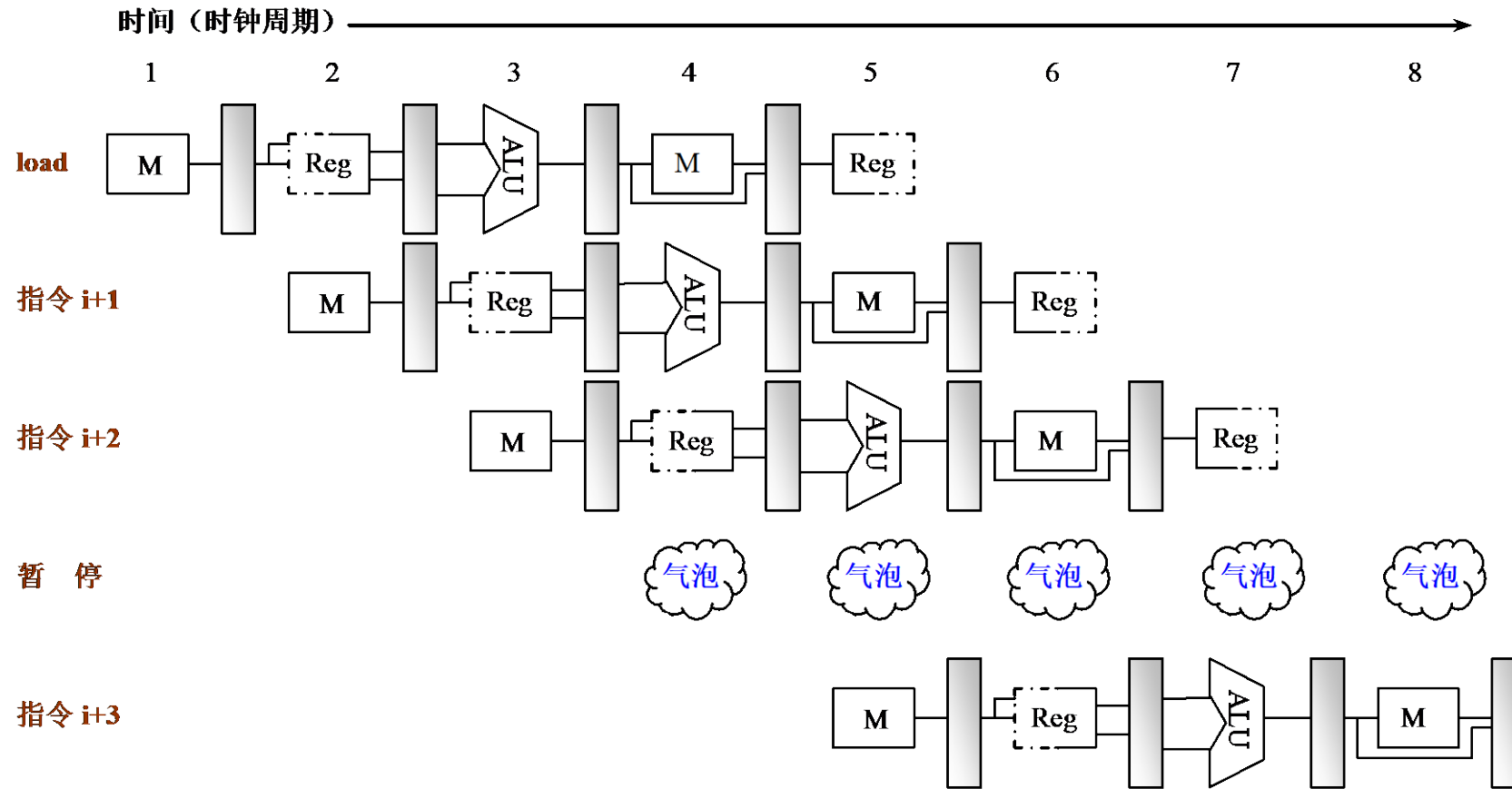
线程数 16

3.4 流水线的相关与冲突



由于访问同一个存储器而引起的结构冲突

3.4 流水线的相关与冲突



为消除结构冲突而插入的流水线气泡

引入暂停后的时空图

指令编号	时钟周期									
	1	2	3	4	5	6	7	8	9	10
load	IF	ID	EX	MEM	WB					
指令i+1		IF	ID	EX	MEM	WB				
指令i+2			IF	ID	EX	MEM	WB	WB		
指令i+3				stall	IF	ID	EX	MEM	WB	
指令i+4						IF	ID	EX	MEM	WB
指令i+5							IF	ID	EX	MEM

有时流水线设计者允许结构冲突的存在，主要原因：减少硬件成本

- 如果把流水线中的所有功能单元完全流水化，或者重复设置足够份数，那么所花费的成本将相当高。

3.4 流水线的相关与冲突

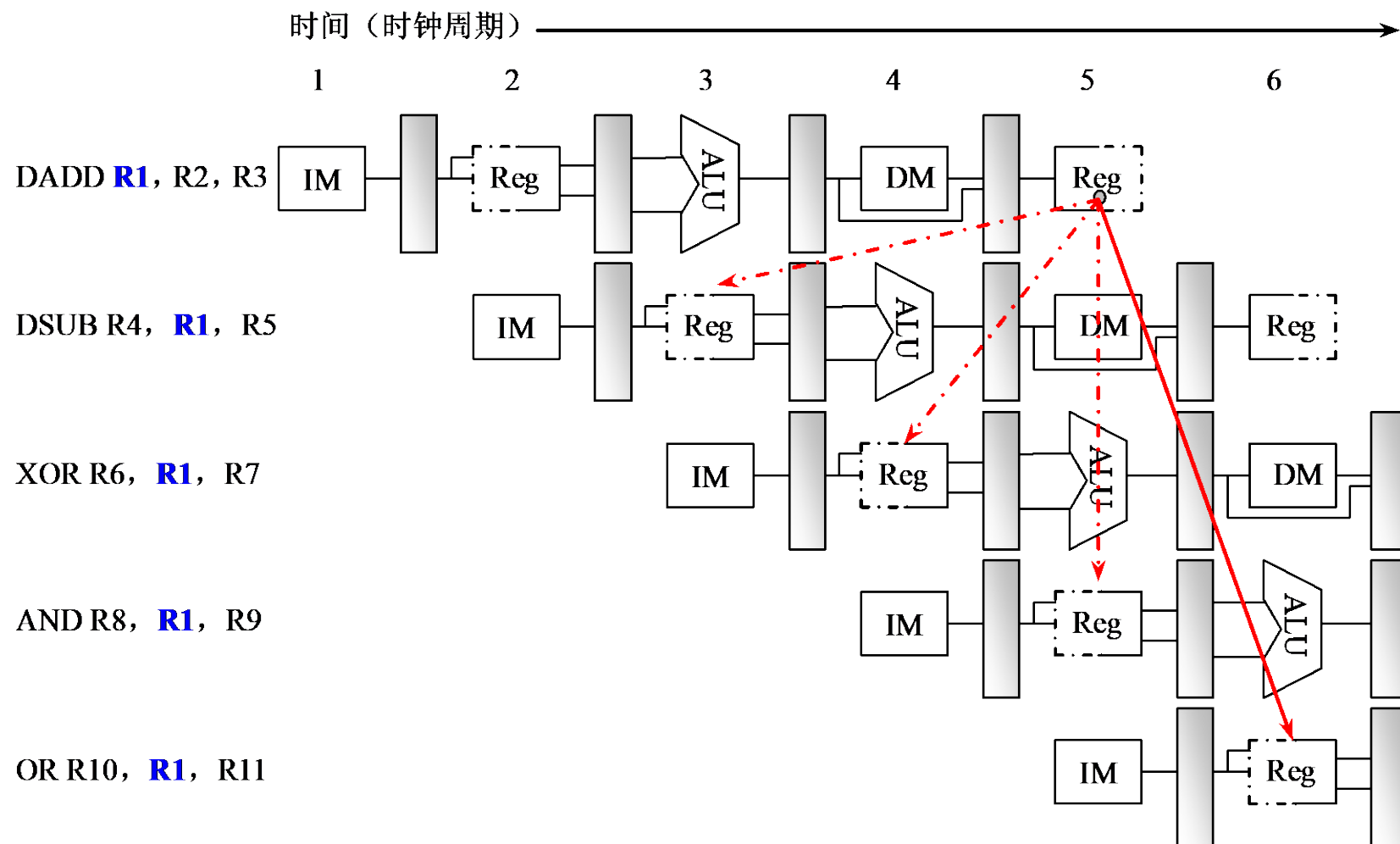
2. 数据冲突

当**相关的指令靠得足够近**时，它们在流水线中的重叠执行或者重新排序会改变指令读/写操作数的顺序，使之不同于它们串行执行时的顺序，则发生了**数据冲突**。

例如 (时空图)：

```
DADD  R1, R2, R3
DSUB  R4, R1, R5
AND   R6, R1, R7
OR    R8, R1, R9
XOR   R10, R1, R11
```

3.4 流水线的相关与冲突



流水线的数据冲突举例

3.4 流水线的相关与冲突

- 根据指令读访问和写访问的顺序，可以将数据冲突分为3种类型。

考虑两条指令*i*和*j*，且*i*在*j*之前进入流水线，可能发生的数据冲突有：

□ 写后读冲突（RAW）

在 *i* 写入之前，*j* 先去读

j 读出的内容是错误的!!!

这是最常见的一种数据冲突，它对应于真数据相关

3.4 流水线的相关与冲突

□ 写后写冲突 (WAW)

在 i 写入之前, j 先写最后写入的结果是 i 的错误! 这种冲突对应于输出相关

写后写冲突仅发生在这样的流水线中:

- 流水线中不只一个段可以进行写操作;
- 指令被重新排序了

前面介绍的5段流水线不会发生写后写冲突
(只在WB段写寄存器)

什么情况下
出现冲突?

3.4 流水线的相关与冲突

□ 读后写冲突（WAR）

在 i 读之前， j 先写

i 读出的内容是错误的！

由反相关引起

这种冲突仅发生在这样的情况下：

- 有些指令的写结果操作提前了，而且有些指令的读操作滞后了；
- 指令被重新排序了

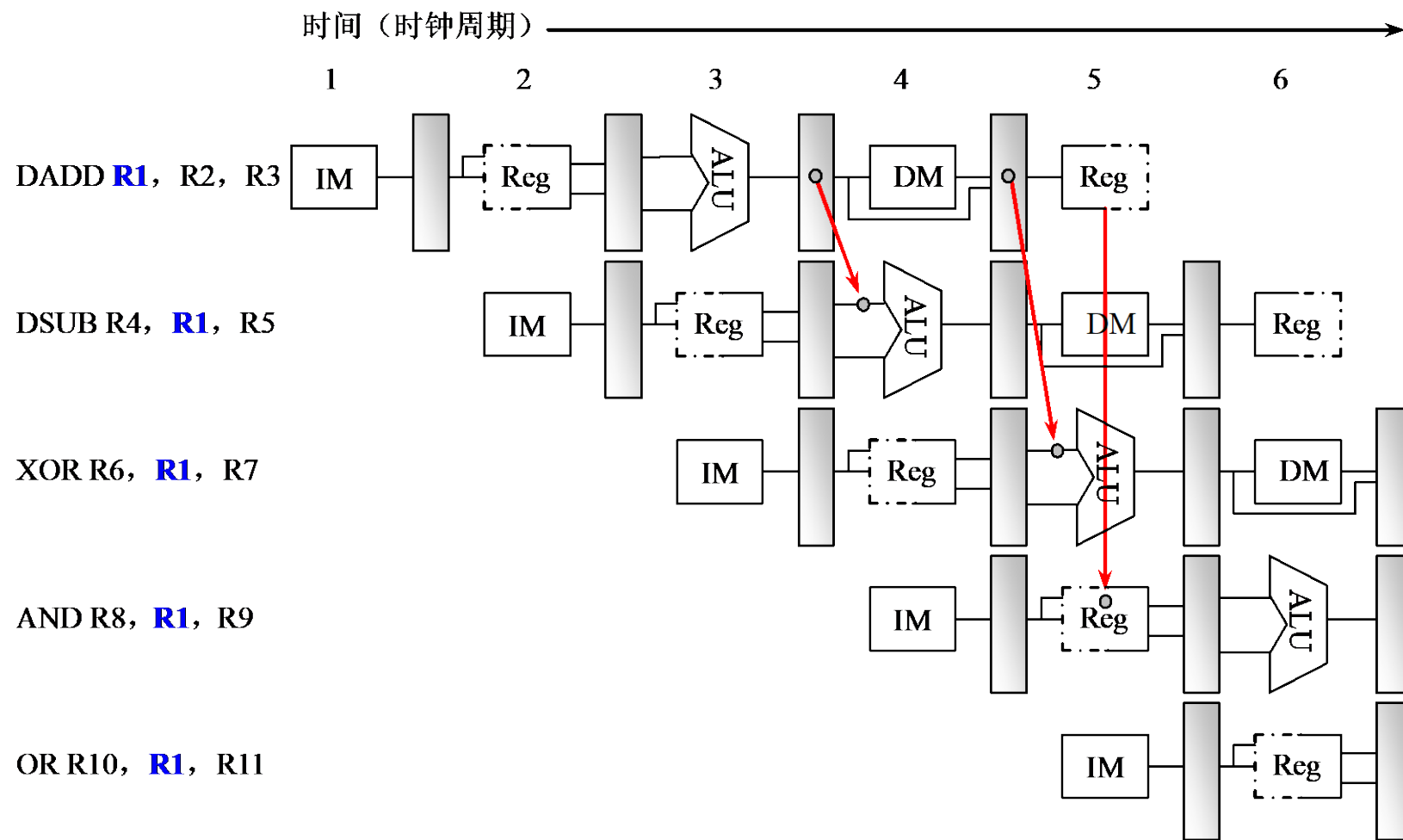
总结： 两条指令对同一寄存器读写相关性

3.4 流水线的相关与冲突

- 通过**定向技术**减少数据冲突引起的停顿
(定向技术也称为旁路或短路)
 - **关键思想**：在计算结果尚未出来之前，后面等待使用该结果的指令并不真正立即需要该计算结果，如果能够**将该计算结果从其产生的地方直接送到其它指令需要它的地方**，那么就可以避免停顿。

抄近道

3.4 流水线的相关与冲突



采用定向技术后的流水线数据通路

3.4 流水线的相关与冲突

连接通路

+

控制开关

- 定向的实现
 - EX段和MEM段之间的流水寄存器中保存的ALU运算结果总是回送到ALU的入口。
 - 当定向硬件检测到前一个ALU运算结果写入的寄存器就是当前ALU操作的源寄存器时，那么控制逻辑就选择定向的数据作为ALU的输入，而不采用从通用寄存器组读出的数据。
- 结果数据不仅可以从某一功能部件的输出定向到其自身的输入，而且还可以定向到其它功能部件的输入。

3.4 流水线的相关与冲突

➤ 需要停顿的数据冲突

- 并不是所有的数据冲突都可以用定向技术来解决。

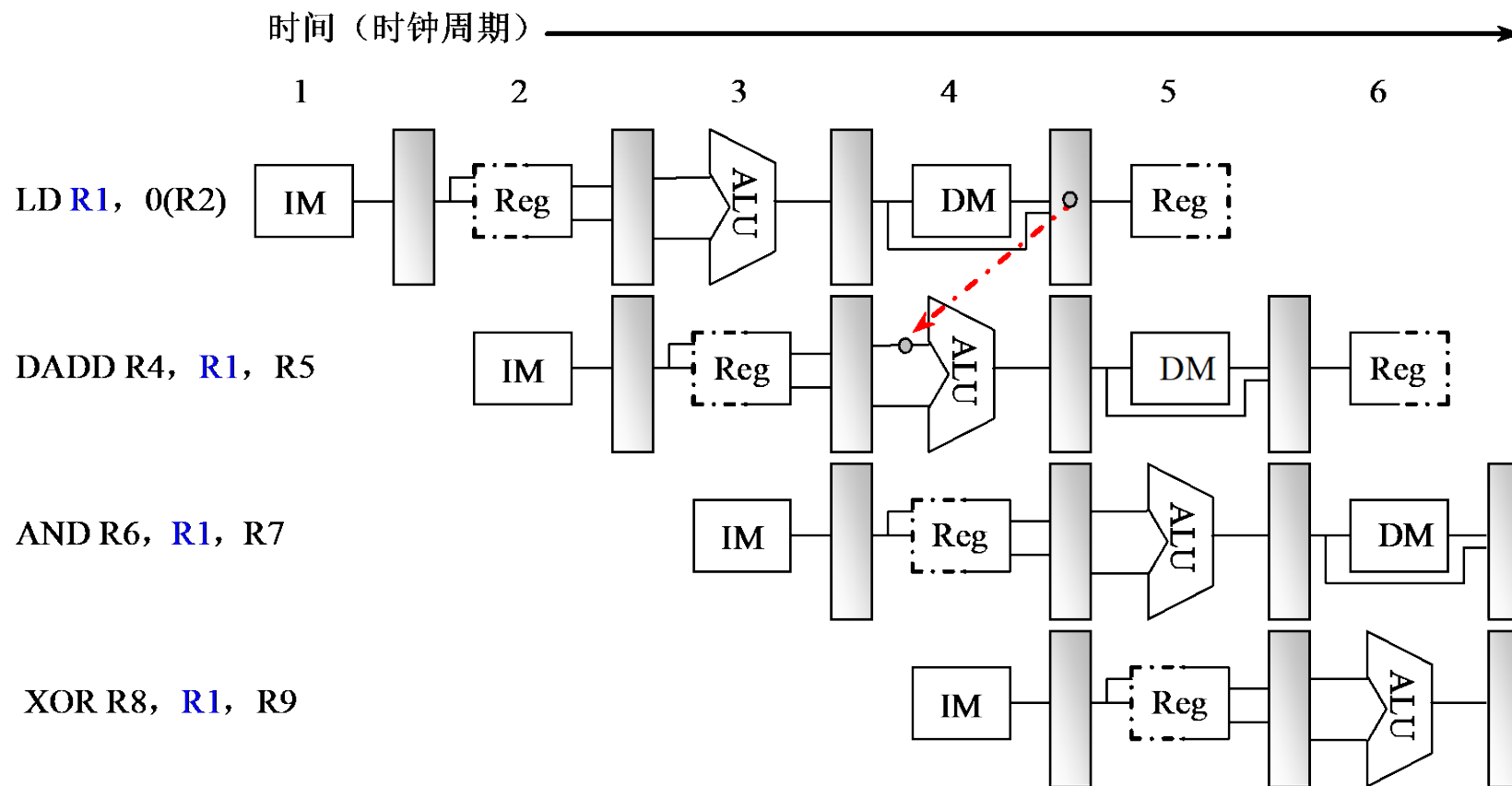
举例：

```
LD      R1, 0(R2)
DADD    R4, R1, R5
AND      R6, R1, R7
XOR     R8, R1, R9
```

- 增加流水线互锁机制（Pipeline Interlock），插入“暂停”。

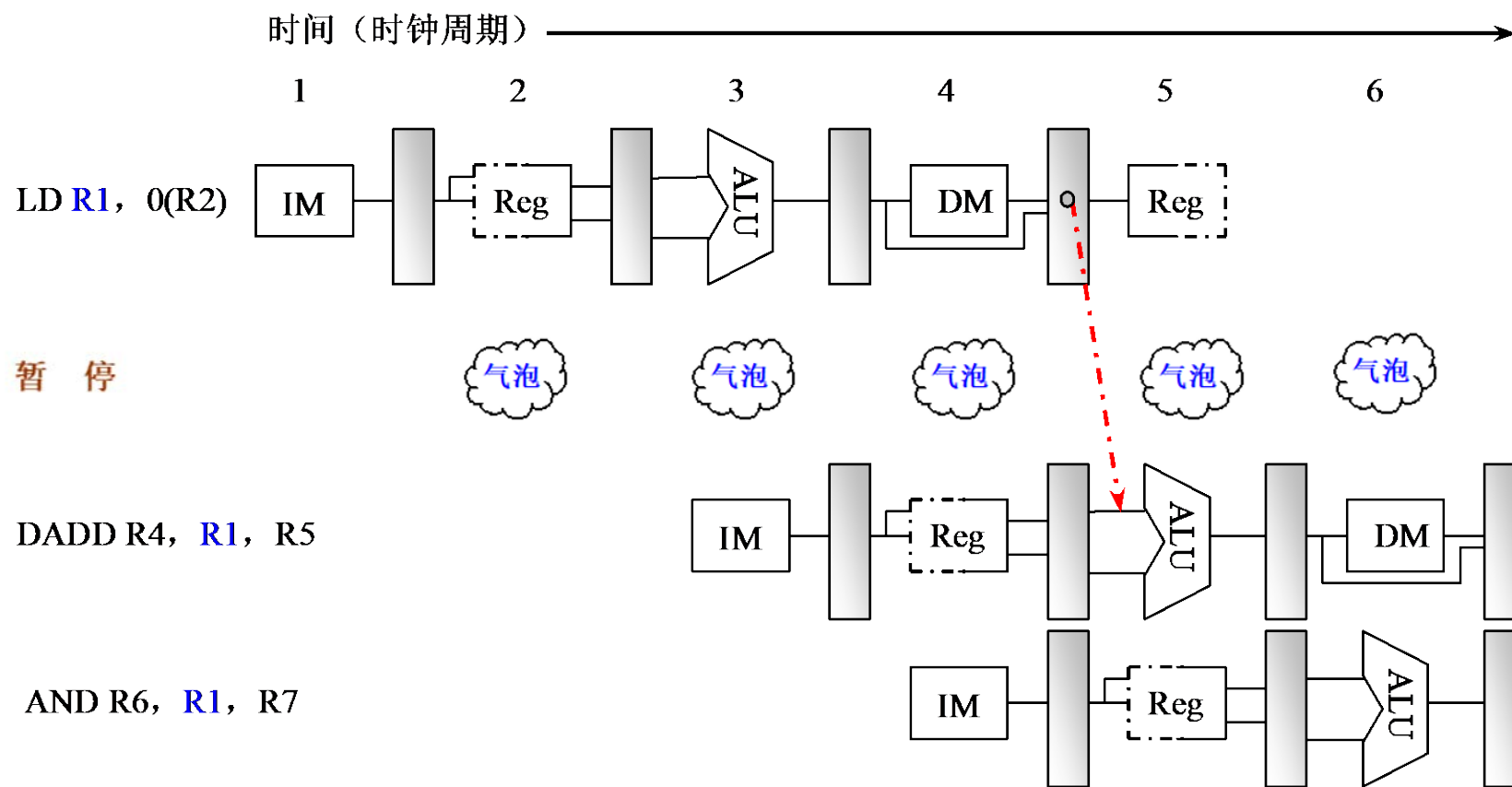
作用：检测发现数据冲突，并使流水线停顿，直至冲突消失。

3.4 流水线的相关与冲突



无法将LD指令的结果定向到DADD指令

3.4 流水线的相关与冲突



必须至少停顿一个周期

依靠编译器解决数据冲突（软件方法）

让编译器重新组织指令顺序来消除冲突，这种技术称为**指令调度**或**流水线调度**

□ 举例

下列表达式生成没有暂停的指令序列：

$A = B + C$ ；

$D = E - F$ ；

循环展开、VLIW等技术，更多软件调度方法见第6章

调度前的代码		
LD	Rb, B	
LD	Rc, C	
DADD	Ra, Rb, Rc	
SD	Ra, A	
LD	Re, E	
LD	Rf, F	
DSUB	Rd, Re, Rf	
SD	Rd, D	

3.4 流水线的相关与冲突

3. 控制冲突

- 执行分支指令的结果有两种
 - **分支成功**: PC值改变为分支转移的目标地址在条件判定和转移地址计算都完成后, 才改变PC值
 - **分支失败**: PC的值保持正常递增, 指向顺序的下一条指令
- 处理分支指令最简单的方法: **等结果**
 - “冻结”或者“排空”流水线 **优点: 简单**
 - 前述5段流水线中, 改变PC值是在MEM段进行的
给流水线带来了**3个时钟周期的延迟**

简单处理分支指令：分支成功的情况

分支指令	IF	ID	EX	MEM	WB					
分支目标指令		IF	stall	stall	IF	ID	EX	MEM	WB	
分支目标指令+1						IF	ID	EX	MEM	WB
分支目标指令+2							IF	ID	EX	MEM
分支目标指令+3								IF	ID	EX

分支延迟

后果？

3.4 流水线的相关与冲突

- 把由分支指令引起的延迟称为分支延迟
- 分支指令在目标代码中出现的频度
 - 每3~4条指令就有一条是分支指令

假设：分支指令出现的频度是30%

流水线理想 $CPI=1$

那么：流水线的实际 $CPI = ?$

$$\text{实际CPI} = \text{理想CPI} + \text{平均每条指令的停顿周期数} \quad 1 + 3 * 0.3 = 1.9$$



3.4 流水线的相关与冲突

可采取措施来减少分支延迟

- 在流水线中尽早判断出分支转移是否成功；
- 尽早计算出分支目标地址。

假设：

这两步工作被提前到ID段完成，即分支指令是在ID段的末尾执行完成，所带来的分支延迟为一个时钟周期

分支指令	IF	ID	EX	MEM	WB					
分支目标指令		stall	IF	ID	EX	MEM	WB			
分支目标指令+1				IF	ID	EX	MEM	WB		
分支目标指令+2					IF	ID	EX	MEM	WB	
分支目标指令+3						IF	ID	EX	MEM	WB

3.4 流水线的相关与冲突

3种通过软件（编译器）来减少分支延迟的方法

1. 预测分支失败
2. 预测分支成功
3. （隐藏）分支延迟

共同点：

- 对分支的处理方法在程序的执行过程中始终是不变的，是**静态**的
- 要么总是预测分支成功，要么总是预测分支失败

3.4 流水线的相关与冲突

1. 预测分支失败

- 允许分支指令后的指令继续在流水线中流动，就好象什么都没发生似的；
- 若确定分支失败，将分支指令看作是一条普通指令，流水线正常流动；
- 若确定分支成功，流水线就把在分支指令之后取出的所有指令转化为空操作，并按分支目地重新取指令执行。



要保证：分支结果出来之前**不能改变处理机的状态**，以便一旦猜错时，处理机能够回退到原先的状态。

流水线对分支的处理过程

分支指令 i(失败)	IF	ID	EX	MEM	WB				
指令 i+1		IF	ID	EX	MEM	WB			
指令 i+2			IF	ID	EX	MEM	WB		
指令 i+3				IF	ID	EX	MEM	WB	
指令 i+4					IF	ID	EX	MEM	WB

分支指令 i(成功)	IF	ID	EX	MEM	WB				
指令 i+1		IF	ID	idle	idle	idle			
分支目标 j			IF	ID	EX	MEM	WB		
分支目标 j+1				IF	ID	EX	MEM	WB	
分支目标 j+2					IF	ID	EX	MEM	WB

3.4 流水线的相关与冲突

2. 预测分支成功

假设分支转移成功，并从分支目标地址处取指令执行

起作用的前题：先知道分支目标地址，后知道分支是否成功

前述5段流水线中，这种方法没有任何好处

3. （隐藏）延迟分支

主要思想：

从逻辑上“延长”分支指令的执行时间。把延迟分支看成是由原来的分支指令和若干个延迟槽构成，不管分支是否成功，都要按顺序执行延迟槽中的指令。

具有一个分支延迟槽的流水线的执行过程

分支失败	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	ID	EX	MEM	WB			
	指令 i+2			IF	ID	EX	MEM	WB		
	指令 i+3				IF	ID	EX	MEM	WB	
	指令 i+4					IF	ID	EX	MEM	WB

分支成功	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	ID	EX	MEM	WB			
	分支目标指令j			IF	ID	EX	MEM	WB		
	分支目标指令j+1				IF	ID	EX	MEM	WB	
	分支目标指令j+2					IF	ID	EX	MEM	WB

分支延迟槽中的指令“掩盖”了流水线原来必需插入的暂停周期。

3.4 流水线的相关与冲突

分支延迟指令的调度

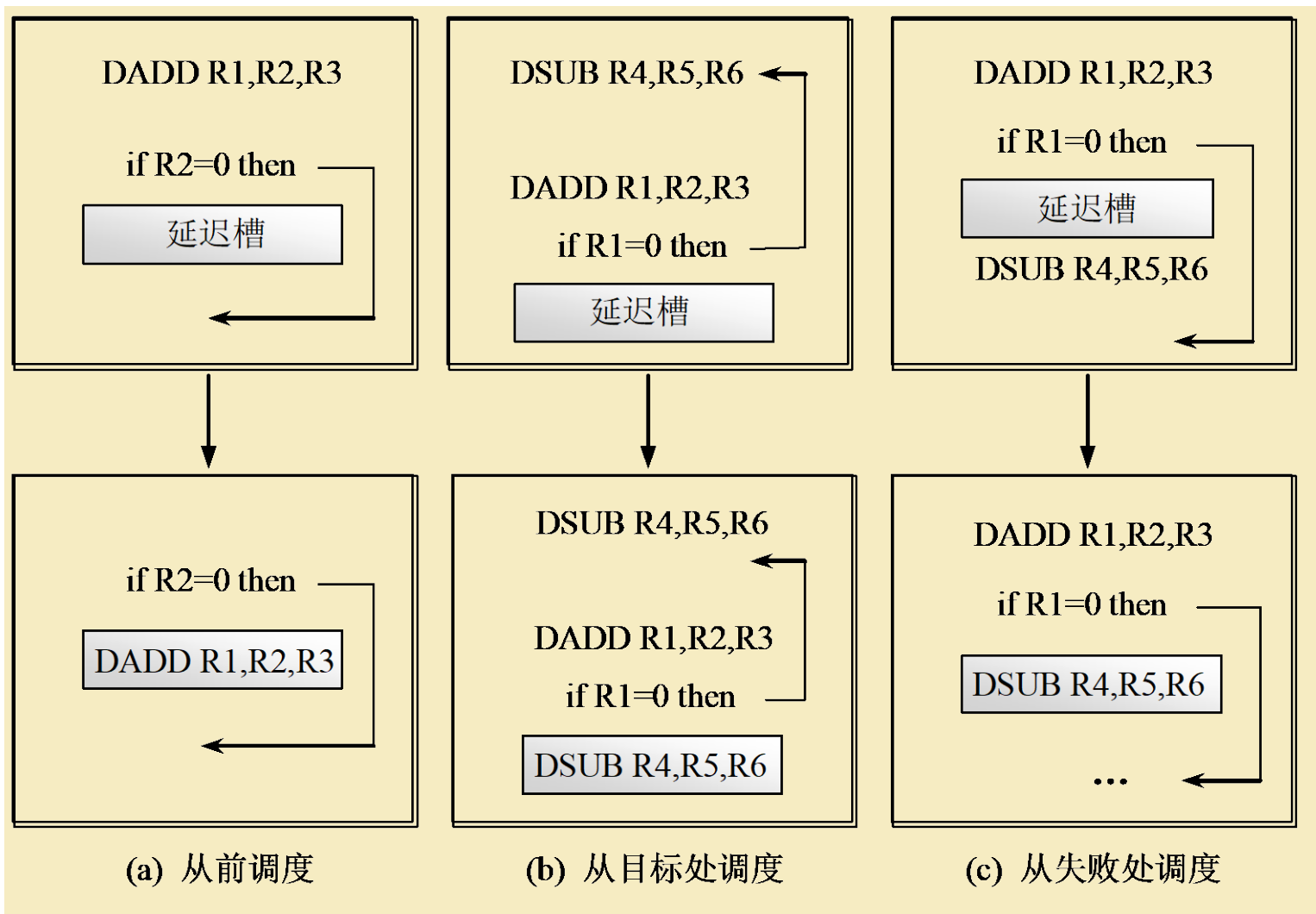
任务：在延迟槽中放入有用的指令

由编译器完成。能否带来好处取决于编译器能否把有用的指令调度到延迟槽中。

三种调度方法：

- 从前调度
- 从目标处调度
- 从失败处调度

调度前和调度后的代码



3.4 流水线的相关与冲突

三种方法的要求及效果

调 度 策 略	对调度的要求	什么情况下起作用？
从 前 调 度	被调度的指令必须与分支无关	任何情况
从目标处调度	必须保证在分支失败时执行被调度的指令不会导致错误。 有可能需要复制指令	分支成功时(但由于复制指令，有可能会增大程序空间)
从失败处调度	必须保证在分支成功时执行被调度的指令不会导致错误	分支失败时



3.4 流水线的相关与冲突

分支延迟受到两个方面的限制：

- 可以被放入延迟槽中的指令要满足一定的条件；
- 编译器预测分支转移方向的能力。(`if(likely(a > b))...`)

进一步改进：分支取消机制（取消分支）

当分支的实际执行方向和事先所预测的一样时，执行分支延迟槽中的指令，否则就将分支延迟槽中的指令转化成一个空操作。

“预测成功-取消”分支的执行过程

分支失败	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	idle	idle	idle	idle			
	指令 i+2			IF	ID	EX	MEM	WB		
	指令 i+3				IF	ID	EX	MEM	WB	
	指令 i+4					IF	ID	EX	MEM	WB

分支成功	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	ID	EX	MEM	WB			
	分支目标指令j			IF	ID	EX	MEM	WB		
	分支目标指令j+1				IF	ID	EX	MEM	WB	
	分支目标指令j+2					IF	ID	EX	MEM	WB

预测分支成功的情况下，分支取消机制的执行情况

本章作业

3.8

3.10

3.11(加 “在图3.33所示...”)

1. 在MIPS流水线(教材图3.33)上运行如下代码系列

```
LOOP: LW      R1, 0 (R2)
      ADDIU   R1, R1, #1
      SW      R1, 0 (R2)
      ADDIU   R2, R2, #4
      DSUB    R4, R3, R2
      BNEZ    R4, LOOP
```

其中R3的初始值是R2+200，该流水线有正常的定向路径，在一个时钟周期中对同一寄存器的读操作和写操作可以用过寄存器“定向”，采用预测分支失败的策略处理分支指令，所有的存储器访问都命中Cache，请在下表 填写执行一轮循环的时空图，然后计算完成全部循环所需的周期数

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW R1, 0 (R2)															
ADDIU R1, R1, #1															
SW R1, 0 (R2)															
ADDIU R2, R2, #4															
DSUB R4, R3, R2															
BNEZ R4, LOOP															
LW R1, 0 (R2)															

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW R1, 0 (R2)	F	D	E	M	W										
ADDIU R1, R1, #1		F	D	S	E	M	W								
SW R1, 0 (R2)			F	S	D	E	M	W							
ADDIU R2, R2, #4					F	D	E	M	W						
DSUB R4, R3, R2						F	D	E	M	W					
BNEZ R4, LOOP							F	S	D	E	M	W			
LW R1, 0 (R2)								S	F	E	D	E	M	W	

总周期数：9 X 50 + 3 = 453

这里有数据相关，分支指令译码阶段判断分支需要等上一条指令执行阶段完毕才可以执行，故增加了一个停顿

1. E. S. Davidson, "The Design and Control of Pipelined Function Generators," Proc. 1971 Int'l. IEEE Conf. on Systems, Networks and Computers, Oaxtepec, Mexico. January 1971.
(非线性流水线调度)

[Return](#)