



计算机系统结构

第三章 流水线技术

主 讲：刘超

中国地质大学（武汉）计算机学院

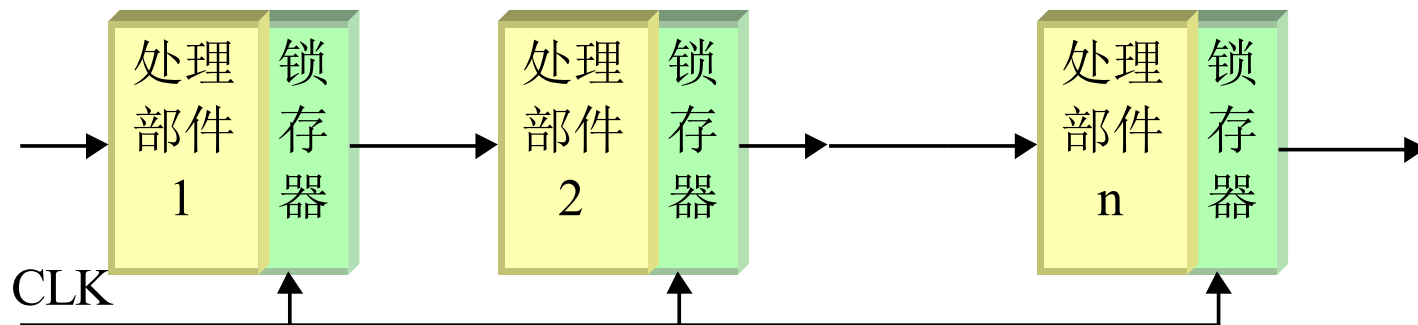


小节目录

- 3.1 流水线的基本概念
- 3.2 流水线的性能指标
- 3.3 非线性流水线的调度
- 3.4 流水线的相关与冲突
- 3.5 习题

3.1 流水线的基本概念

- **流水工作方式**：将一个计算任务细分成若干个子任务，每个子任务由专门的部件处理，**多个计算任务依次进行并行处理**。



3.1.1 流水线的由来

- 考虑设计一个洗衣机的工作流程，假定它有三道工序：洗涤、清洗、甩干。每个环节为**5分钟**。
- 需要完成任务为**3批**，则考虑下述两种工作方式的工作效率：
 - 顺序工作方式（串行工作方式）
 - 重叠工作方式（交叉工作方式）

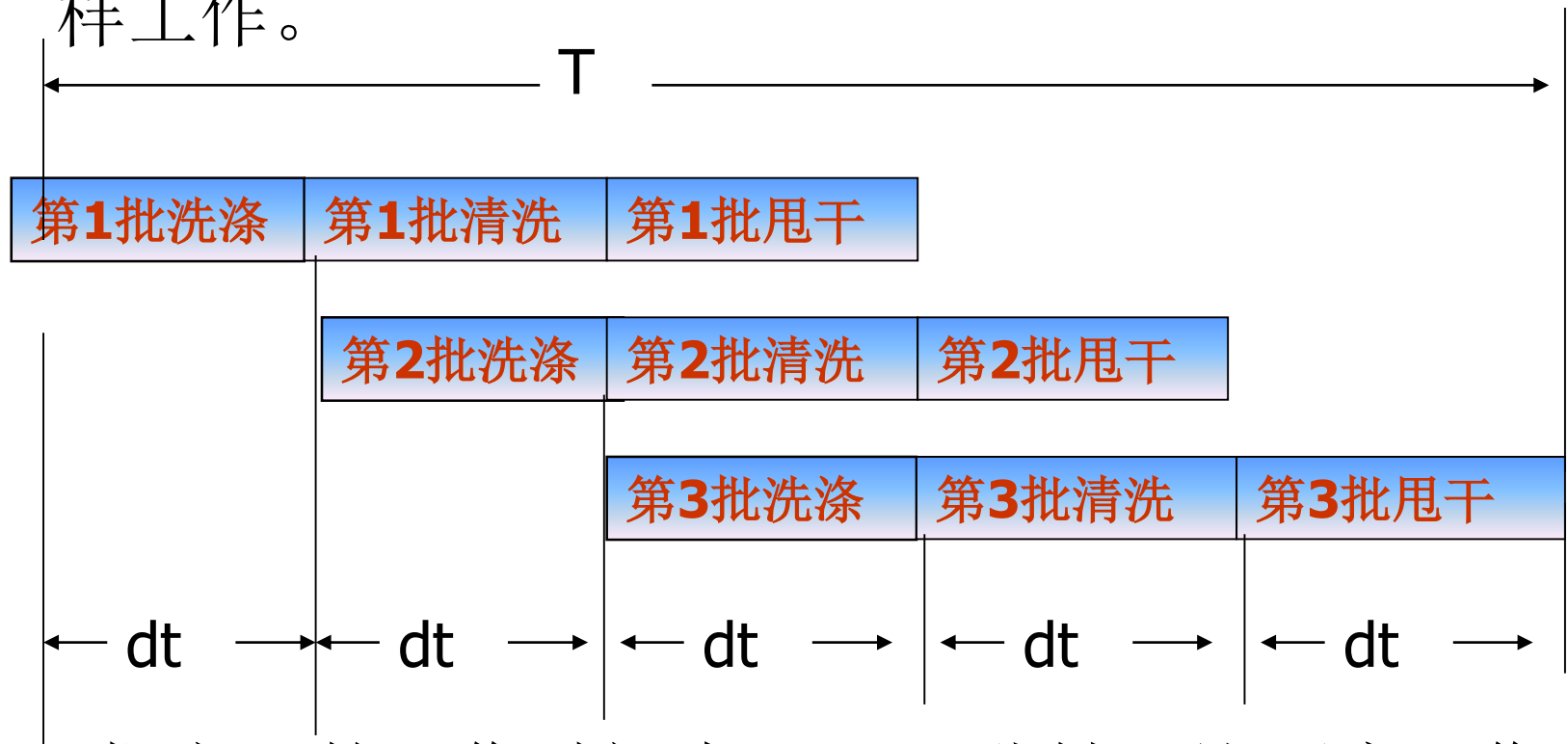
第一种：顺序工作方式

第1批 洗涤	第1批 清洗	第1批 甩干	第2批 洗涤	第2批 清洗	第2批 甩干	第3批 洗涤	第3批 清洗	第3批 甩干
-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

- 3批衣服的整体工作时间为 $3 \times 3 \times 5 = 45$ 分钟

第二种：重叠工作方式

- 设计三个部件，可以同时工作，每个部件只做一样工作。



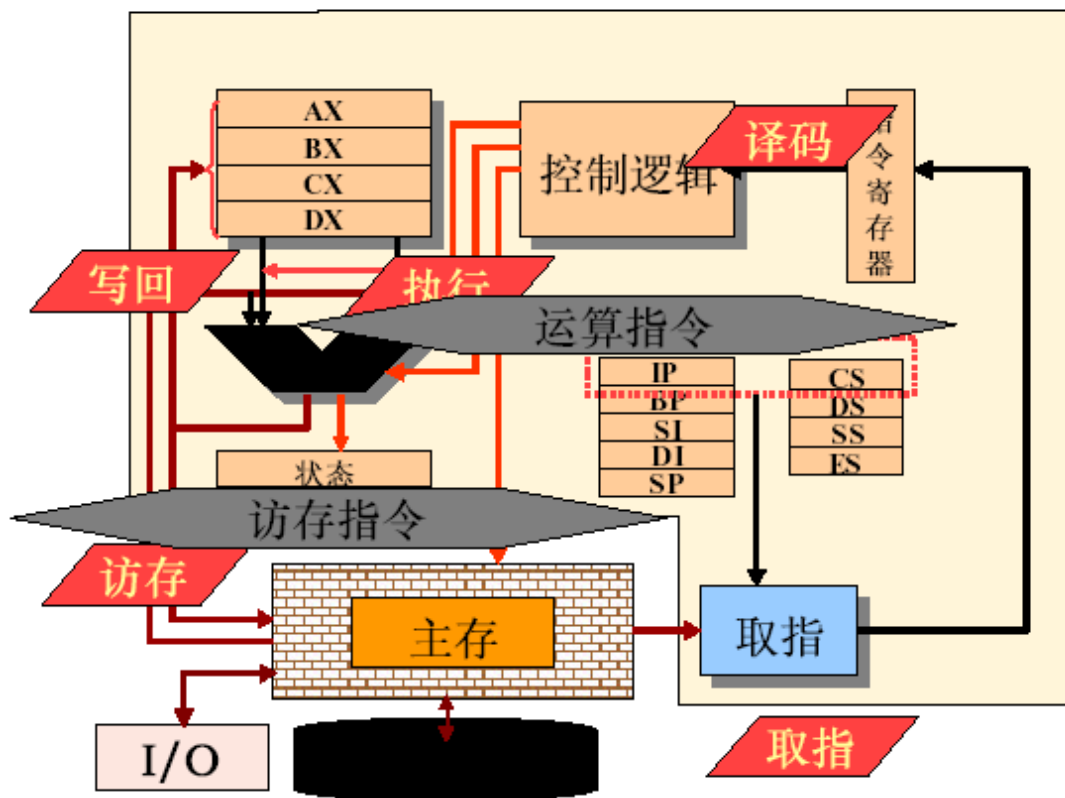
- 3批衣服的工作时间为 $5 \times 5 = 25$ 分钟，比顺序工作方式节省20分钟。

计算机中的流水线技术

- 处理机解释程序的方式有顺序方式、重叠方式、流水方式等。
 - **顺序方式**：解释完一条指令再开始解释下一条；
 - **重叠方式**：是一种简单的流水方式，它把指令分成2个或3个子过程，每条指令只与下一条或下两条指令相重叠。
 - **流水方式**：是把一个重复的过程分解为若干个子过程，每个子过程可以与其它子过程同时进行，以此提高单位时间内解释指令的数目；

指令的解释过程

Add ax,[bx]; ax←ax+[bx]



重叠

取指

分析

执行

流水

取指

译码

形成操作数地址

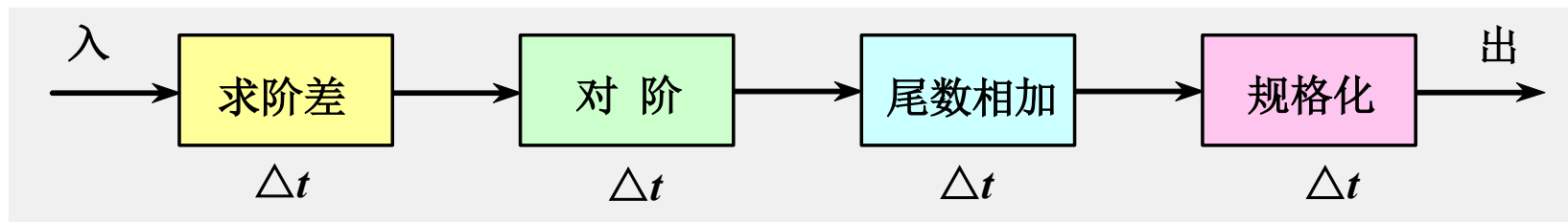
取操作数

执行

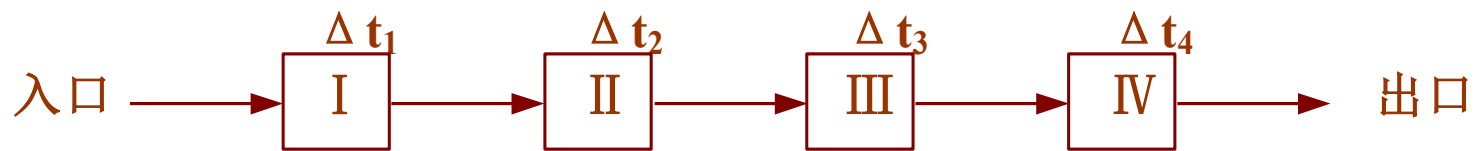
保存结果

例子：浮点加法流水线

- 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**



- **流水线技术**：将一条指令的执行分为几个阶段，让几条指令按流水线工作。
- 流水线的每一个阶段称为**流水步骤**、**功能段**、**流水节拍**等。
- 一个流水步骤与另一个流水步骤相连形成流水线。
- 指令从流水线一端进入，经过流水线的处理，从另一端流出。



流水线结构图

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，以保证提供后面要用到的信息，并把各段的处理工作相互隔离。

流水技术的特点

- 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。

指令的流水分析

■ 1、指令的顺序执行方式

- 一条指令的执行过程：取指令->分析->执行



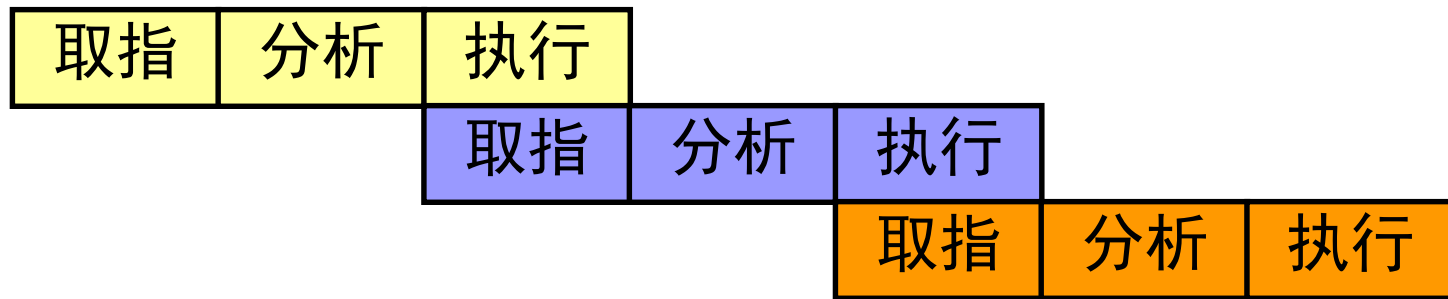
- 执行n条指令所用的时间为：

$$T = \sum_{i=1}^n (t_{\text{取指令}i} + t_{\text{分析}i} + t_{\text{执行}i})$$

- 如每段时间都为t，则执行n条指令所用的时间为：T=3nt
- 主要优点：控制简单，节省设备。
- 主要缺点：执行指令的速度慢，功能部件的利用率很低。

指令的流水（续）

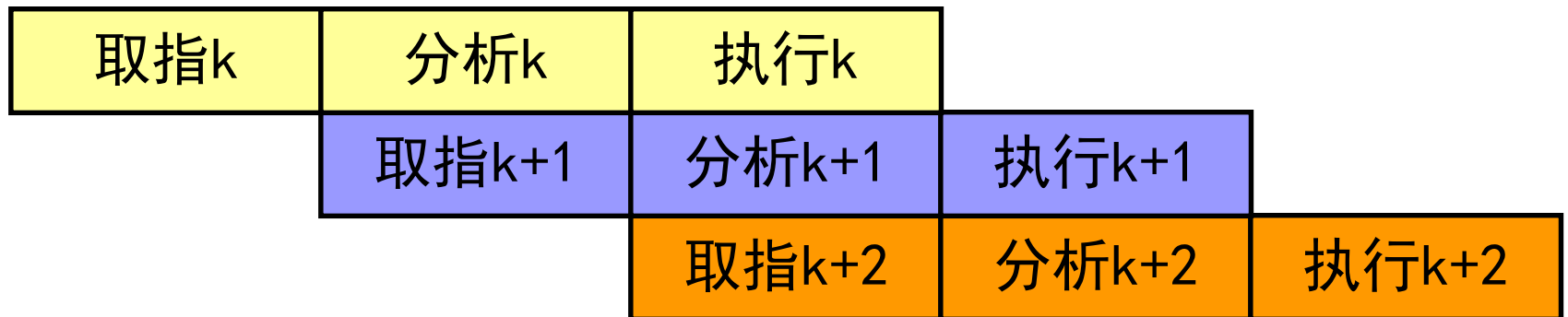
■ 2、一次重叠执行方式(一种最简单的流水线方式)



- 如果两个过程的时间相等，则执行n条指令的时间为：
 $T=(1+2n)t$
- 主要优点：
 - 指令的执行时间缩短
 - 功能部件的利用率明显提高
- 主要缺点：
 - 需要增加一些硬件
 - 控制过程稍复杂

指令的流水（续）

■ 3、二次重叠执行方式



- 如果三个过程的时间相等，执行n条指令的时间为：
 $T=(2+n)t$
- 理想情况下同时有三条指令在执行

3.1.2 流水线的分类

■ 按处理级别分类

- 部件级流水线
- 处理机级流水线
- 系统级流水线

■ 按功能分类

- 单功能流水线
- 多功能流水线。

■ 静态流水线

■ 动态流水线

■ 按连接方式分类

- 线性流水线
- 非线性流水线

■ 按处理对象分类

- 标量流水处理机
- 向量流水处理机

■ 按任务流出顺序分类

- 顺序流水线
- 乱序流水线。

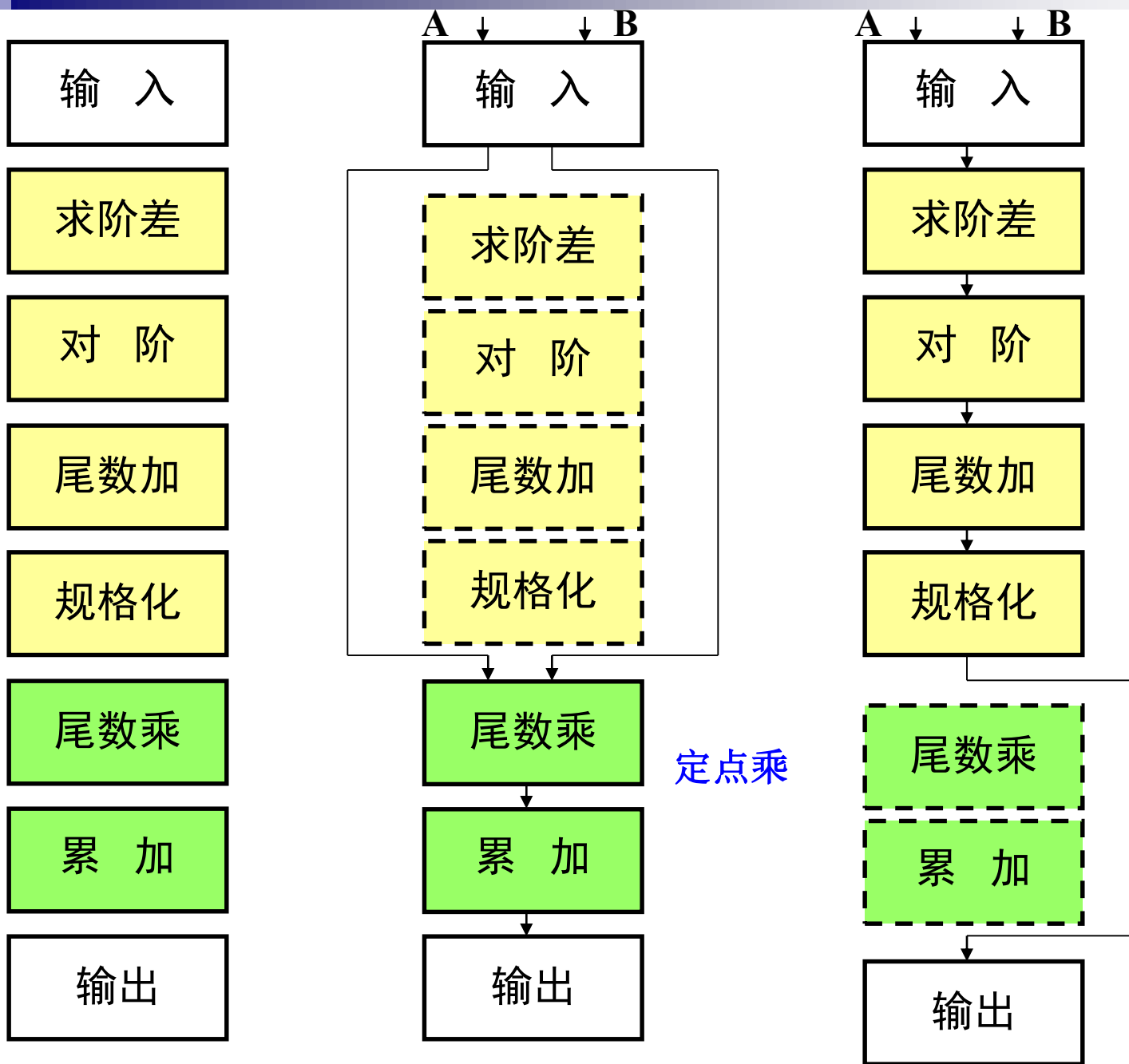
按处理级别分类

- **操作部件级流水线**（运算操作流水线）：指部件内部的各个子部件之间的流水，如运算器内浮点加法流水线以及**Cache**和多体交叉主存的流水。
 - **处理机级流水线**（运算操作流水线）：指构成处理机的各个部件之间的流水线。例如，由取指部件、指令分析部件和指令执行部件组成的指令流水线，就是典型的处理机级的流水线。
 - **系统级流水线**（宏流水线）：是指构成计算机系统的多个处理机之间的流水。
- (以上三个级别是从细到粗排列的)

按功能分类

- **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**：指流水线的各段可以实现不同的连接，在不同时间内或同一时间内，流水线能够通过不同的连接实现不同的处理功能。又可分为：
 - **静态流水线**：指在一段时间内，多功能流水线只能实现一种连接，从而只能实现一种功能，且只能等所有任务都流出，才可以重新连接实现另一种功能。如下页图中所示的ASC机运算器就是采用的静态流水线，不同的运算可以通过不同的连接实现。
 - **动态流水线**：指在同一时间段内，多功能流水线的各段可以实现多种连接，从而并发地实现多种功能。当然，一个功能段只能参加到某一种连接上。优点是灵活，能够提高流水线各段的使用率，从而提高处理速度。缺点是控制复杂。

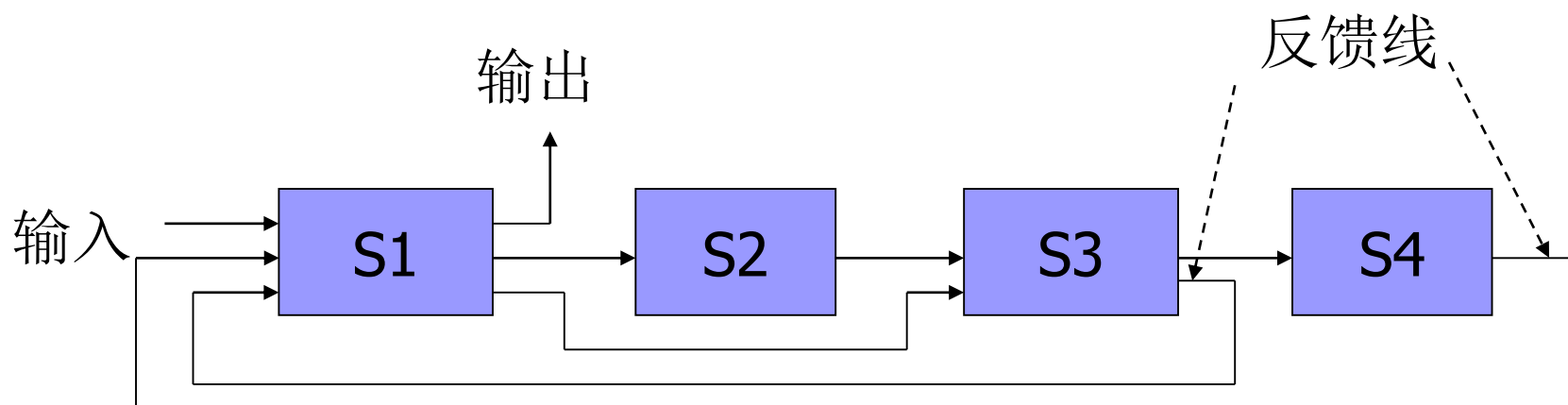
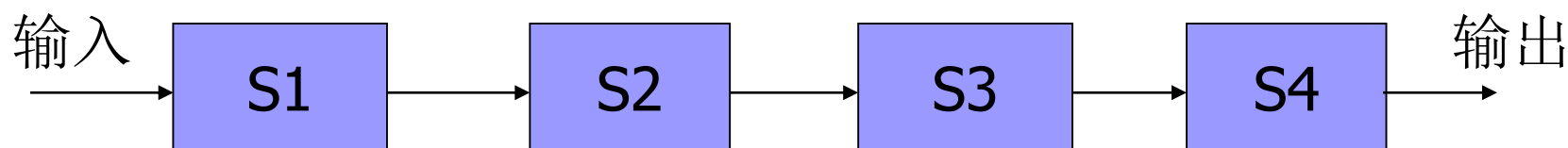
ASC机运算器的流水线



按连接方式分类

- **线性流水线**：流水线各个段间串行连接，各个任务顺序流经各段，没有反馈回路的流水线。对于一个任务而言，每个功能段只能经过一次。
- **非线性流水线**：流水线各个段间除了串行连接外，还有反馈回路，从而使被处理的任务要多次流过某些段。

线性流水线、非线性流水线举例



按处理对象分类

- **标量流水处理机**：只有标量表示和标量处理指令，没有向量表示和相应的向量指令，处理向量时，只能采用流水方式对向量的各个元素（都是标量）按标量指令的要求进行处理。
- **向量流水处理机**：有向量表示及向量指令流水线，一个向量指令序列可以在向量流水处理机上流水地执行。向量数据表示和流水技术的结合。

按任务流出顺序分类

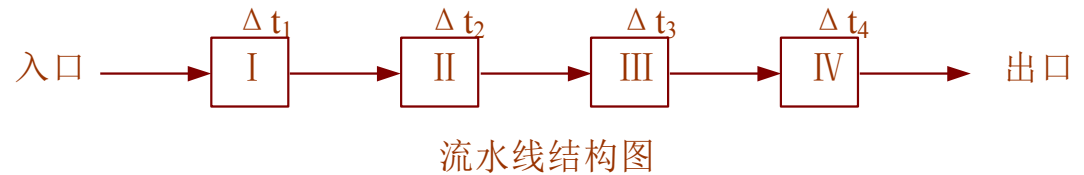
- **顺序流水线**：流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。每一个任务在流水线的各段中是一个跟着一个顺序流动的。
- **乱序流水线**：流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，允许后进入流水线的任务先完成（从输出端流出）。也称为无序流水线、错序流水线、异步流水线。

3.2 流水线的性能指标

- 流水线的表示方法
- 吞吐率 (**Through Put**)
- 加速比 (**Speedup**)
- 效率 (**Efficiency**)
- 流水线性能分析举例

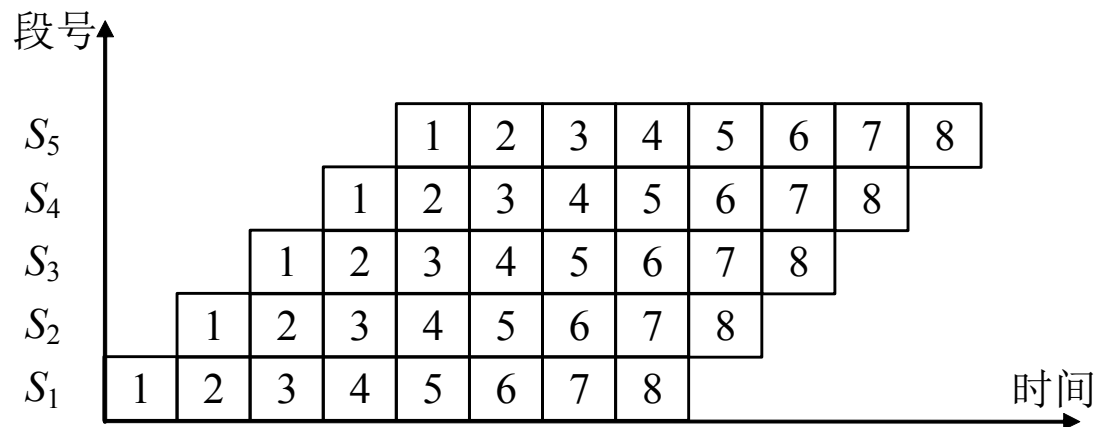
流水线的表示方法

■ 连接图



■ 时空图

- 即时间—空间图
- 横轴表时间
- 纵轴表流水段



流水线的时空图例子

1、吞吐率（Through Put）

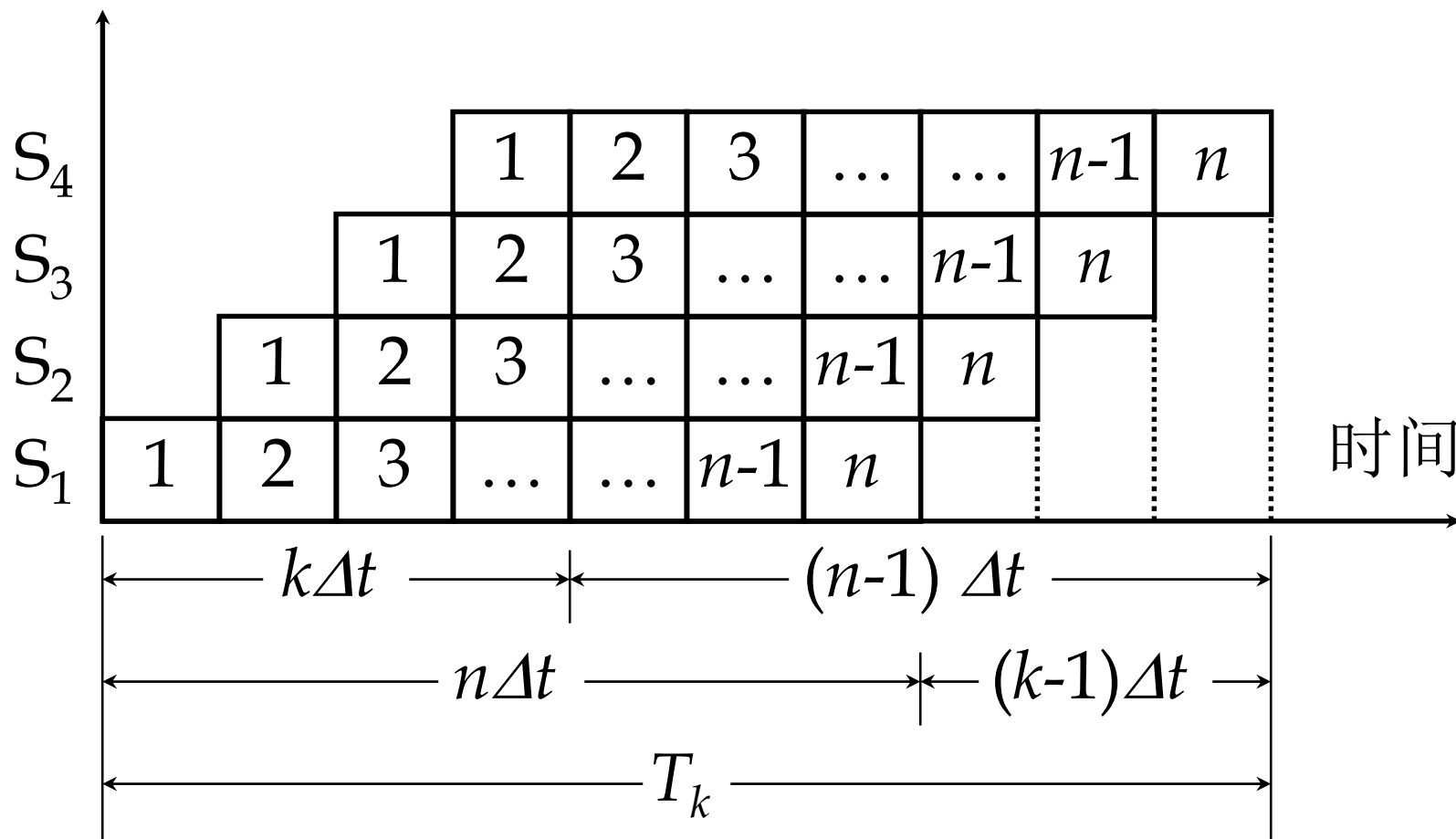
- **吞吐率（TP — ThroughPut）**指流水线在单位时间内执行的任务数（可以用输入任务数或输出任务数表示）。
- **最大吞吐率：**流水线达到不间断流水的稳定状态后可获得的吞吐率。通常，在任务连续不断进入时会达到最大吞吐率。
- **求流水线吞吐率的最基本公式：**

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

其中 T_k 为流水线执行 n 条指令所用时间
 k 为流水线的段数

各段等长、输入任务连续时

空间 ■ 流水线各段执行时间相等时的时空图



各段等长、输入任务连续时

- 在各段执行时间相等（均为 Δt ），且输入任务连续的情况下，完成 n 个连续任务需要的总时间为：

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

- 流水线的实际吞吐率为：

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

- 流水线的最大吞吐率为：

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

各段等长、输入任务连续时

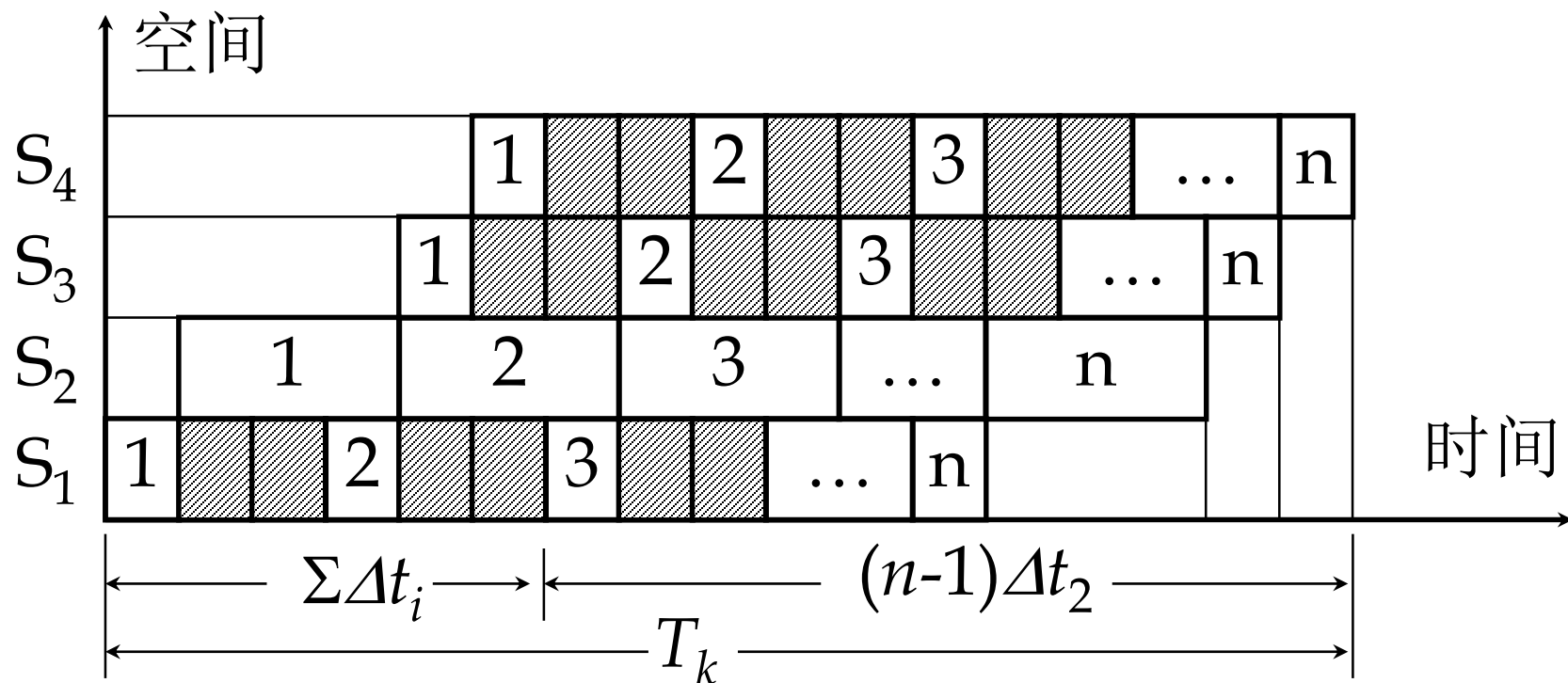
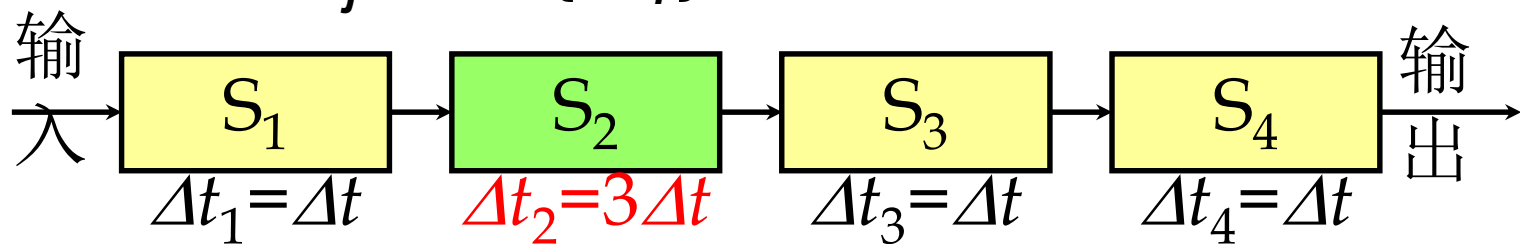
- 最大吞吐率与实际吞吐率的关系

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

- 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 $TP \approx TP_{\max}$ 。

各段不等长、输入任务连续时

- 流水线各段执行时间不相等时的时空图
- 隔 $\Delta t_j = \max\{\Delta t_i\}$ 输入一个任务



各段不等长、输入任务连续时

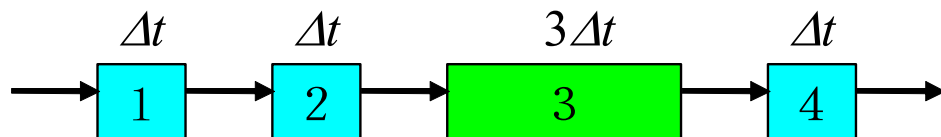
- 若有一条 k 段的线性流水线，段 i 的执行时间为 Δt_i ，其中， $\Delta t_j = \max\{\Delta t_i\}$ 为瓶颈段的执行时间，当 n 个任务连续进入流水线被处理完时，流水线的**实际吞吐率**为：

$$TP = \frac{n}{\sum_{i=1}^k \Delta t_i + (n-1)\Delta t_j}$$

- 当流水线的处理对象 $n \rightarrow \infty$ 时，可得到流水线的**最大吞吐率**

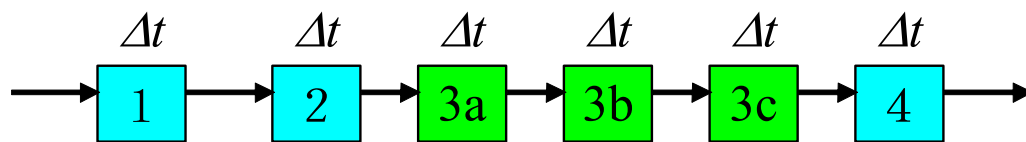
$$TP_{\max} = \frac{1}{\Delta t_j}$$

- 由前面的公式可知：流水线吞吐率取决于瓶颈段的执行时间。
- 消除瓶颈段来提高吞吐率的方法有以下两种：



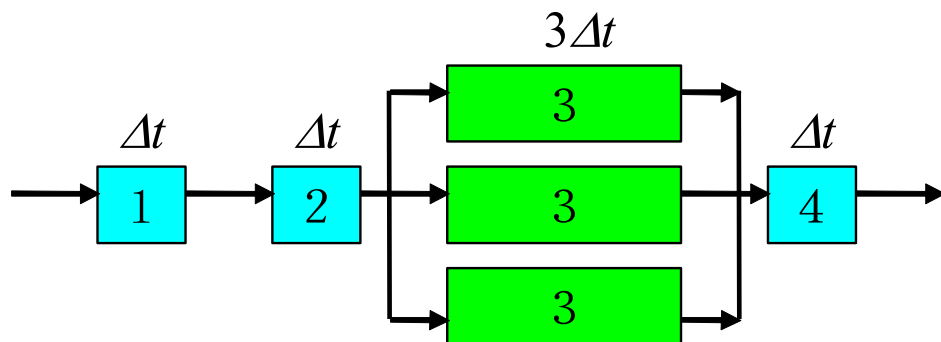
(a) 原流水线，第3段为瓶颈

改进后的流水线的吞吐率：



(b) 瓶颈段细分

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

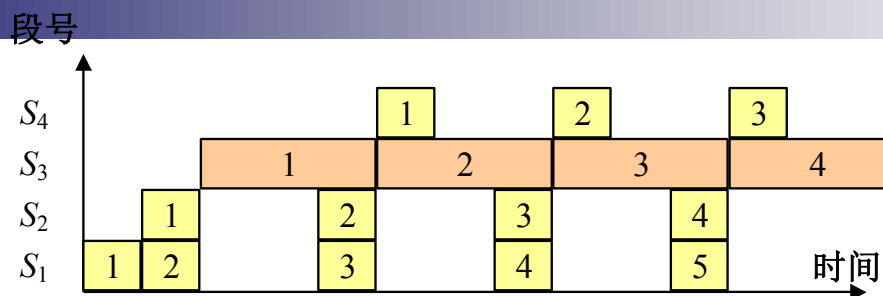


(c) 瓶颈段重复设置部件

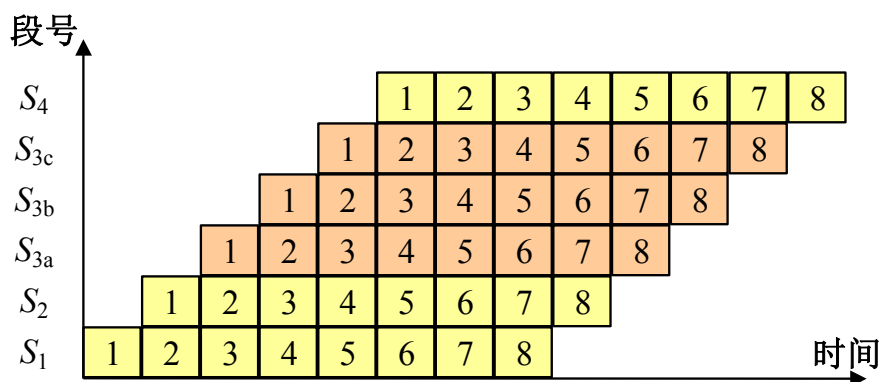
控制逻辑比较复杂，所需的硬件增加了。

图 消除流水线瓶颈段的两种方法

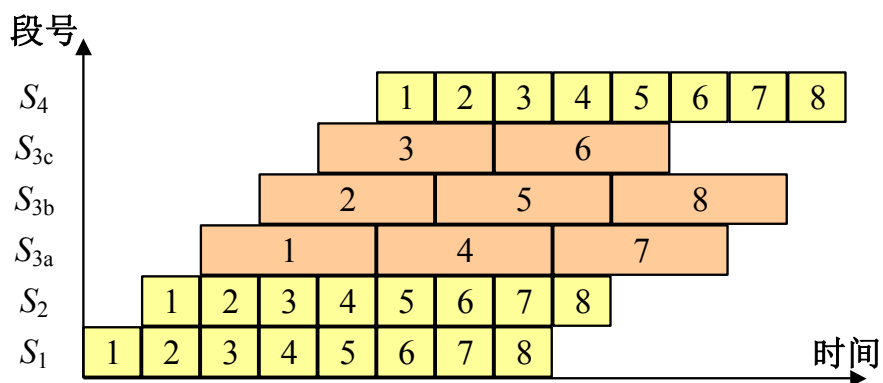
消除瓶颈段的两种方法 的时空图



(a) 原流水线时空图



(b) 流水段细分后的时空图



(c) 流水段重复的时空图

2、加速比 (Speedup)

- 是指一批任务采用顺序执行方式处理所需时间 T_o 与采用流水执行方式处理所需时间 T_k 的比值。

$$S = \frac{\text{顺序执行时间}}{\text{流水执行时间}} = \frac{T_o}{T_k}$$

- 各段执行时间不等，输入 n 个连续任务情况下加速比：

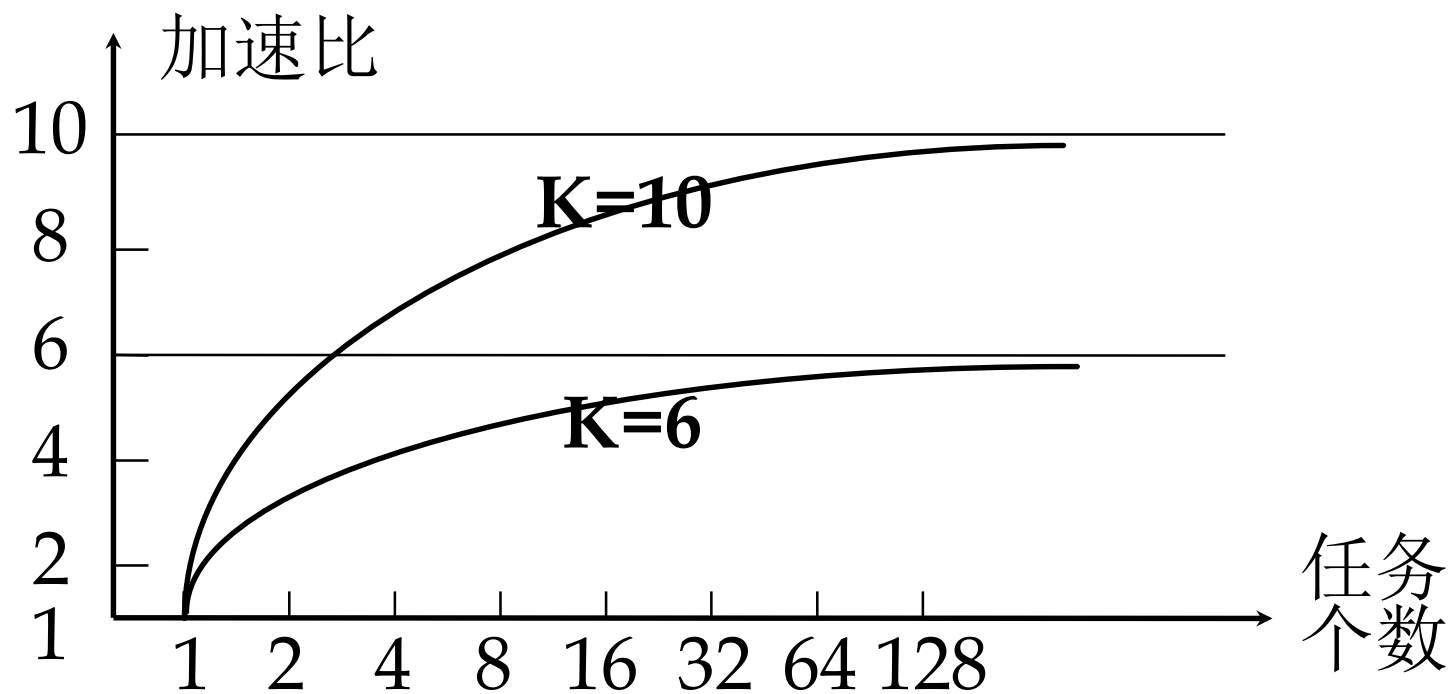
$$S = \frac{T_o}{T_k} = \frac{n \cdot \sum_{i=1}^k \Delta t_i}{\sum_{i=1}^k \Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$

- 各段执行时间相等，输入 n 个连续任务情况下加速比：

$$S = \frac{T_o}{T_k} = \frac{k \cdot n \cdot \Delta t}{(k+n-1)\Delta t} = \frac{k \cdot n}{k+n-1}$$

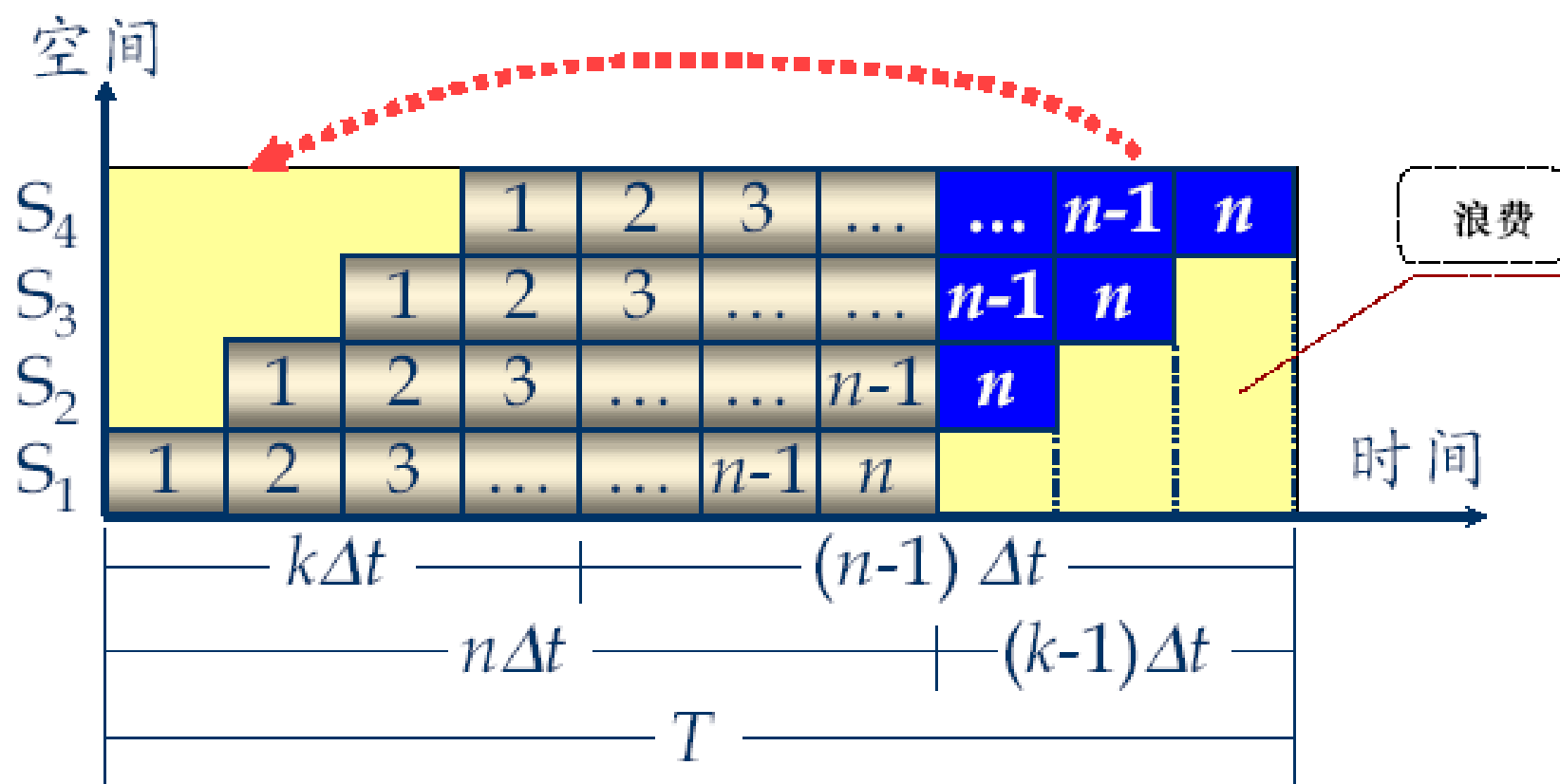
$$\text{最大加速比 } S_{\max} = \lim_{n \rightarrow \infty} \frac{k \cdot n}{k+n-1} = k$$

- 任务数 $\rightarrow\infty$ 时, 加速比 $\rightarrow K$



3、效率（Efficiency）

- 什么是效率？流水线设备的使用率
- 实际的硬件使用率 / 硬件提供的可能性



- **流水线效率**是指流水线设备的时间利用率，它是流水线各段设备有效工作时间之和与流水线被占用时间（从第一个对象流入至最后一个对象流出）的比值。

$$E = \frac{n \text{个任务占用的时空区}}{k \text{个流水段的总的时空区}} = \frac{T_0}{k \cdot T_k}$$

- 各段执行时间不等，输入 n 个连续任务情况下**流水线效率**：

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

- 各段执行时间相等，输入 n 个连续任务情况下**流水线效率**：

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

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

上述公式的使用条件

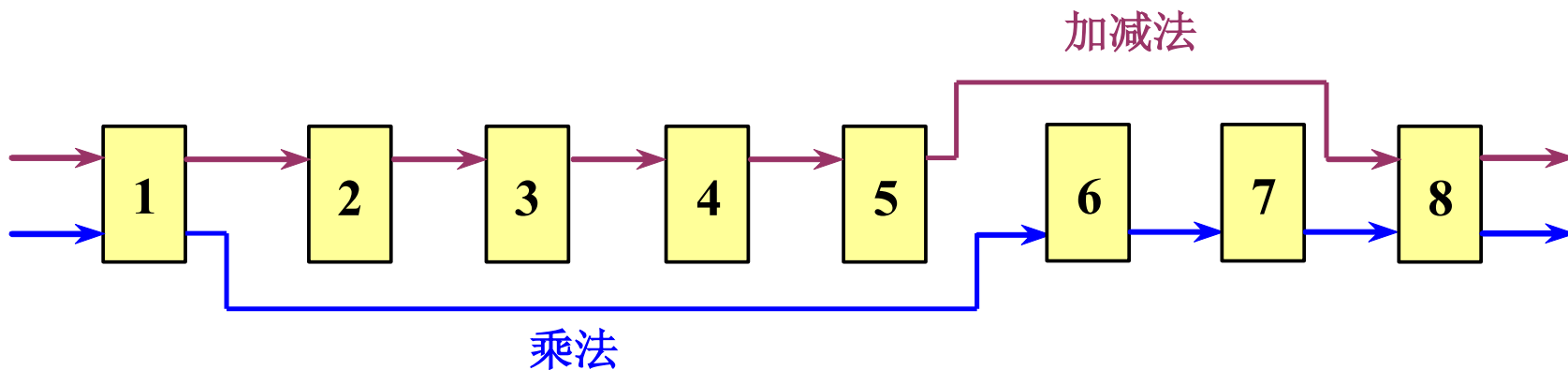
- 需要注意的是：应用上述公式计算流水线的实际吞吐率、加速比、效率的时候，必须注意公式的适用条件：即**流水线是线性的，且任务按流水线允许的时间间隔连续流入**。
- 如果流水线是非线性的，或者任务因各种原因未连续流入，那么，就不能够按照上述公式计算，在此情况下，可使用时空图方法进行计算。

流水线性能分析举例一

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)

解：（1）选择适合于流水线工作的算法

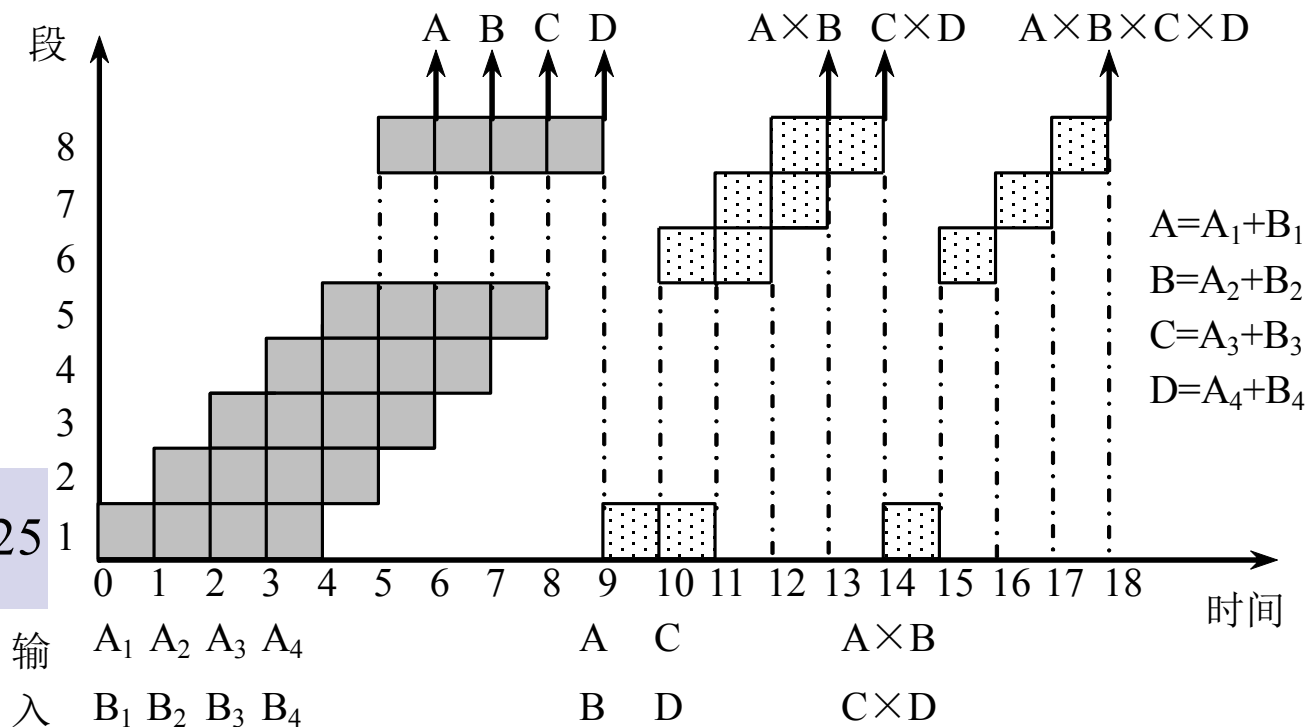
- 先计算 A_1+B_1 、 A_2+B_2 、 A_3+B_3 和 A_4+B_4 ；
- 再计算 $(A_1+B_1) \times (A_2+B_2)$ 和 $(A_3+B_3) \times (A_4+B_4)$ ；
- 然后求总的乘积结果。

（2）画出时空图

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

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

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



(3) 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：

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

流水线的效率为：

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

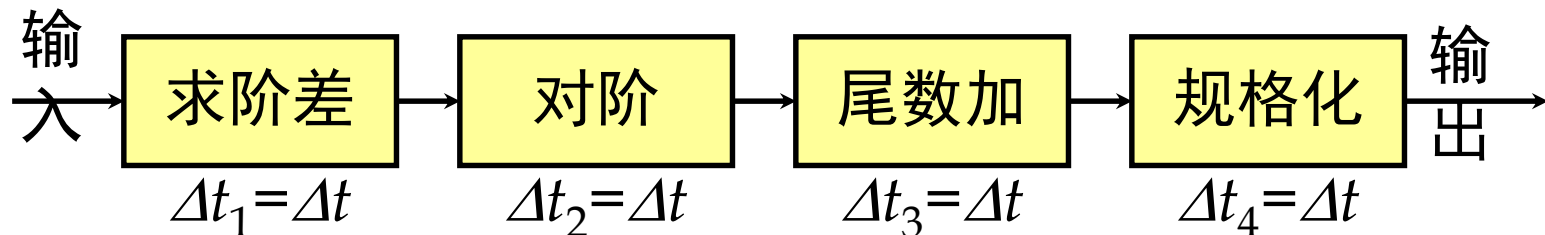
可以看出，在求解此问题时，该流水线的效率不高。原因在于：

- 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。

流水线性能分析举例二

- 单功能、线性流水线，输入任务不连续的情况，计算流水线的吞吐率、加速比和效率。
- 例：用图中所示的一条4段浮点加法流水线计算8个浮点数的和：

$$Z = A + B + C + D + E + F + G + H$$



解:

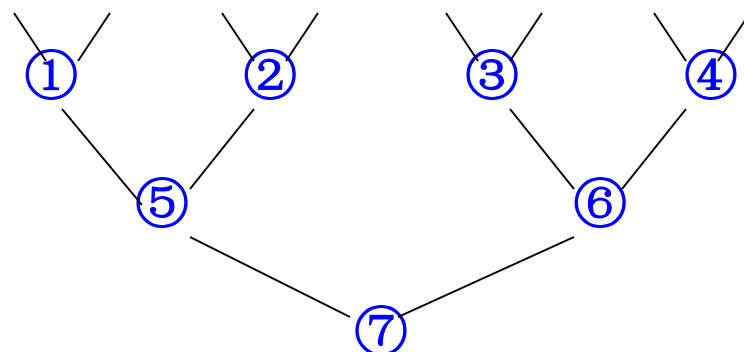
- 先将表达式转换为:

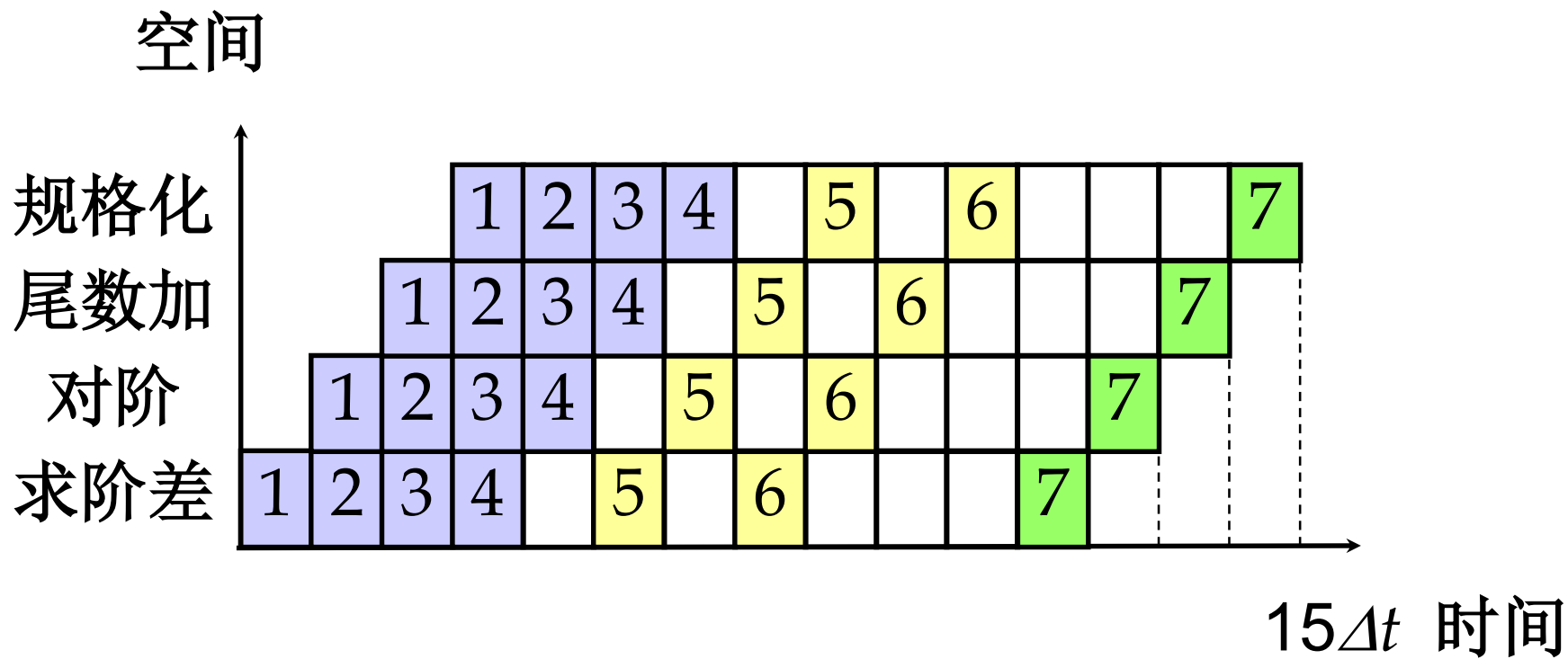
$Z = [(A+B) + (C+D)] + [(E+F) + (G+H)]$ 通过二叉数算法获得最大的并行度

可见, 共执行7次浮点加法, 这7次任务的次序为:

- 1、A+B
 - 2、C+D
 - 3、E+F
 - 4、G+H
 - 5、 $[(A+B) + (C+D)]$
 - 6、 $[(E+F) + (G+H)]$
 - 7、 $[(A+B) + (C+D)] + [(E+F) + (G+H)]$
- 指令间存在相关, 所以指令不能完全顺序流入流水线
 - 5必须在1,2后面执行
 - 6必须在3,4后面执行
 - 7必须在5,6后面执行
 - 流水线的时空图见下一页

$$Z = A + B + C + D + E + F + G + H$$





用一条4段浮点加法流水线求8个数之和的流水线时空图

共有**K=4**个流水段，**n= 7**次浮点加法运算共用了**15**个时钟周期。一次浮点加法运算要**4**个周期。当每个流水段的延迟都为 Δt 时：

- 流水线的吞吐率为：

$$TP = \frac{n}{T_k} = \frac{7}{15 \cdot \Delta t} = 0.47 \frac{1}{\Delta t}$$

- 流水线的加速比为：

$$S = \frac{T_0}{T_k} = \frac{4 \times 7 \cdot \Delta t}{15 \cdot \Delta t} = 1.87$$

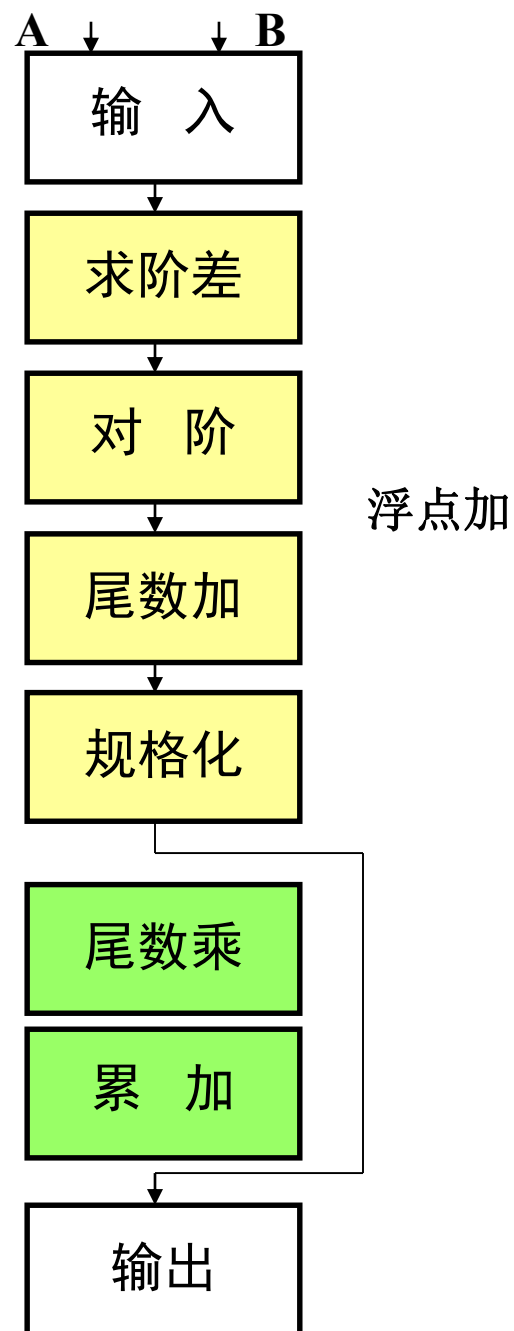
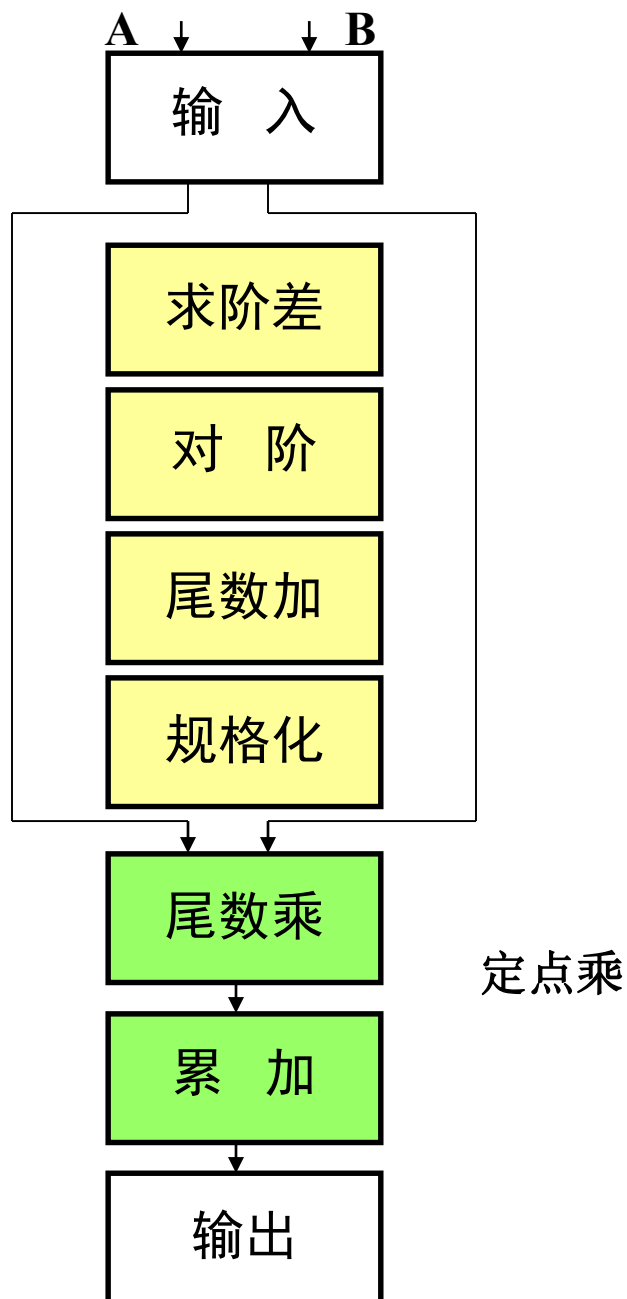
- 流水线的效率为：

$$E = \frac{T_0}{k \cdot T_k} = \frac{4 \times 7 \cdot \Delta t}{4 \times 15 \cdot \Delta t} = 0.47$$

流水线性能分析举例三

- 多功能、线性流水线，输入任务不连续的情况，计算流水线的吞吐率、加速比和效率。
- 例：用下图所示的**TI-ASC**计算机的多功能静态流水线计算两个向量的点积

$$\mathbf{Z} = \mathbf{A} \cdot \mathbf{B} + \mathbf{C} \cdot \mathbf{D} + \mathbf{E} \cdot \mathbf{F} + \mathbf{G} \cdot \mathbf{H}$$



解:

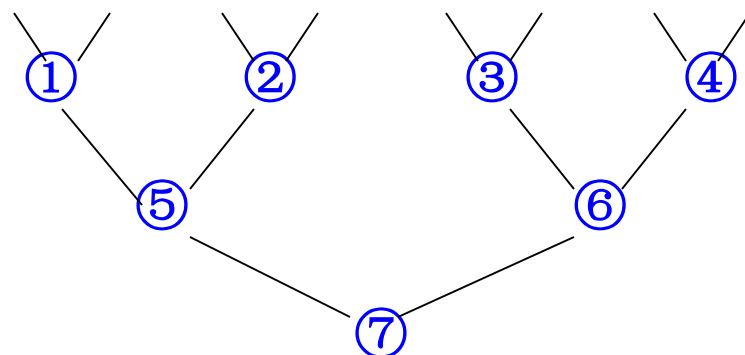
- 先将表达式转换为:

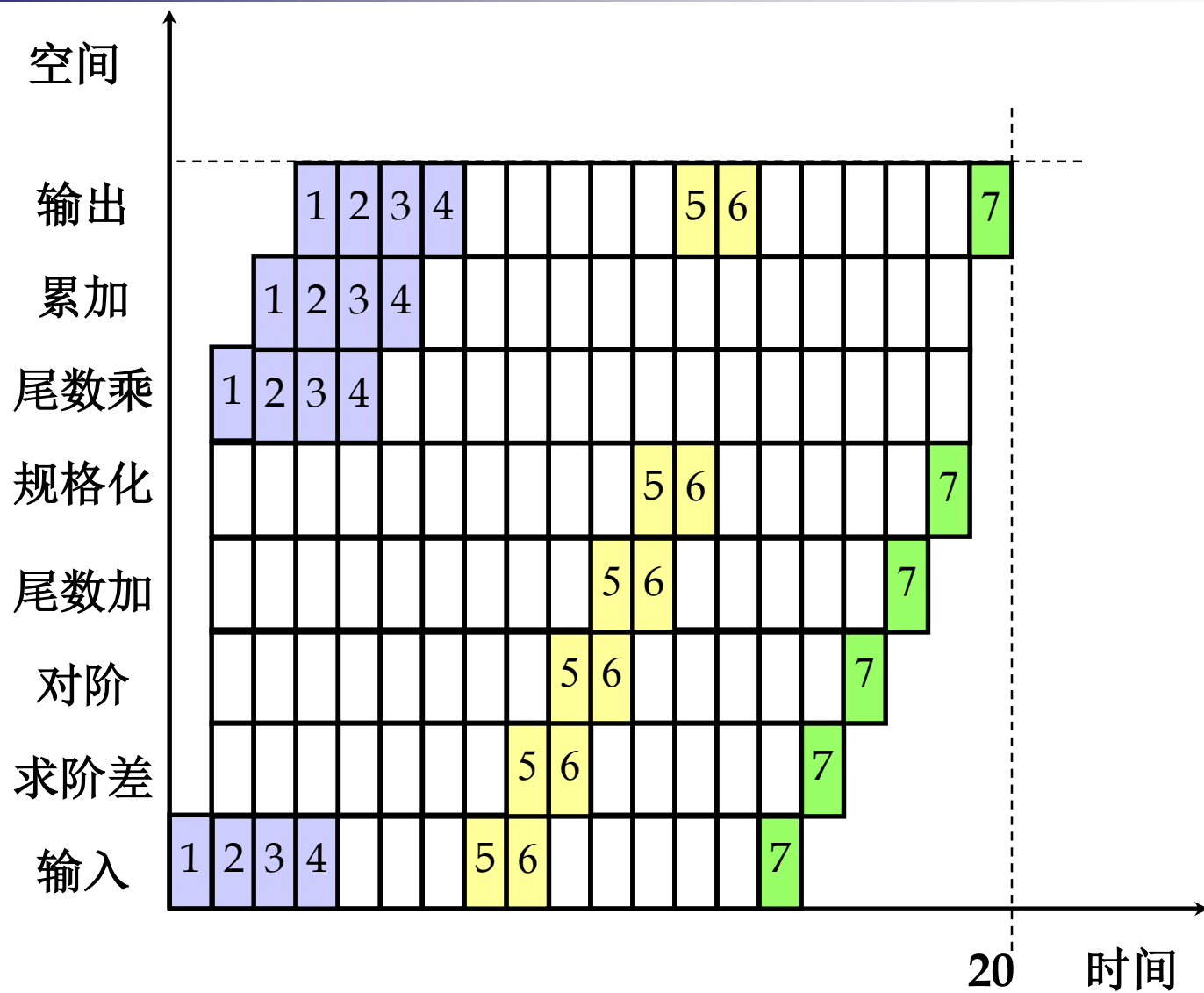
$$Z = [(A.B) + (C.D)] + [(E.F) + (G.H)]$$

通过二叉数算法获得最大的并行度
可见, 共执行4次乘法、3次浮点加法,
这7次任务的次序为:

- 1、A.B
 - 2、C.D
 - 3、E.F
 - 4、G.H
 - 5、[(A.B) + (C.D)]
 - 6、[(E.F) + (G.H)]
 - 7、[(A.B) + (C.D)] + [(E.F) + (G.H)]
- 指令间存在相关, 所以指令不能完全顺序流入流水线
 - 5必须在1,2后面执行
 - 6必须在3,4后面执行
 - 7必须在5,6后面执行
 - 流水线的时空图见下一页

$$Z = A \bullet B + C \bullet D + E \bullet F + G \bullet H$$





用TI-ASC多功能静态流水线求两个向量点积的流水线时空图

共有**K=8**个流水段，**n=7**个运算共用了**20**个时钟周期。一次加法运算要**6**个周期，一次乘法运算需**4**个周期。当每个流水段的延迟都为 Δt 时：

■ 流水线的吞吐率为：

$$TP = \frac{n}{T_k} = \frac{7}{20\Delta t} = 0.35 \frac{1}{\Delta t}$$

■ 流水线的加速比为：

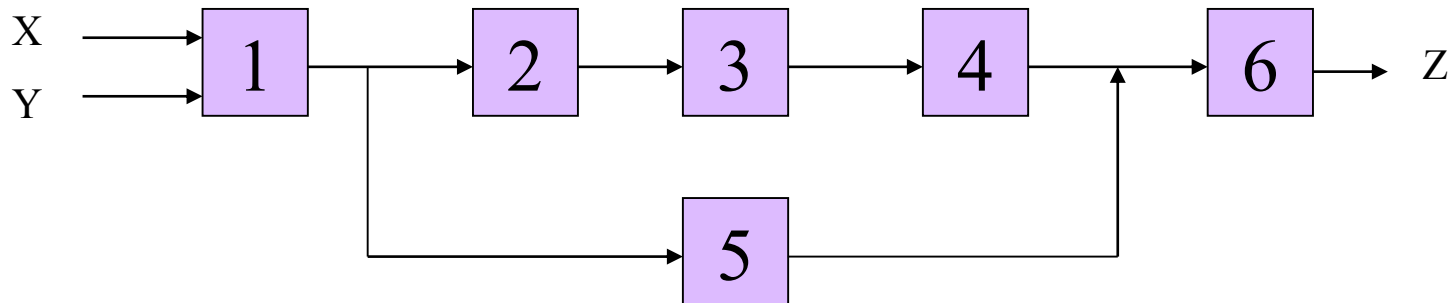
$$S = \frac{T_0}{T_k} = \frac{4 \times 4\Delta t + 3 \times 6\Delta t}{20\Delta t} = \frac{34\Delta t}{20\Delta t} = 1.70$$

■ 流水线的效率为：

$$E = \frac{T_0}{k \cdot T_k} = \frac{34\Delta t}{8 \times 20\Delta t} = 0.21$$

流水线性能分析举例四

- 一条线性静态多功能流水线由如图所示六个功能段组成, 加法操作使用其中的1, 5, 6功能段, 乘法操作使用其中的1, 2, 3, 4, 6功能段, 每个功能段的延迟时间均为 Δt , 流水线的输出端和输入端之间有直接数据通路, 而且设置有足够的缓冲寄存器。现在用这条流水线计算 $(a_1 + b_1) \times (a_2 + b_2) \times (a_3 + b_3) \times (a_4 + b_4)$, 画出流水线的时—空图, 并计算流水线的实际吞吐率, 加速比和效率。

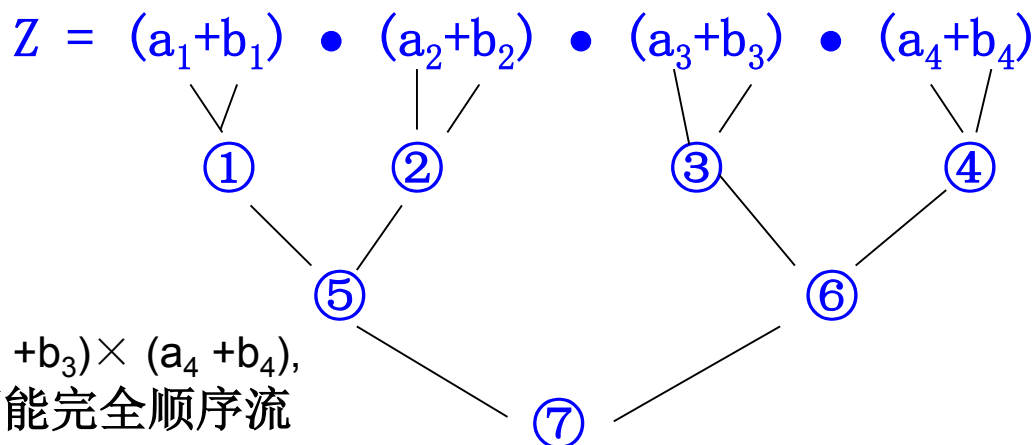


解:

- 先将表达式

$(a_1 + b_1) \times (a_2 + b_2) \times (a_3 + b_3) \times (a_4 + b_4)$, 进行任务分解, 共执行4次加法、3次乘法, 这7次任务的次序为:

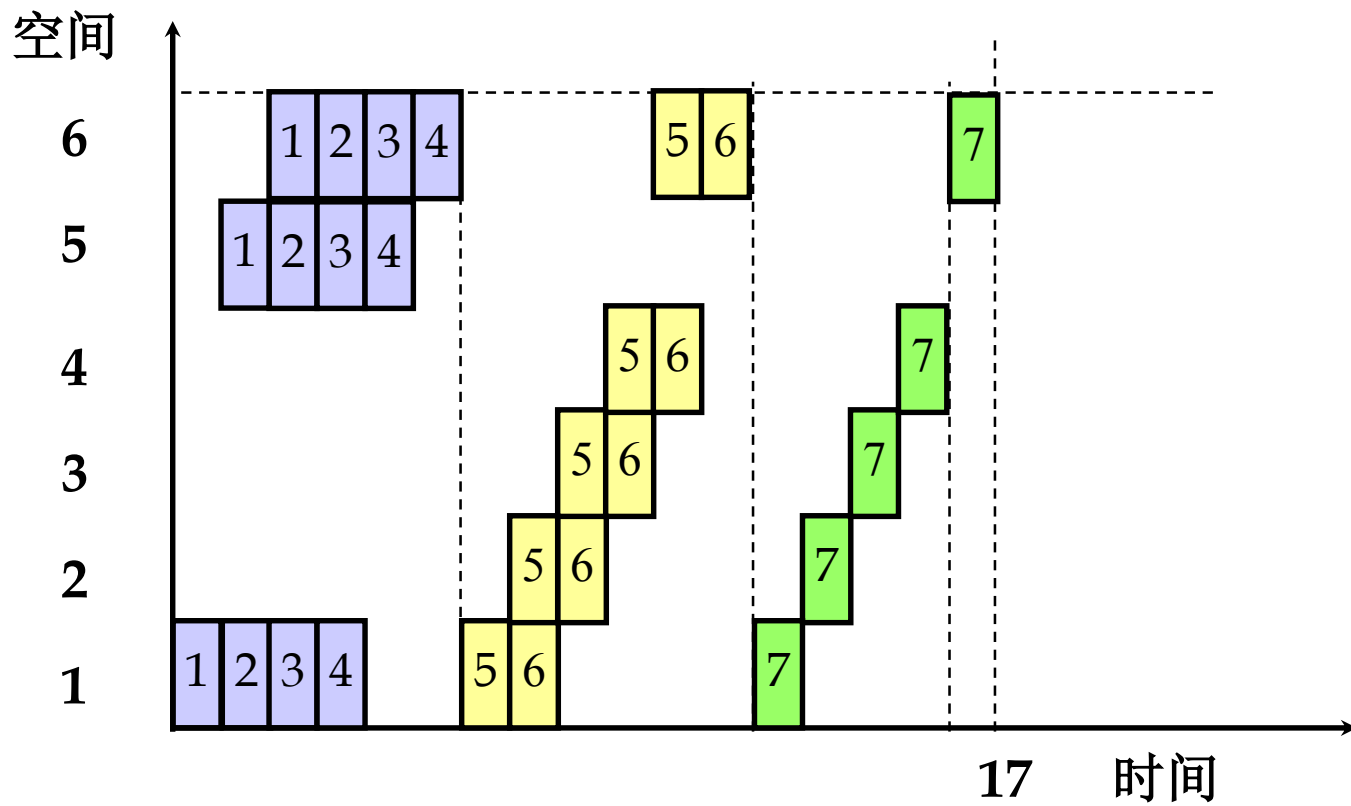
- 1、 $a_1 + b_1$
- 2、 $a_2 + b_2$
- 3、 $a_3 + b_3$
- 4、 $a_4 + b_4$
- 5、 $(a_1 + b_1) \times (a_2 + b_2)$
- 6、 $(a_3 + b_3) \times (a_4 + b_4)$
- 7、 $(a_1 + b_1) \times (a_2 + b_2) \times (a_3 + b_3) \times (a_4 + b_4)$,



- 指令间存在相关, 所以指令不能完全顺序流入流水线

- 5必须在1,2后面执行
- 6必须在3,4后面执行
- 7必须在5,6后面执行

- 流水线的时空图见下一页



$$\text{吞吐率 } TP = \frac{n}{T_k} = \frac{7}{17\Delta t}$$

$$\text{加速比 } S = \frac{T_0}{T_k} = \frac{4 \times 3\Delta t + 3 \times 5\Delta t}{17\Delta t} = \frac{27\Delta t}{17\Delta t} = 1.589$$

$$\text{效率 } E = \frac{T_0}{k \times T_k} = \frac{27\Delta t}{6 \times 17\Delta t} = 0.265 = 26.5\%$$

流水线设计中的若干问题

■ 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。

■ 流水线的额外开销

- 流水寄存器延迟
- 时钟偏移开销

流水线设计中的若干问题

- 流水寄存器需要建立时间和传输延迟
 - **建立时间**：在触发写操作的时钟信号到达之前，寄存器输入必须保持稳定的时间。
 - **传输延迟**：时钟信号到达后到寄存器输出可用的时间。
- 时钟偏移开销
 - 流水线中，时钟到达各流水寄存器的最大差值时间。（时钟到达各流水寄存器的时间不是完全相同）

流水线设计中的若干问题

■ 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。
- 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。

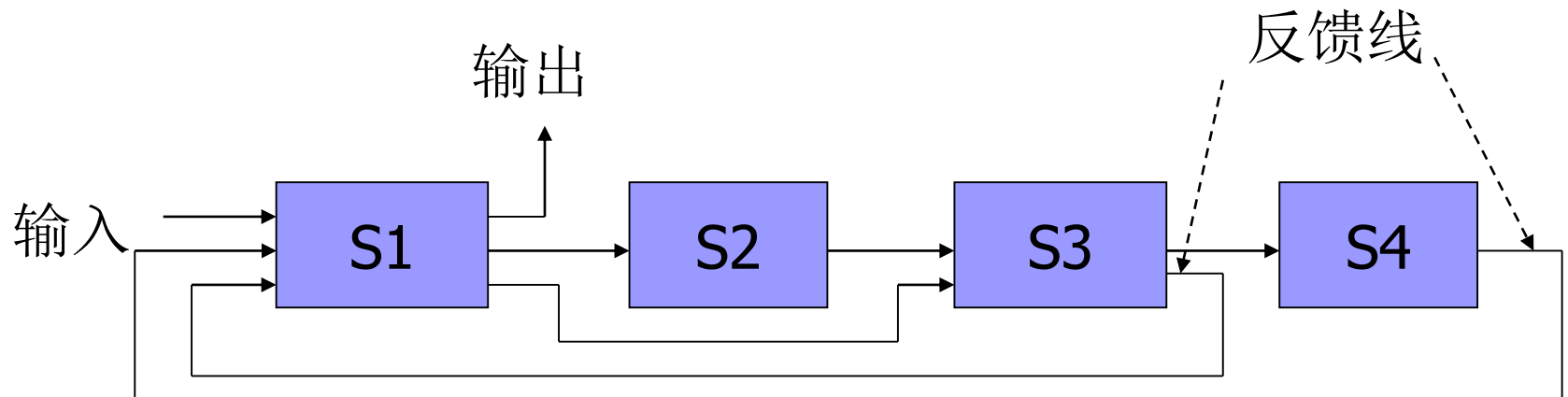
■ 冲突问题

流水线设计中要解决的**重要问题之一**。

3.3 非线性流水线的调度

- **非线性流水线**：流水线各个段间除了串行连接外，还有反馈回路，从而使处理的任务要多次流过某些段的流水线。
- **非线性流水线的调度要解决的问题**：确定任务流入流水线的時間间隔，使其既不发生任务争用流水段的冲突，又有较高的吞吐率和效率。

非线性流水线的连接图



单功能非线性流水线最优调度

- 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。

单功能非线性流水线最优调度算法

- 1)根据任务对流水线各段的使用时间建立一个**预约表**;
- 2)由预约表得出**禁止表**, 禁止表是禁止后续任务流入流水线的时间间隔的集合;
- 3)由禁止表得出**初始冲突向量**;
- 4)由初始冲突向量得出**状态有向图**;
- 5)由状态有向图得出**最优调度策略**;

预约表 (Reservation Table)

- **预约表**：表示某任务占用各流水段的预约情况。横坐标表示处理该任务所使用的流水线时钟周期，纵坐标表示流水线的各个流水段，中间有“✓”表示该流水段在这一个时钟周期处于工作状态，空白表示该流水段在这一个时钟周期不工作。

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
S_1	✓								✓
S_2		✓	✓					✓	
S_3				✓					
S_4					✓	✓			
S_5							✓	✓	

非线性流水线的预约表示例

由预约表得出禁止表F(Forbidden List)

- **禁止表F**：一个由禁用启动距离构成的集合。
- 将预约表的每一行中任意两个“√”之间的距离都计算出来，去掉重复的，这种数组成的一个数列就是这条非线性流水线的禁止向量。
- 例如：前述的非线性流水线，
 - 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；
- 其禁止向量， $F = \{1, 5, 6, 8\}$

由禁止表得出初始冲突向量 (Collision Vector)

- 初始冲突向量 $C_0 = (C_n C_{n-1} \dots C_k \dots C_2 C_1)$, 其中

$$C_k = \begin{cases} 1 & \text{禁止间隔 } k\Delta t \text{ 流入一个后续任务} \\ 0 & \text{允许间隔 } k\Delta t \text{ 流入一个后续任务} \end{cases}$$

Δt 为流水线各段的同步时钟周期。

冲突向量的位数为禁止表中的最大量

- 例如: 前述的禁止表为 $F = \{1, 5, 6, 8\}$
得到初始冲突向量 $C_0 = (10110001)$

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）。

- 对它们的冲突向量进行“或”运算。

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

其中： $\text{SHR}^{(j)}$ 表示逻辑右移j位

■ 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：

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

- 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。

- 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）

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：

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

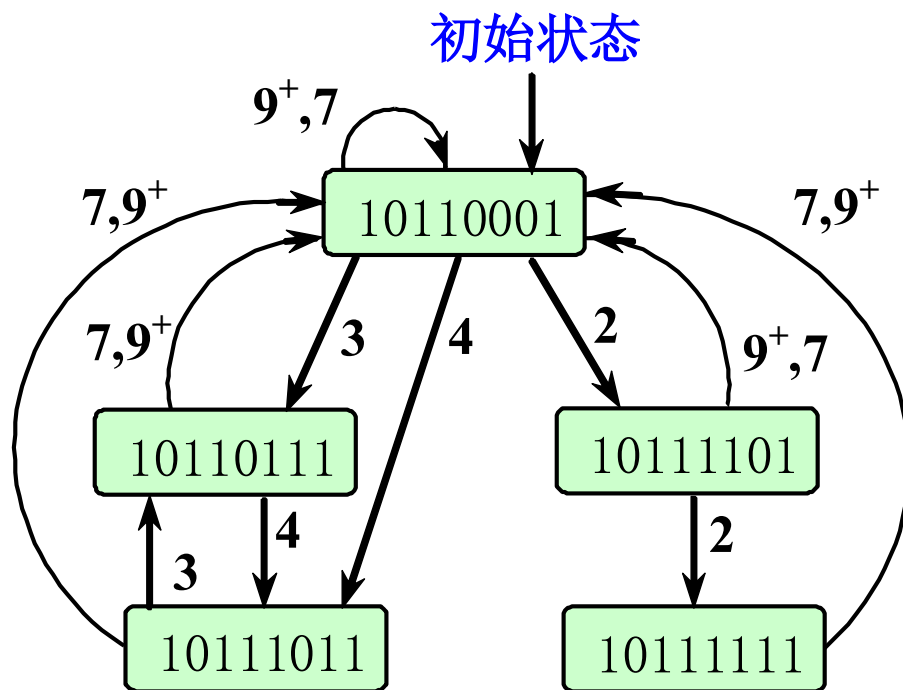
如果采用4，则新的冲突向量为：

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

如果采用7，则新的冲突向量为：

$$(00000001) \vee (10110001) = (10110001)$$

- (2) 对于新向量 (10111101)，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 (10111101) 和 (10110001)。
- (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。



由初始冲突向量得出状态有向图小结

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

C_0 10110001

$K=2,3,4,7$

00101100

10110001

10111101

$K=2$ 得 C_1

00010110

10110001

10110111

$K=3$ 得 C_2

00001011

10110001

10111011

$K=4$ 得 C_3

00000001

10110001

10110001

$K=7$ 得 C_0

C_1 10111101

$K=2,7$

00101111

10110001

10111111

$K=2$ 得 C_4

00000001

10110001

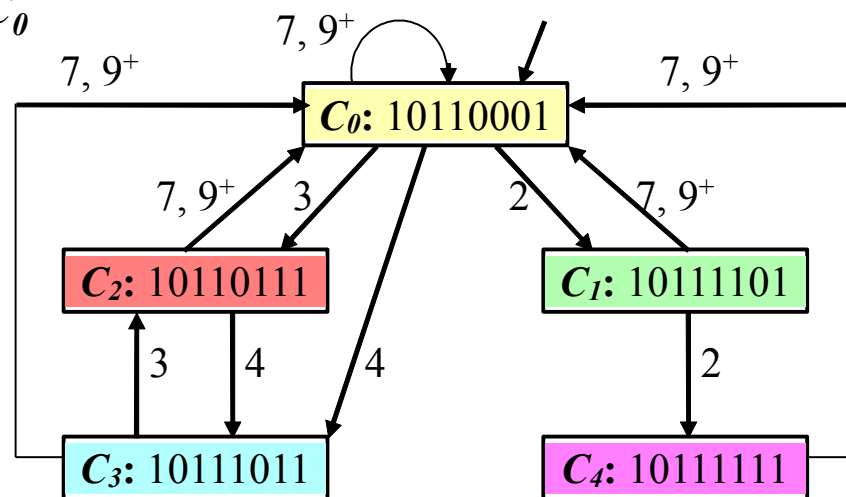
10110001

$K=7$ 得 C_0

↓

对 C_2 、 C_3 、 C_4 分别计算
发现不再产生新的状态向量。

从而得出状态向量图：



状态有向图

后续状态向量的计算公式小结

后续状态计算公式：

$$C_j = SHR^{(k)}(C_i) \vee C_0$$

当前状态 C_i

k 为 C_i 允许输入下任务的一个时间间隔

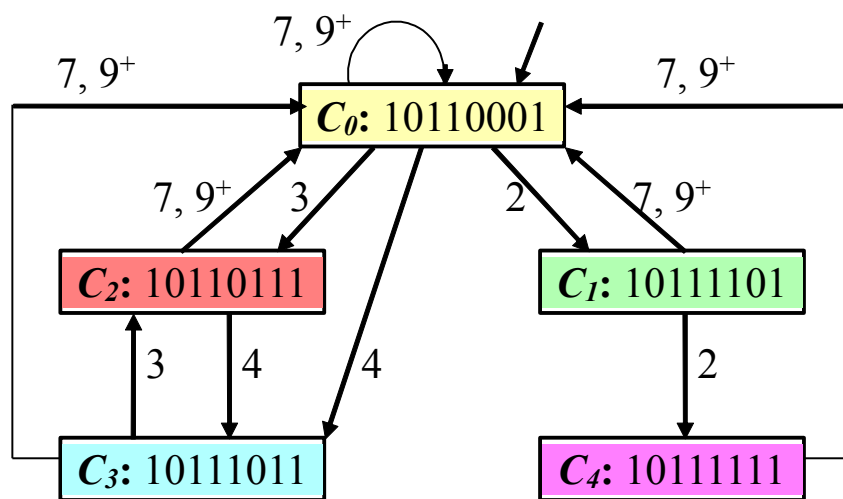
$SHR^{(k)}$ 表示右移 k 位，高位补0

后继状态 C_j 由向量 $SHR^{(k)}(C_i)$ 和初始冲突向量 C_0 “或”运算得出

循环计算，直到不再有新的状态向量产生为止

最后用有向线段连接各个状态，并标记值

由状态有向图得出各种调度方案，比较后，得出最优调度策略



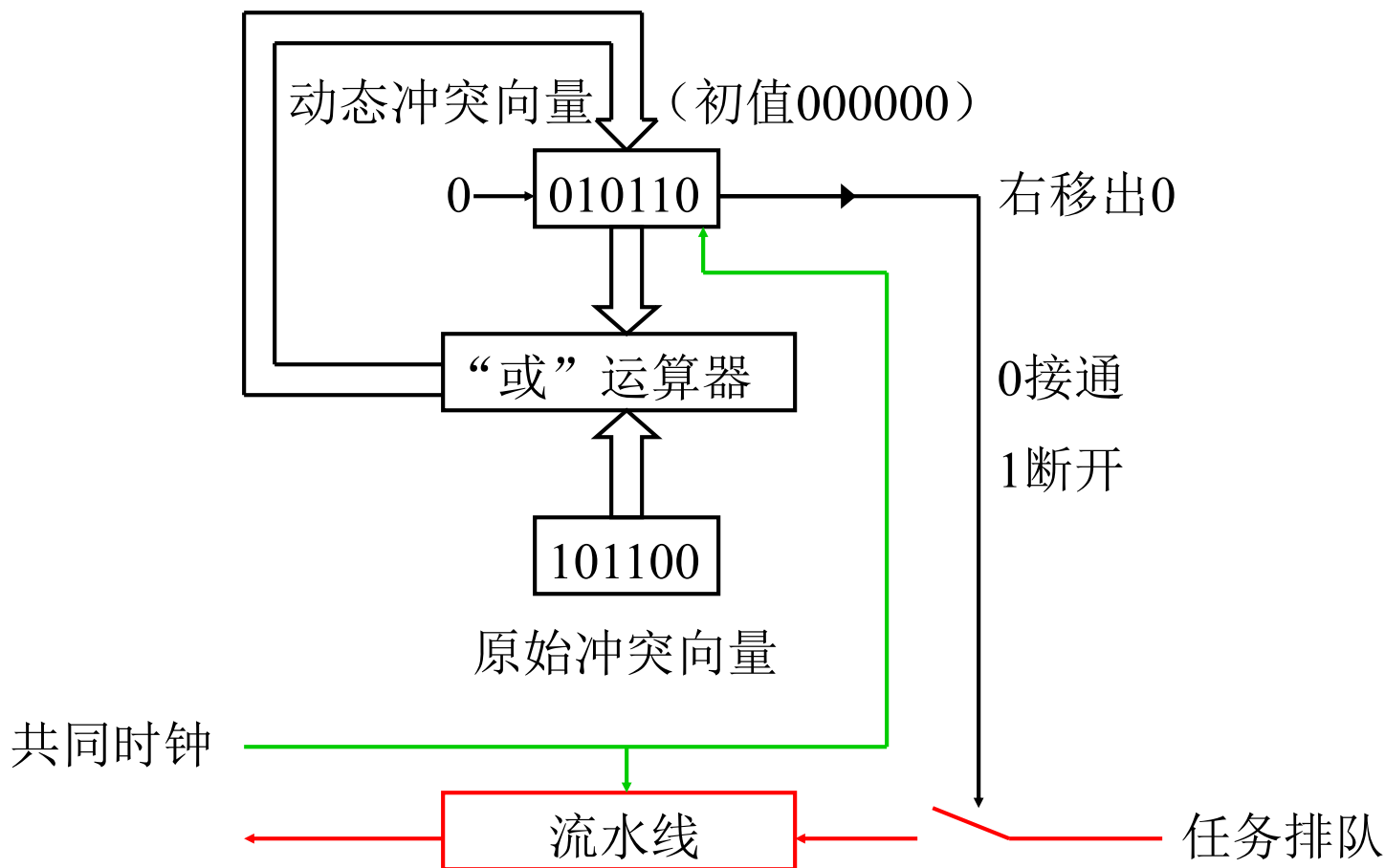
单功能非线性流水线状态有向图

调度策略	平均间隔周期
bc: (3,4)	3.50
ad: (2,7)	4.50
ade: (2,2,7)	3.67
abcb: (3,4,3,7)	4.25
abc: (3,4,7)	4.67
acb: (4,3,7)	4.67
ab: (3,7)	5.00
ac: (4,7)	5.50
a: (7)	7.00

- 找出所有的环路，既可得到调度方案
- 比较各个方案的平均时间间隔，最小的为最佳调度策略
- 本例中 bc 为最佳调度策略，平均间隔时间3.5个时钟周期（吞吐率最高）。注意：方案（4，3）不是最佳方案，why？

- 方案（3，4）是一种不等时间间隔的调度方案，与等间隔的调度方案相比，在控制上要复杂得多。为了简化控制，也可以采用等间隔时间的调度方案，但吞吐率和效率往往会下降不少。
- 在上述例子中，等时间间隔的方案只有一个：
（7），其吞吐率下降了一半。

非线性流水线调度的实现



3.3.2 多功能非线性流水线的调度

以双功能（功能A和B）非线性流水线为例。

■ 状态转移图中结点状态的表示

- 由两个冲突向量构成的冲突矩阵，这两个冲突向量分别对应于下一个任务的功能是A类和B类的情况。

■ 初始结点有两个，分别对应于第一个任务是A类和B类的情况。

- 当第一个任务是A类时，冲突矩阵为 $M^{(0)}_A$ 。
- 当第一个任务是B类时，冲突矩阵为 $M^{(0)}_B$ 。

3.3.2 多功能非线性流水线的调度

$$M_A^{(0)} = \begin{bmatrix} C_{AA} \\ C_{AB} \end{bmatrix} \quad M_B^{(0)} = \begin{bmatrix} C_{BA} \\ C_{BB} \end{bmatrix}$$

其中：

- C_{pq} ($p, q \in \{A, B\}$) 表示的是：在一个 p 类任务流入流水线后，对后续 q 类任务的冲突向量。

它们可以由预约表求得。

- C_{pq} 共有 $2^2=4$ 个。
- 对于 N 功能流水线，这种冲突向量有 N^2 个。

3.3.2 多功能非线性流水线的调度

3. 由下式求得后续状态的冲突矩阵

$$\text{SHR}^{(i)}(M_k) \vee M_r^{(0)}$$

其中：

- M_k ：当前状态
- r ：下一个流入任务的类型（A或B）
- i ：当前状态允许的流入 r 型任务的时间间隔
- $\text{SHR}^{(i)}(M_k)$ ：把当前状态中的各冲突向量逻辑右移 i 位

例如： $\text{SHR}^{(3)}(M_k) \vee M_A^{(0)}$ 表示的是：把当前状态 M_k 中的各冲突向量逻辑右移3位，再与初始矩阵 $M_A^{(0)}$ 进行“或”运算。

4. 举例说明双功能非线性流水线的最优调度

例题：有一条3段双功能非线性流水线，实现的功能是A和B，其预约表分别如表1和表2所示。各段的通过时间都是一个时钟周期。请找出该流水线单独处理A类任务和单独处理B类任务以及混合处理两类任务的最优调度方案。

表1 A类对象预约表

段 \ 时间	1	2	3	4	5
S1	√			√	
S2		√			
S3			√		√

表2 B类对象预约表

段 \ 时间	1	2	3	4	5
S1		√			√
S2				√	
S3	√		√		

解：（1）把两个预约表重叠起来，得到如表3所示的预约表。

段 \ 时间	1	2	3	4	5
S1	A	B		A	B
S2		A		B	
S3	B		AB		A

（2）由预约表求初始冲突向量和初始冲突矩阵

$$M_A^{(0)} = \begin{bmatrix} C_{AA} \\ C_{AB} \end{bmatrix}$$

$$M_B^{(0)} = \begin{bmatrix} C_{BA} \\ C_{BB} \end{bmatrix}$$

- C_{AA} : 一个A类任务流入流水线后, 对下一个A类任务进入流水线的时间间隔的限制。

根据预约表表1可知, 禁用时间间隔是2和3, 故禁止表为: $\{2, 3\}$, 所以 $C_{AA} = (0110)$ 。

- C_{BB} : 一个B类任务流入流水线后, 对下一个B类任务进入流水线的时间间隔的限制。

根据预约表表2可知, 禁用时间间隔是2和3, 所以 $C_{BB} = (0110)$ 。

- C_{AB} : 一个A类任务流入流水线后, 对下一个B类任务进入流水线的时间间隔的限制。根据预约表表3可知:

- 为了避免在S1发生冲突, 禁用时间间隔是: $4-2=2$

■ 在S2，不会发生冲突，这是因为根据表3，A类任务先于B类任务通过S2，而现在的实际情况又是A类任务先于B类任务流入流水线；

■ 为了避免在S3发生冲突，禁用时间间隔是：

$$3-1=2, 5-1=4, 5-3=2$$

■ 综合起来，有： $C_{AB} = (1010)$

■ C_{BA} 表示一个B类任务流入流水线后，对下一个A类任务进入流水线的的时间间隔的限制。根据预约表表3可知：

■ 为了避免在S1发生冲突，禁用时间间隔有：

$$2-1=1, 5-1=4, 5-4=1$$

■ 为了避免在S2发生冲突，禁用时间间隔是： $4-2=2$

- 在S3，不会发生冲突，这是因为根据表3，B类任务先于A类任务通过S3，或者同时通过S3（第3个时钟周期），而现在的实际情况又是B类任务先于A类任务流入流水线。
- 综合起来，有： $C_{AB} = (1011)$
初始矩阵为：

$$M_A^{(0)} = \begin{bmatrix} C_{AA} \\ C_{AB} \end{bmatrix} = \begin{bmatrix} 0110 \\ 1010 \end{bmatrix} \quad M_B^{(0)} = \begin{bmatrix} C_{BA} \\ C_{BB} \end{bmatrix} = \begin{bmatrix} 1011 \\ 0110 \end{bmatrix}$$

(3) 由初始冲突矩阵画出状态图

如果第一个流入的任务是A类，初始状态就是 $M_A^{(0)}$ ；如果第一个流入的任务是B类，则初始状态是 $M_B^{(0)}$ 。

求所有后续状态的冲突矩阵：

例如：在流入一个A类任务后，从 $C_{AA} = (0110)$ 可知，可以隔一个或4个时钟周期再流入一个A类任务。

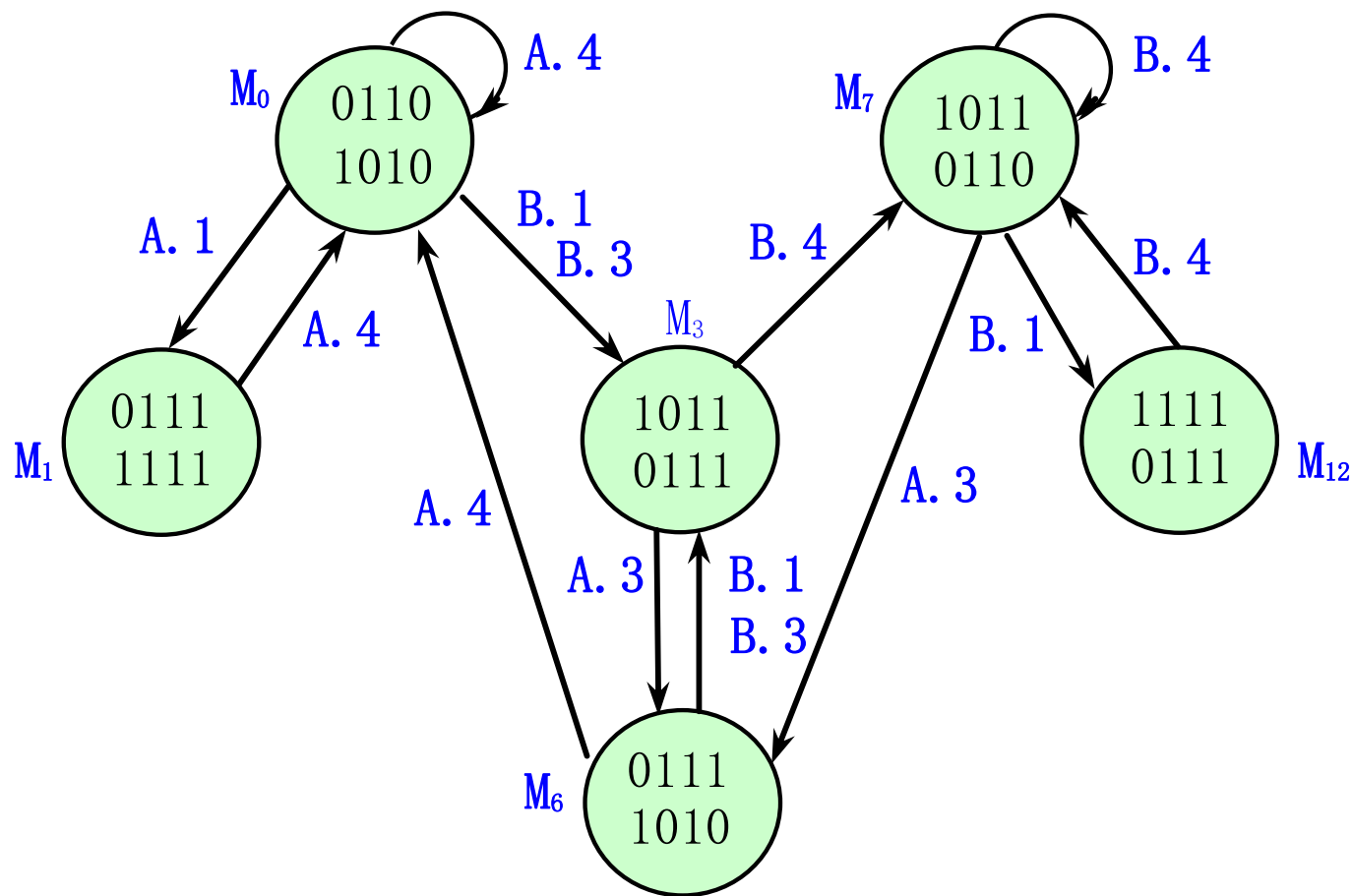
假设是前者，则把初始状态 $M_A^{(0)}$ 中的各冲突向量同时右移一位（即进行SHR⁽¹⁾操作），再与 $M_A^{(0)}$ 进行或运算，可以得到新的冲突矩阵。根据该矩阵的第一个冲突向量 (0111) 可知，只有隔4个时钟周期流入一个A类任务，才能不发生冲突。把冲突矩阵中的各冲突向量同时右移4位（即进行SHR⁽⁴⁾操作），再与 $M_A^{(0)}$ 进行按位或运算，可以得到新的冲突矩阵。

再如，在流入一个A类任务后，从 $C_{AB} = (1010)$ 可知，可以隔一个或3个时钟周期再流入一个B类任务。假设是前者，则把初始状态 $M_A^{(0)}$ 中的各冲突向量同时右移一位（即进行SHR⁽¹⁾操作），再与 $M_B^{(0)}$ 进行或运算，可以得到新的冲突矩阵。

据此可知，允许的流入为：或者是隔3个时钟周期流入一个A类任务，或者是隔4个时钟周期流入一个B类任务。按类似与上述类似的方法，又可以得到新的冲突矩阵。

求出所有可能的状态后，就可以画出状态图。

弧线上的标记“ $r.i$ ”表示：隔 i 个时钟周期流入 r 类的任务。



双功能非线性流水线的状态图

(4) 由状态图得出最优调度方案

从状态图可以找出各种情况下的最优调度方案：

- ◆ 只流入A类任务的最优调度方案是： (A.1, A.4)
- ◆ 流入B类任务的最优调度方案是： (B.1, B.4)
- ◆ 混合流入A、B两类任务的最优调度方案是：
(B.1, A.3, A.4)

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
- 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。

■ 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。

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。

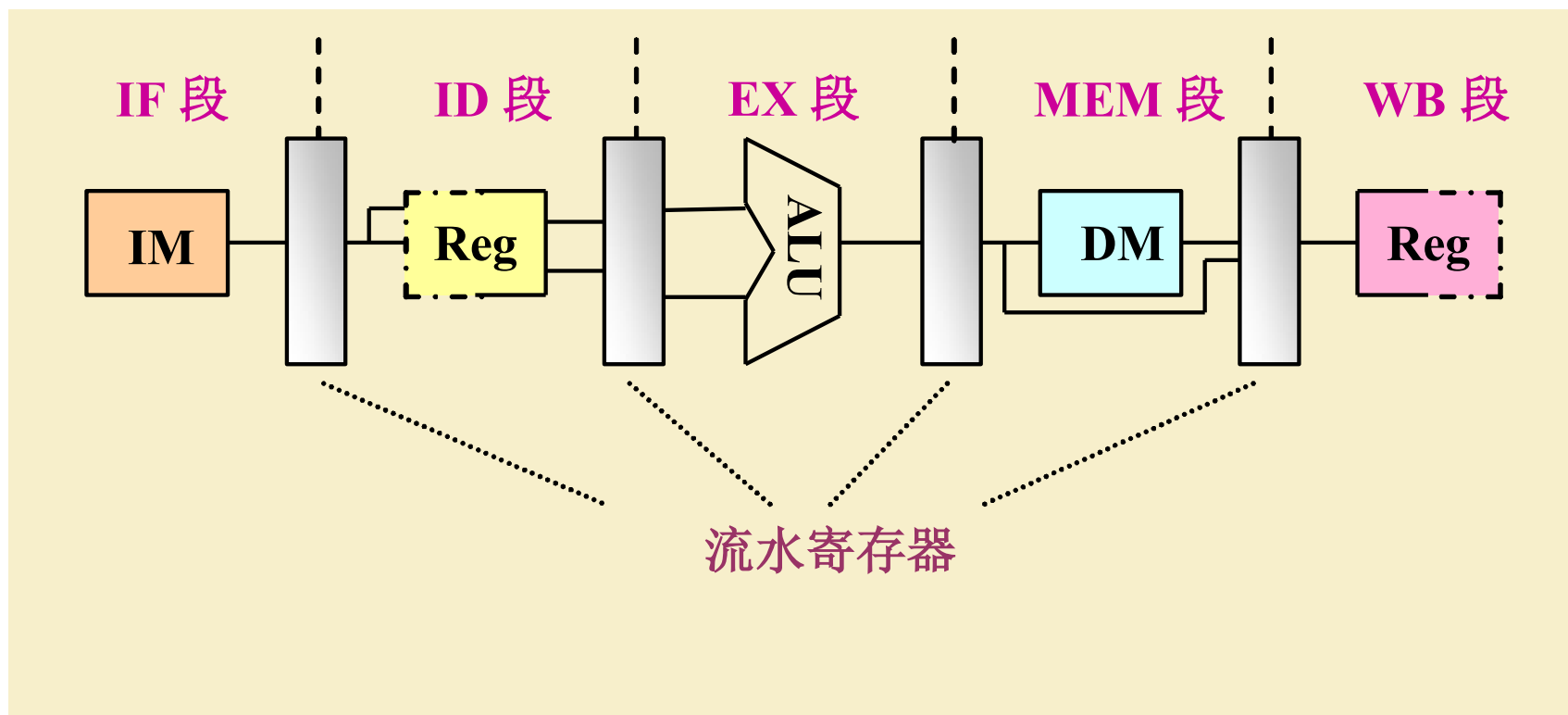
load指令：结果数据来自存储器。

在这个实现方案中：

- ▣ 分支指令需要4个时钟周期（如果把分支指令的执行提前到ID周期，则只需要2个周期）；
- ▣ store指令需要4个周期；
- ▣ 其它指令需要5个周期才能完成。

2. 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. 采用流水线方式实现时，应解决的问题：

- 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。

ID段：读

WB段：写

如何解决对同一寄存器的访问冲突？

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。

请考虑一下，如何处理分支指令？

3.4 流水线的相关与冲突

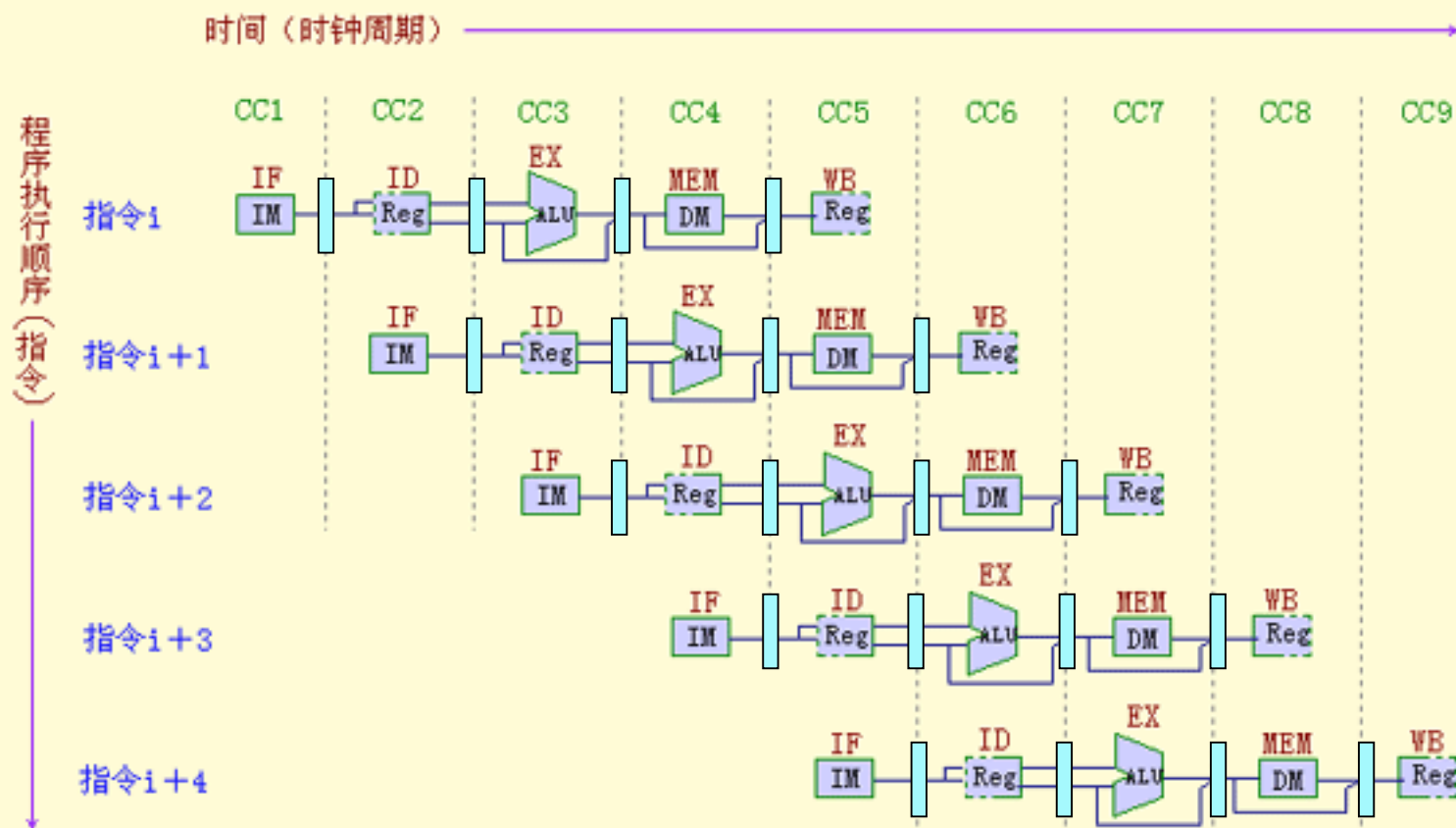
4. 5段流水线的两种描述方式

■ 第一种描述（类似于时空图）

指令编号	时钟周期								
	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）

流水线可以看成是按时间错开的数据通路序列



3.4 流水线的相关与冲突

3.4.2 相关与流水线冲突

3.4.2.1 相关dependence

- **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）
- 名字相关
- 控制相关

3.4 流水线的相关与冲突

■ 数据相关(Data Dependence)

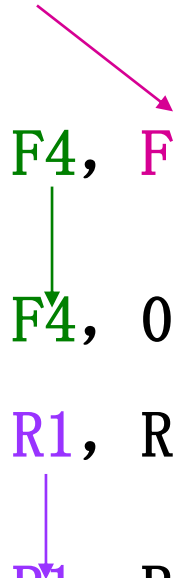
- 对于两条指令*i*（在前，下同）和*j*（在后，下同），如果下述条件之一成立，则称指令*j*与指令*i*数据相关。
 - 指令*j*使用指令*i*产生的结果；
 - 指令*j*与指令*k*数据相关，而指令*k*又与指令*i*数据相关。
- 数据相关具有传递性。

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

例如：下面这一段代码存在数据相关。

```
Loop:  L. D      F0, 0 (R1)      // F0为数组元素
        ADD. D   F4, F0, F2      // 加上F2中的值
        S. D     F4, 0 (R1)      // 保存结果
        DADDIU   R1, R1, -8      // 数组指针递减8个字节
        BNE      R1, R2, Loop    // 如果R1≠R2, 则分支
```



3.4 流水线的相关与冲突

- 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。

2. 名相关(Name Dependence)

- **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**。

3.4 流水线的相关与冲突

- 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。

指令j写的名=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。

指令j写的名=指令i写的名

3.4 流水线的相关与冲突

- 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：

DIV. D F2, F8, F4

ADD. D F8, F0, F12

SUB. D F10, F8, F14

DIV. D和ADD. D存在反相关。

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。

3.4 流水线的相关与冲突

3.4.2.2 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

带来的几个问题：

- 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

■ 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

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。

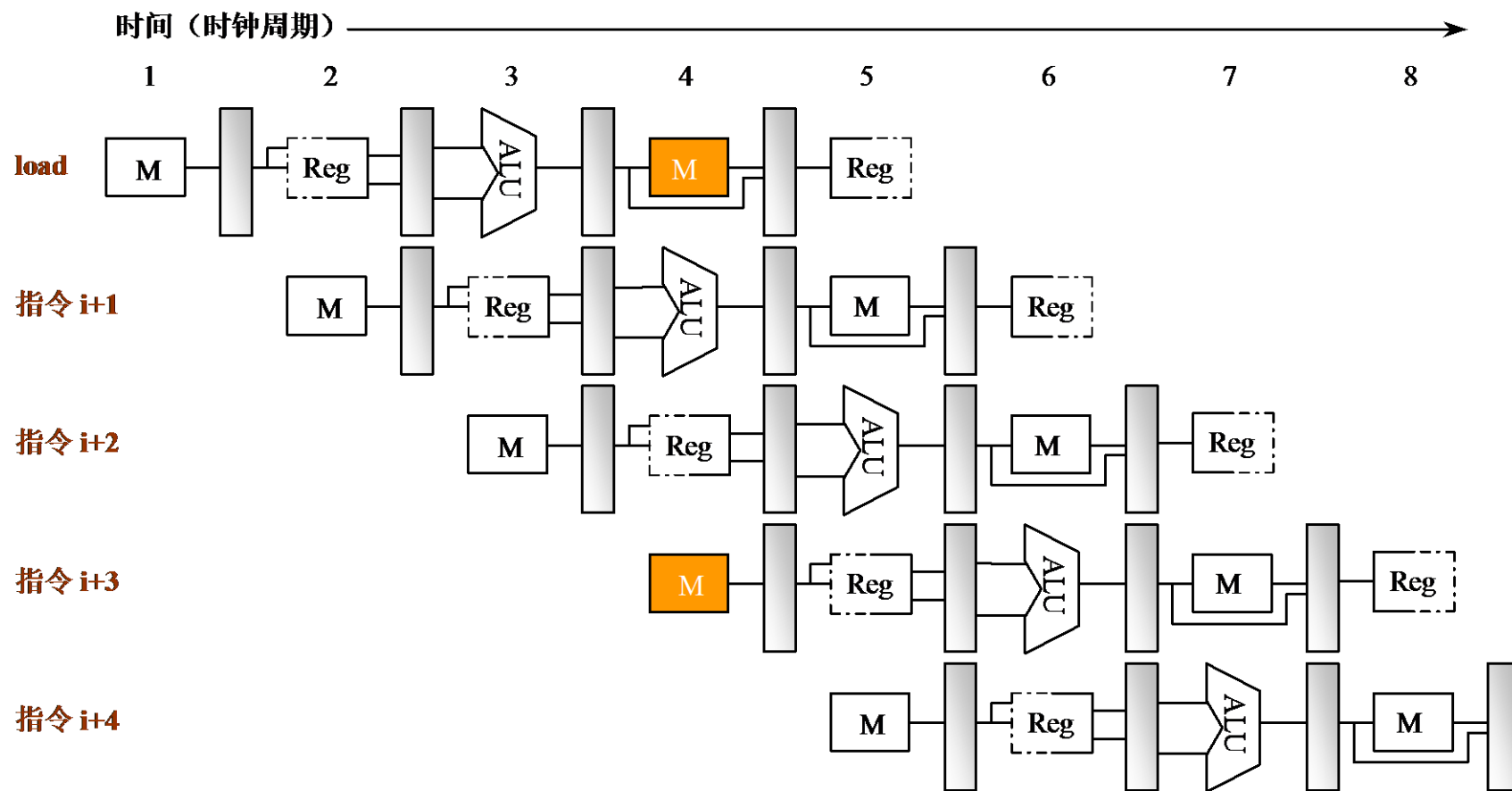
- 解决办法Ⅰ：插入暂停周期

（“流水线气泡”或“气泡”）

- 解决方法Ⅱ：

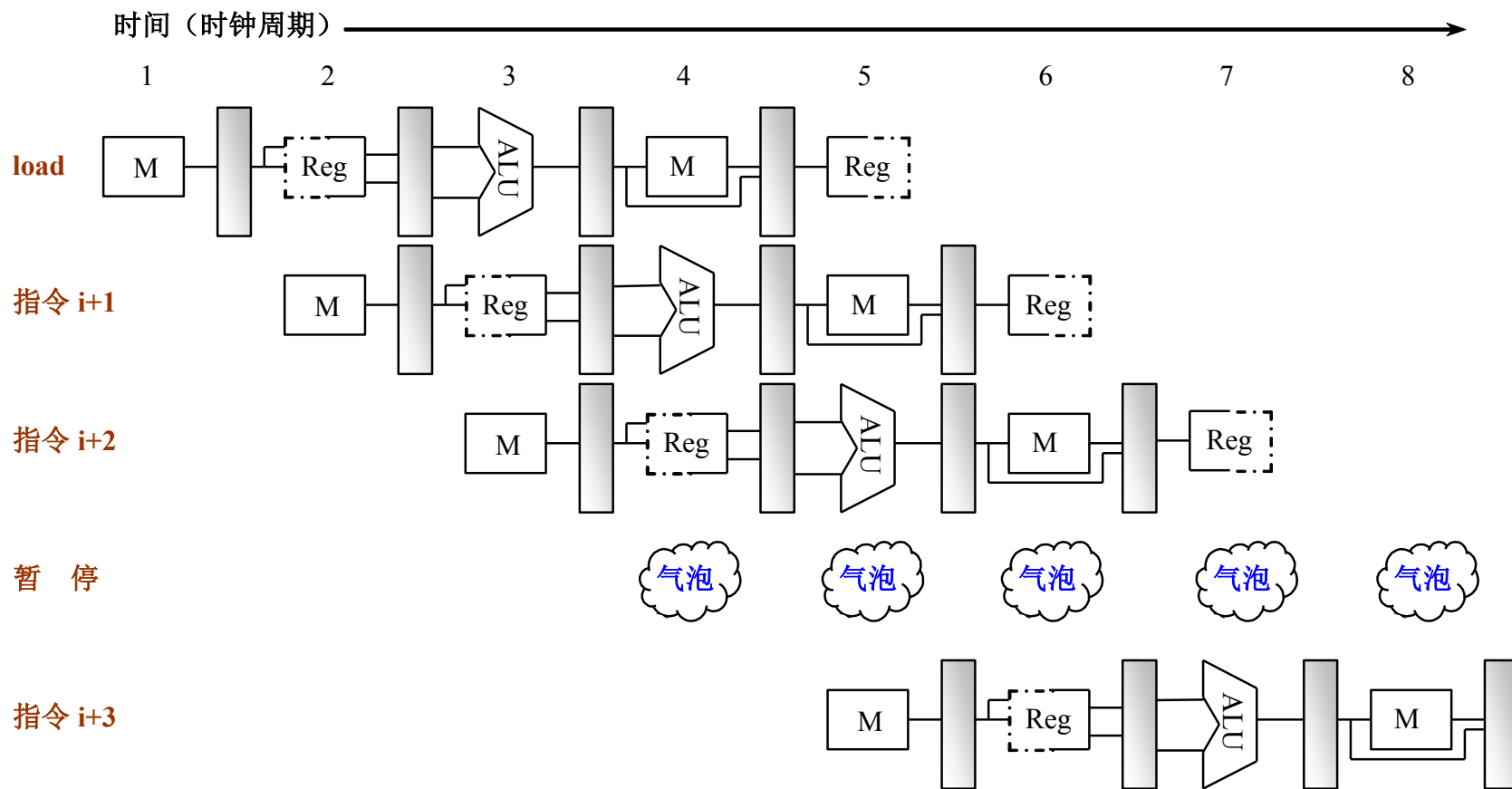
设置相互独立的指令存储器和数据存储器或设置相互独立的指令Cache和数据Cache。

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

3.4 流水线的相关与冲突

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
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	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

3.4 流水线的相关与冲突

- 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。

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。

3.4 流水线的相关与冲突

举例：

DADD R1, R2, R3

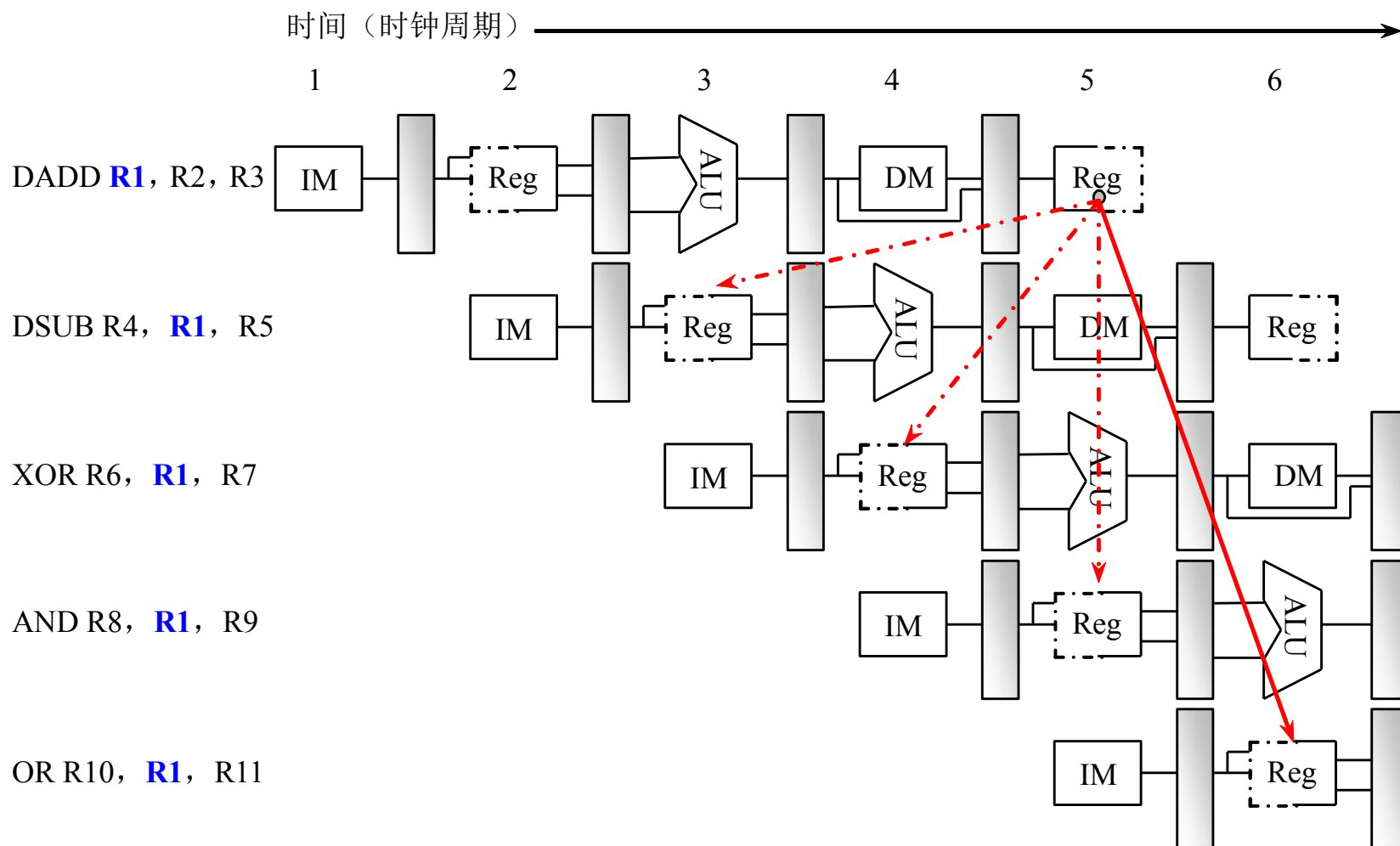
DSUB R4, R1, R5

XOR R6, R1, R7

AND R8, R1, R9

OR 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)

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：

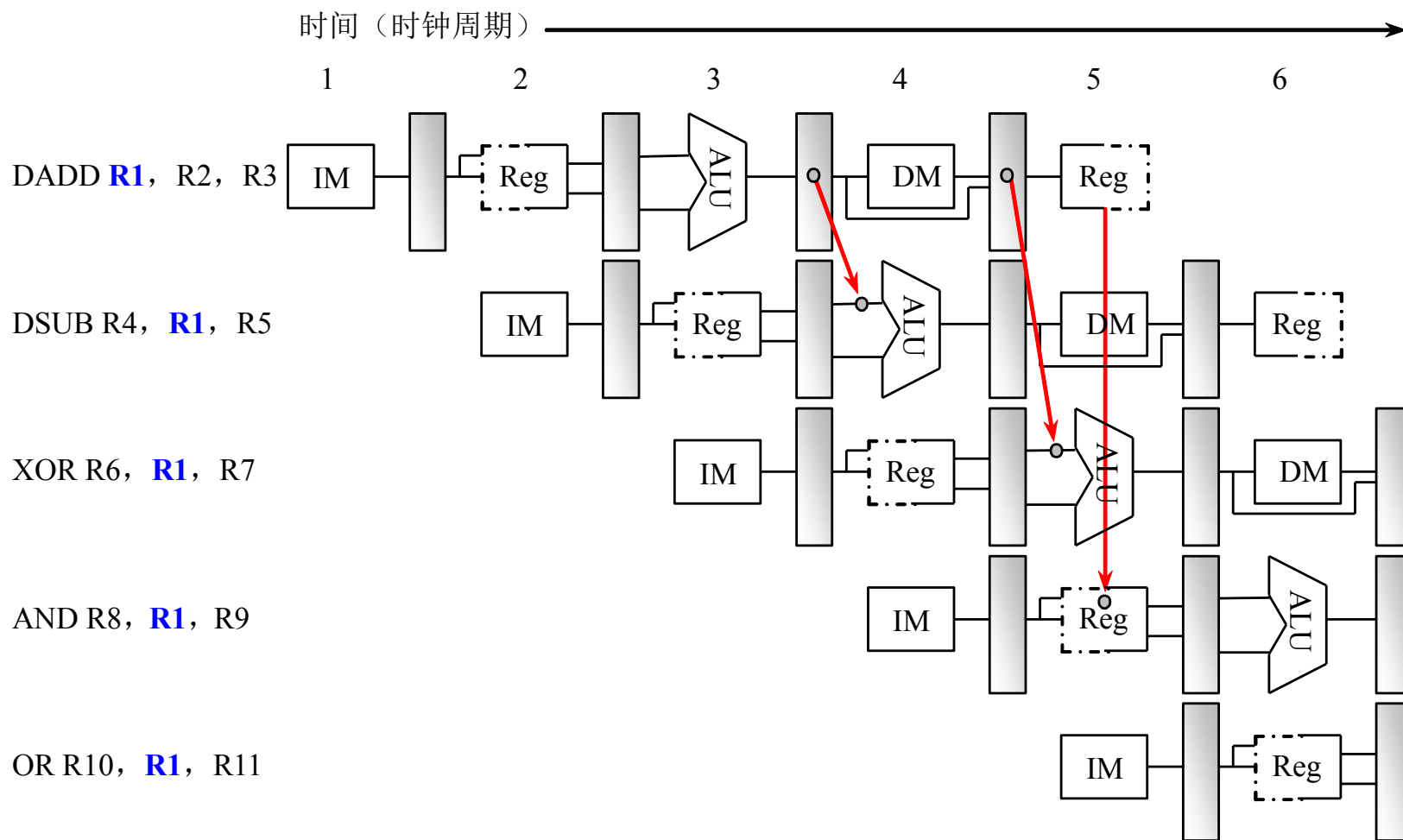
- 有些指令的写结果操作提前了，而且有些指令的读操作滞后了；
- 指令被重新排序了。

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。

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

■ 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。

LD R1, 0 (R2)

DADD R4, R1, R5

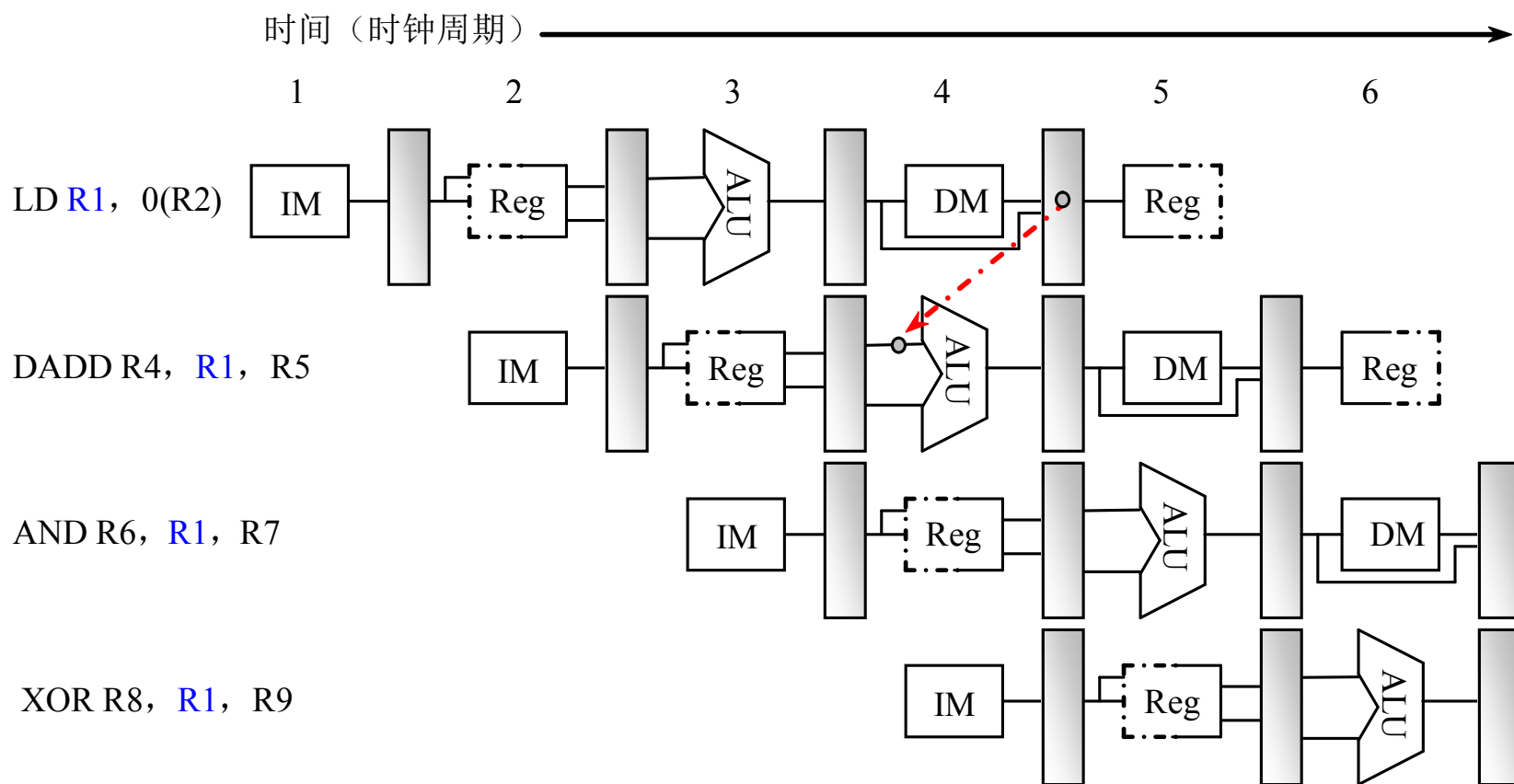
AND R6, R1, R7

XOR R8, R1, R9

- 增加流水线互锁机制，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

- 依靠编译器解决数据冲突
 - 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：

请为下列表达式生成没有暂停的指令序列：

$$A = B + C ;$$

$$D = E - F ;$$

假设载入延迟为1个时钟周期。

题解

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

- 不成功或者失败：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：

“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

3.4 流水线的相关与冲突

简单处理分支指令：分支成功的情况

[illegible]

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 $CPI = 1.9$

- 可采取两种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。

3.4 流水线的相关与冲突

下面的讨论中，我们假设：

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。

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

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。

■ 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，就好像什么都没发生似的；
- 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；

3.4 流水线的相关与冲突

- 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。

DLX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

■ 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。

■ 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

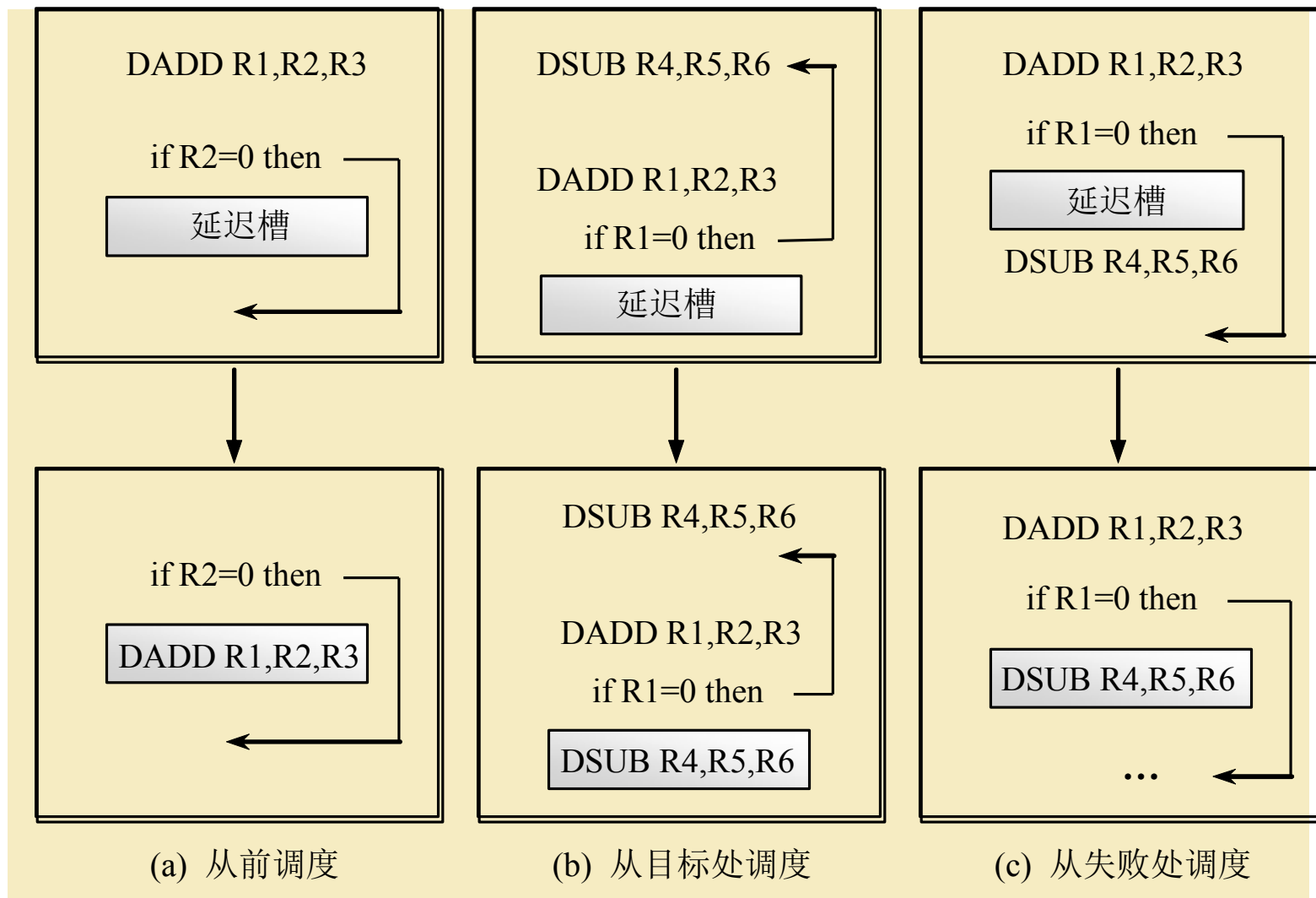
任务：在延迟槽中放入有用的指令

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。

进一步改进：分支取消机制 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

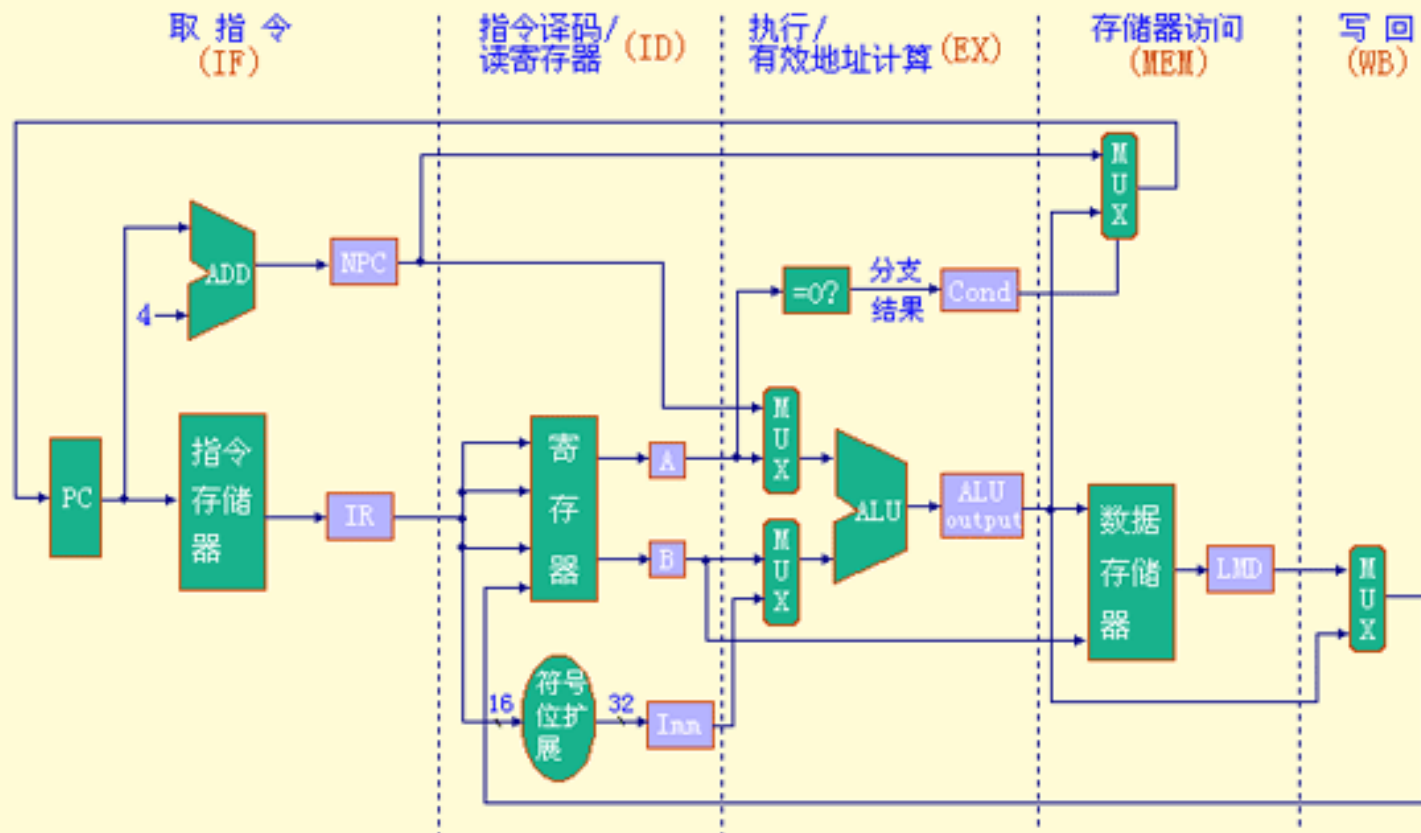
分支成功	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	ID	EX	MEM	WB			
	分支目标指令j			IF	ID	EX	MEM	WB		
	分支目标指令j+1				IF	ID	EX	MEM	WB	
	分支目标指令j+2					IF	ID	EX	MEM	WB

预测分支成功的情况下，分支取消机制的执行情况

介绍：MIPS的一种简单实现

- 实现MIPS指令子集的一种简单数据通路。
 - 该数据通路的操作分成5个时钟周期
 - 取指令
 - 指令译码/读寄存器
 - 执行/有效地址计算
 - 存储器访问/分支完成
 - 写回
 - 只讨论整数指令的实现（包括：load和store，等于0转移，整数ALU指令等。）

实现DLX指令的一种简单数据通路



- 设置了一些临时寄存器。其作用如下：
 - PC：程序计数器，存放当前指令的地址。
 - NPC：下一条程序计数器，存放下一条指令的地址。
 - IR：指令寄存器，存放当前正在处理的指令。
 - A：第一操作数寄存器，存放从通用寄存器组读出来的操作数。
 - B：第二操作数寄存器，存放从通用寄存器组读出来的另一个操作数。
 - Imm：存放符号扩展后的立即数操作数。
 - Cond：存放条件判定的结果。为“真”表示分支成功。
 - ALUo：存放ALU的运算结果。
 - LMD：存放load指令从存储器读出的数据。

一条MIPS指令最多需要以下5个时钟周期:

■ 取指令周期 (IF)

- $IR \leftarrow Mem[PC]$
- $NPC \leftarrow PC + 4$

■ 指令译码/读寄存器周期 (ID)

- $A \leftarrow Regs[rs]$
- $B \leftarrow Regs[rt]$
- $Imm \leftarrow ((IR_{16})^{16} \# IR_{16..31})$

指令的译码操作和读寄存器操作是并行进行的。

原因：在MIPS指令格式中，操作码字段以及rs、rt字段都是在固定的位置。

这种技术称为固定字段译码技术。

■ 执行/有效地址计算周期 (EX)

不同指令所进行的操作不同:

- load指令和store指令

$$ALUo \leftarrow A + Imm$$

- 寄存器—寄存器ALU指令

$$ALUo \leftarrow A \text{ funct } B$$

- 寄存器—立即值ALU指令

$$ALUo \leftarrow A \text{ op } Imm$$

- 分支指令

$$ALUo \leftarrow NPC + (Imm \ll 2) ;$$

$$cond \leftarrow (A == 0)$$

将有效地址计算周期和执行周期合并为一个时钟周期，这是因为MIPS指令集采用load / store结构，没有任何指令需要同时进行数据有效地址的计算、转移目标地址的计算和对数据进行运算。

■ 存储器访问/分支完成周期（MEM）

- 所有指令都要在该周期对PC进行更新。

除了分支指令，其它指令都是做： $PC \leftarrow NPC$

- 在该周期内处理的MIPS指令仅仅有load、store和分支三种指令。

- load指令和store指令

$LMD \leftarrow Mem[ALUo]$

或者 $Mem[ALUo] \leftarrow B$

- 分支指令

if (cond) $PC \leftarrow ALUo$ else $PC \leftarrow NPC$

■ 写回周期 (WB)

不同的指令在写回周期完成的工作也不一样。

- 寄存器—寄存器ALU指令

$Regs[rd] \leftarrow ALUo$

- 寄存器—立即数ALU指令

$Regs[rt] \leftarrow ALUo$

- load指令

$Regs[rt] \leftarrow LMD$

3.5 小结

■ 流水线技术

- 流水基本概念、 2种“瓶颈”解决方法、先行控制技术、流水线分类；
- 时空图及流水线性能分析与计算；
- 非线性流水线调度方法；
- 流水线相关、冲突的概念及解决方法；

3.6 作业与习题

- 作业： 3.1名词解释（流水线技术，单功能流水线，多功能流水线，静态流水线，动态流水线，线性流水线，非线性流水线，数据相关，控制相关。 3.6, 3.7, 3.8, 3.9, 3.10.

[习题1]

假设一条指令的执行过程分为“取指令”、“分析”和“执行”三段，每一段的执行时间分别为 Δt 、 $2\Delta t$ 和 $3\Delta t$ 。在下列各种情况下，分别写出连续执行 n 条指令所需要的时间表达式。

(1) 顺序执行方式。

(2) 仅“取指令”和“执行”重叠。

(3) “取指令”、“分析”和“执行”重叠

[习题1解答]

(1) 顺序执行需要的时间如下：

$$T = (\Delta t + 2\Delta t + 3\Delta t) \times n = 6n\Delta t$$

(2) 取指令和执行重叠，即一次重叠执行方式，我们假设第 $n+1$ 条指令的取指令和第 n 条指令的执行同时结束，那么所需要的时间为：

$$T = \Delta t + (2\Delta t + 3\Delta t) \times n = 5n\Delta t + \Delta t$$

(3) 取指令、分析和执行重叠

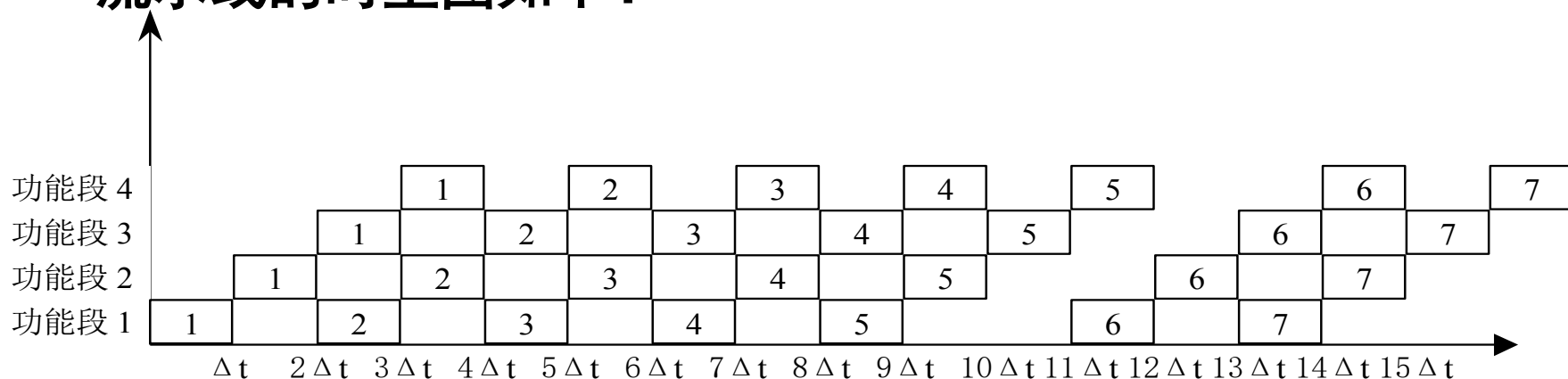
$$T = (\Delta t + 2\Delta t + 3\Delta t) + (n - 1)3\Delta t = 5\Delta t + 3n\Delta t$$

[习题2]

一条线性流水线有4个功能段组成，每个功能段的延迟时间都相等，都为 Δt 。开始5个任务是每间隔一个 Δt 输入一个任务进流水线，然后停顿2个 Δt 再输入下5个任务，如此循环往复。求流水线的实际吞吐率、加速比和效率。

[习题2解答]

流水线的时空图如下：



在 $(11n+1)\Delta t$ 的时间内，可以输出 $5n$ 个结果，如果指令的序列足够长 ($n \rightarrow \infty$)，并且指令间不存在相关，那么，吞吐率可以认为满足：

$$Tp = \frac{5n}{(11n+1)\Delta t} = \frac{5}{(11+1/n)\Delta t} = \frac{5}{11\Delta t} (n \rightarrow \infty)$$

加速比为：

$$S = \frac{5n \times 4\Delta t}{(11n+1)\Delta t} = \frac{20n}{11n+1} = \frac{20}{11+1/n} = \frac{20}{11} (n \rightarrow \infty)$$

从上面的时空图很容易看出，效率为：

$$E = \frac{T_0}{k \times T_k} = \frac{20n\Delta t}{4 \times (11n+1)\Delta t} = \frac{5}{11+1/n} = \frac{5}{11} (n \rightarrow \infty)$$

[习题3]

用一条5个功能段的浮点加法器流水线计算

$$F = \sum_{i=1}^{10} A_i$$

每个功能段的延迟时间均相等，流水线的输出端和输入端之间有直接数据通路，而且设置有足够的缓冲寄存器。要求用尽可能短的时间完成计算，画出流水线时空图，并计算流水线的实际吞吐率、加速比和效率。

求10个数的和需要做9次加法。在尽可能快的情况下，要注意前后指令之间的相关。

I1:	$R1 \leftarrow A1 + A2$
I2:	$R2 \leftarrow A3 + A4$
I3:	$R3 \leftarrow A5 + A6$
I4:	$R4 \leftarrow A7 + A8$
I5:	$R5 \leftarrow A9 + A10$
I6:	$R6 \leftarrow R1 + R2$
I7:	$R7 \leftarrow R3 + R4$
I8:	$R8 \leftarrow R5 + R6$
I9:	$F \leftarrow R7 + R8$

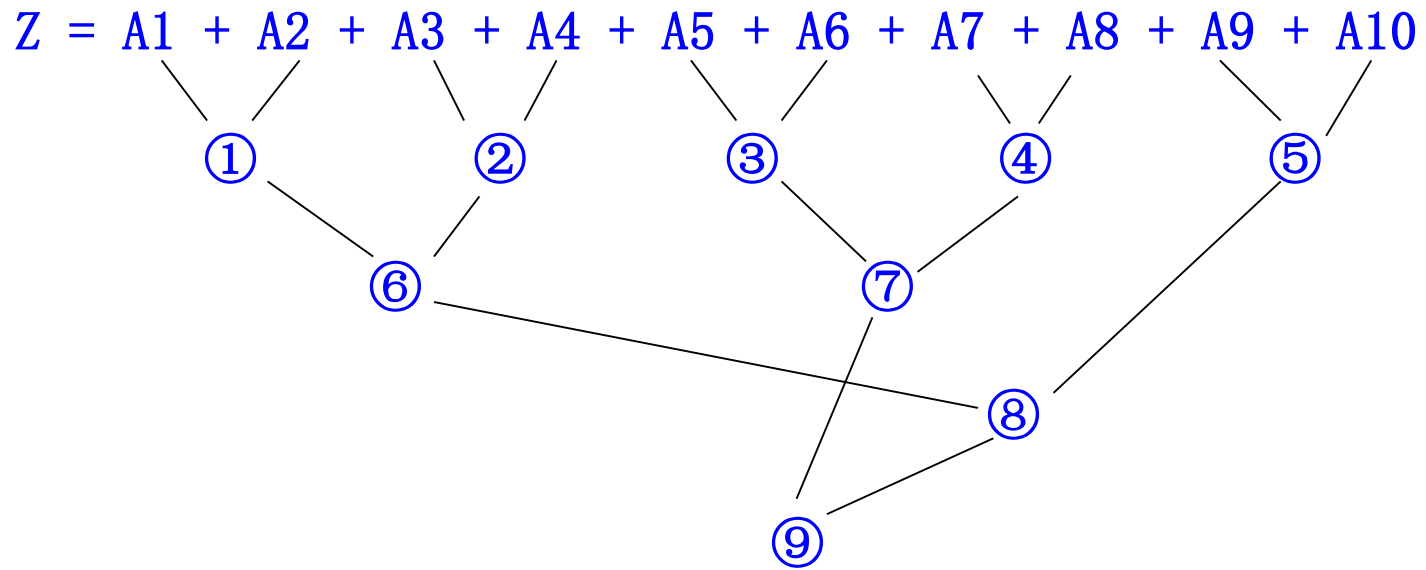
指令间存在相关:

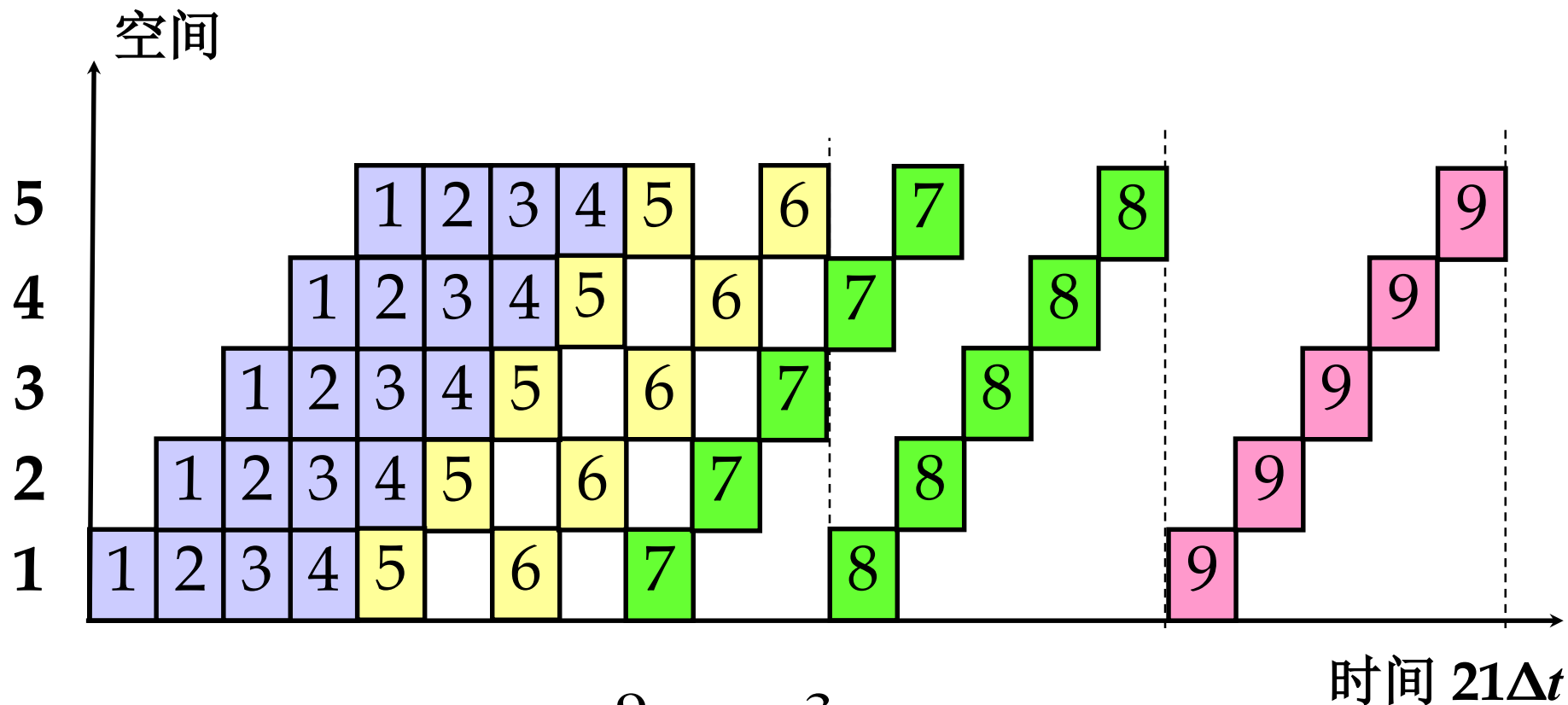
6必须在1,2后面执行

7必须在3,4后面执行

8必须在5,6后面执行

9必须在7,8后面执行





$$Tp = \frac{9}{21\Delta t} = \frac{3}{7\Delta t}$$

$$S = \frac{9 \times 5\Delta t}{21\Delta t} = \frac{45}{21} = 2.1429$$

$$E = \frac{T_0}{k \times T_k} = \frac{9 \times 5\Delta t}{5 \times 21\Delta t} = \frac{3}{7}$$

[习题4]

一条线性动态多功能流水线由6个功能段组成，加法操作使用其中的1、2、3、6功能段，乘法操作使用其中的1、4、5、6功能段，每个功能段的延迟时间均相等。流水线的输入端与输出端之间有直接数据通路，而且设置有足够的缓冲寄存器。现在用这条流水线计算：

$$F = \sum_{i=1}^6 (A_i \times B_i)$$

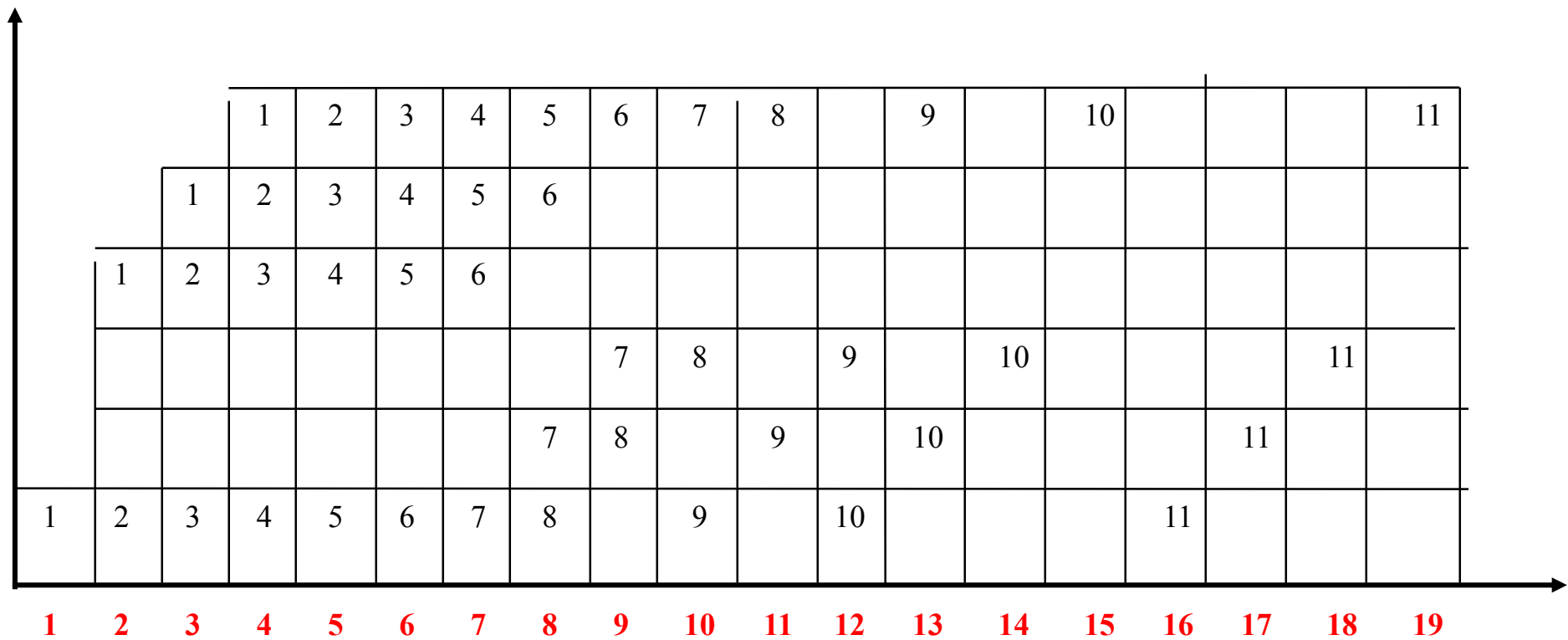
画出流水线时空图，并计算流水线的实际吞吐率、加速比和效率。

[习题4解答]

为了取得较高的速度，我们需要一次将乘法作完，设源操作数存放在寄存器A、B中，中间结果存放在寄存器R中，最后结果存放在寄存器F中，则执行的指令序列如下所示：

I1:	$R1 \leftarrow A1 * B1$
I2:	$R2 \leftarrow A2 * B2$
I3:	$R3 \leftarrow A3 * B3$
I4:	$R4 \leftarrow A4 * B4$
I5:	$R5 \leftarrow A5 * B5$
I6:	$R6 \leftarrow A6 * B6$
I7:	$R7 \leftarrow R1 + R2$
I8:	$R8 \leftarrow R3 + R4$
I9:	$R9 \leftarrow R5 + R6$
I10:	$R10 \leftarrow R7 + R8$
I11:	$F \leftarrow R9 + R10$

这并不是唯一可能的计算方法。假设功能段的延迟为 Δt 。时空图（不完全）如下，图中的数字是指令号。



整个计算过程需要 $19\Delta t$ ，所以吞吐率为：

$$Tp = \frac{11}{19\Delta t}$$

加速比为：

$$S = \frac{11 \times 4\Delta t}{19\Delta t} = \frac{44}{19}$$

效率为：

$$E = \frac{T_0}{k \times T_k} = \frac{11 \times 4\Delta t}{6 \times 19\Delta t} = \frac{22}{57}$$

思考一下：如果改为静态流水线呢？