计算系统结构

Computer Architecture

计算机与大数据学院
林嘉雯
ljw@fzu.edu.cn

如何加快机器语言解释的速度？

1. 加快每条指令的解释。
2. 加快整个机器语言程序的解释，提高指令间并行性。

采用重叠、流水方式来加快整个机器语言程序的解释

第 3章 标量处理机

主要内容：

- 重叠方式
- 流水方式
- 超级处理机

3.1 重叠方式

3.1.1 重叠原理与一次重叠

对一条机器指令的解释

取指令：从存储器中取出指令操作码-----指令寄存器中

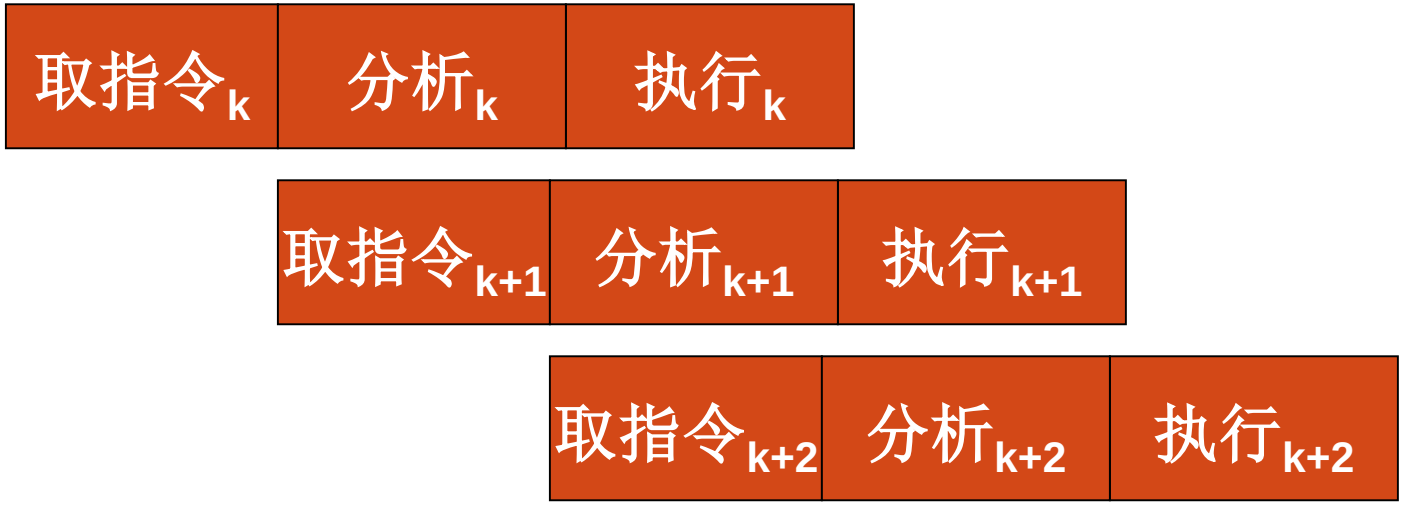
分析指令：指令寄存器----指令译码器----产生控制信号、操作

执行指令：用控制信号----执行部件完成相应操作

取指令$_k$	分析$_k$	执行$_k$	取指令$_{k+1}$	分析$_{k+1}$	执行$_{k+1}$

顺序串行完成指令的解释，每执行一条，地址自动加1，再执行下一条

- 顺序解释：各条机器指令之间顺序串行地执行，执行完一条指令后采取出下条指令来执行，而且每条指令内部的各个微操作也是顺序串行地执行。
  - 控制简单
  - 但速度慢、各部件利用率低。



重叠解释的一种方式

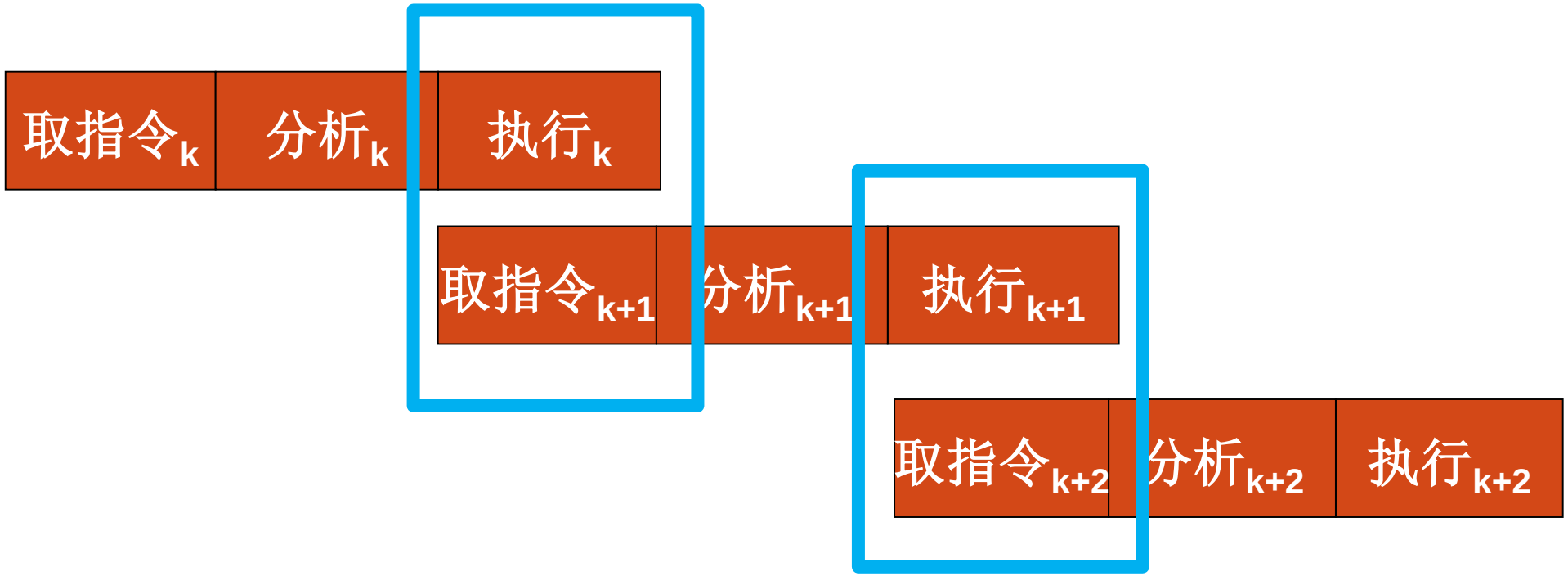
在时间上，让前一条的“分析”过程与后一条的“取指令”重叠在一起，缩短相邻两条指令的解释时间。

- 重叠解释：在解释第k条指令的操作完成之前，就可开始解释第k+1条指令

  不能加快一条指令的实现

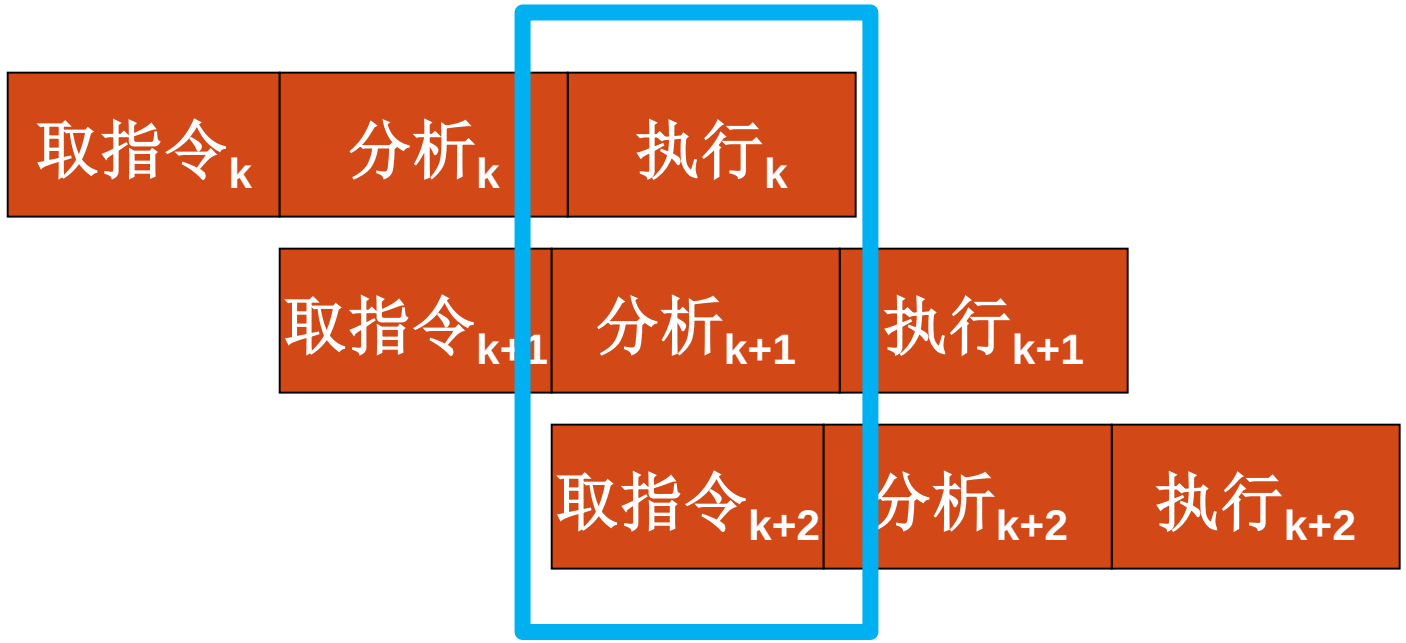
  但能加快相邻两条以至一段程序的解释

“一次重叠”



在任何时刻，指令分析部件和指令执行部件止只有相邻两条指令在重叠解释的方式，称为“一次重叠”。

“二次重叠”



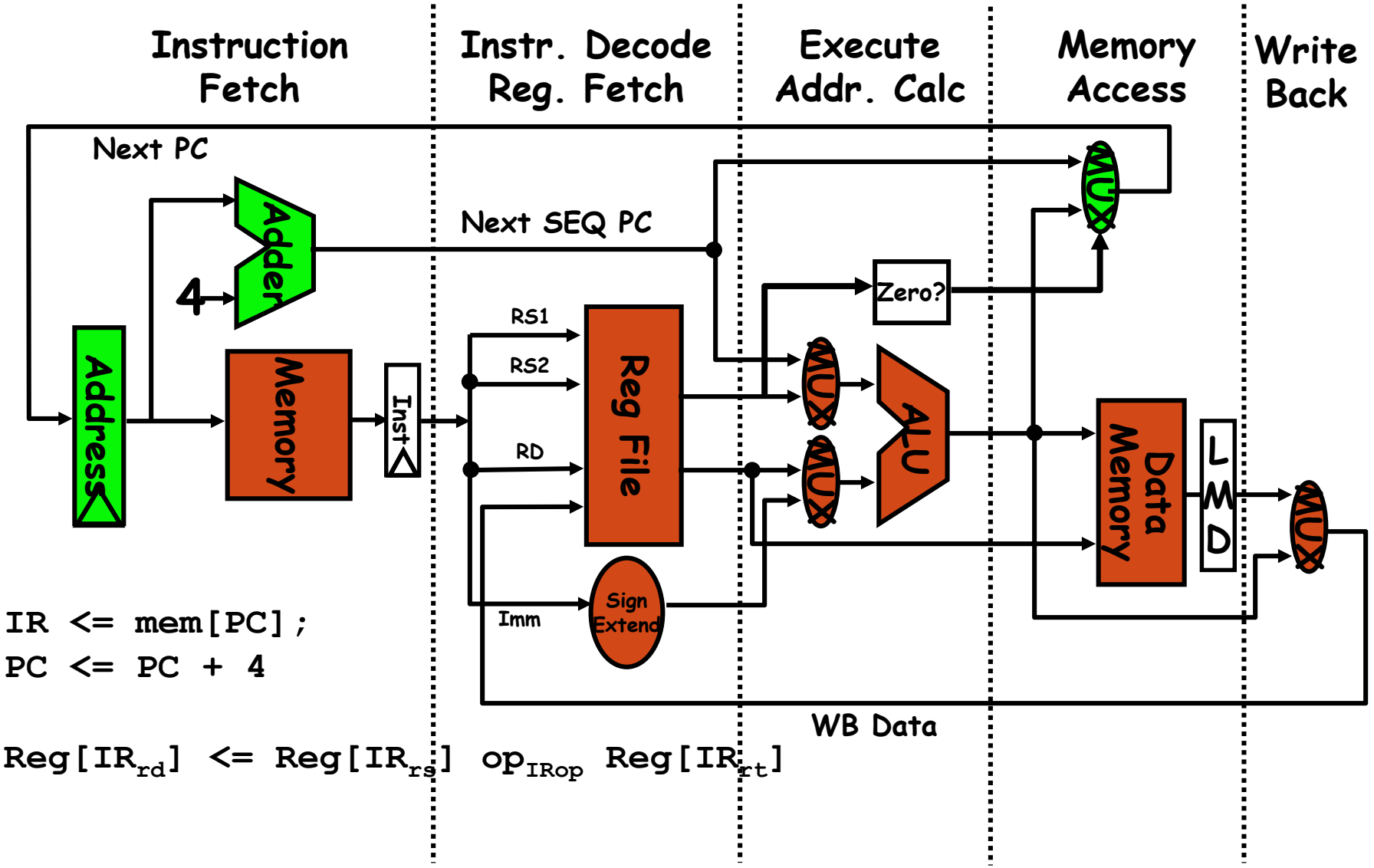
同时解释3条指令

一次重叠的特点

- 节省硬件，机器内指令分析部件和指令执行部件均只需一套，也简化了控制。
- 在一次重叠中，要求分析指令、执行指令的时间要尽可能等长，重叠方式才能有较高的效率。但是在一般情况下，分析指令和执行指令的时间不等长。

采用重叠方式的机器大多采用“一次重叠”，若达不到速度要求，则改用流水线。

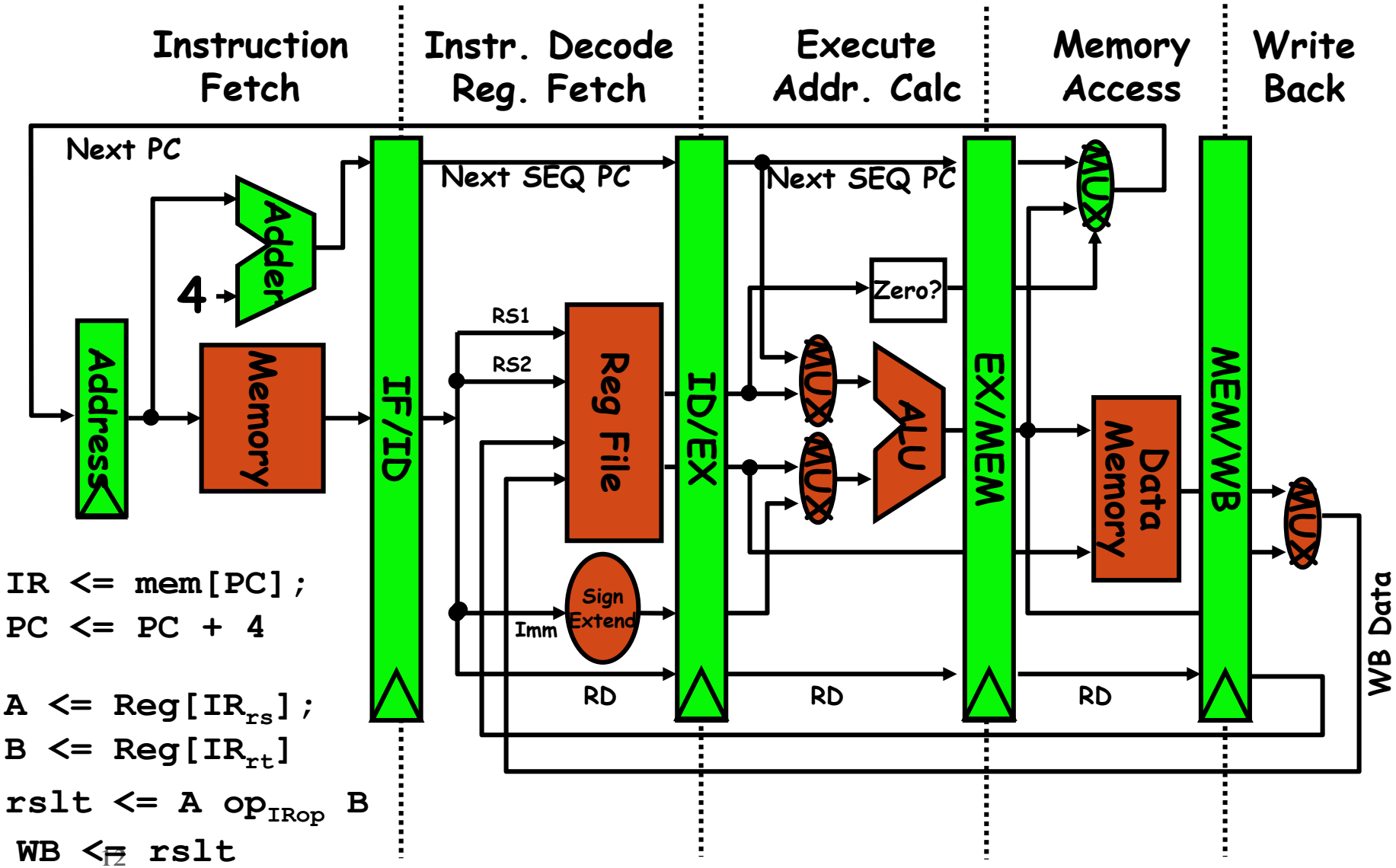
5 Steps of DLX Datapath



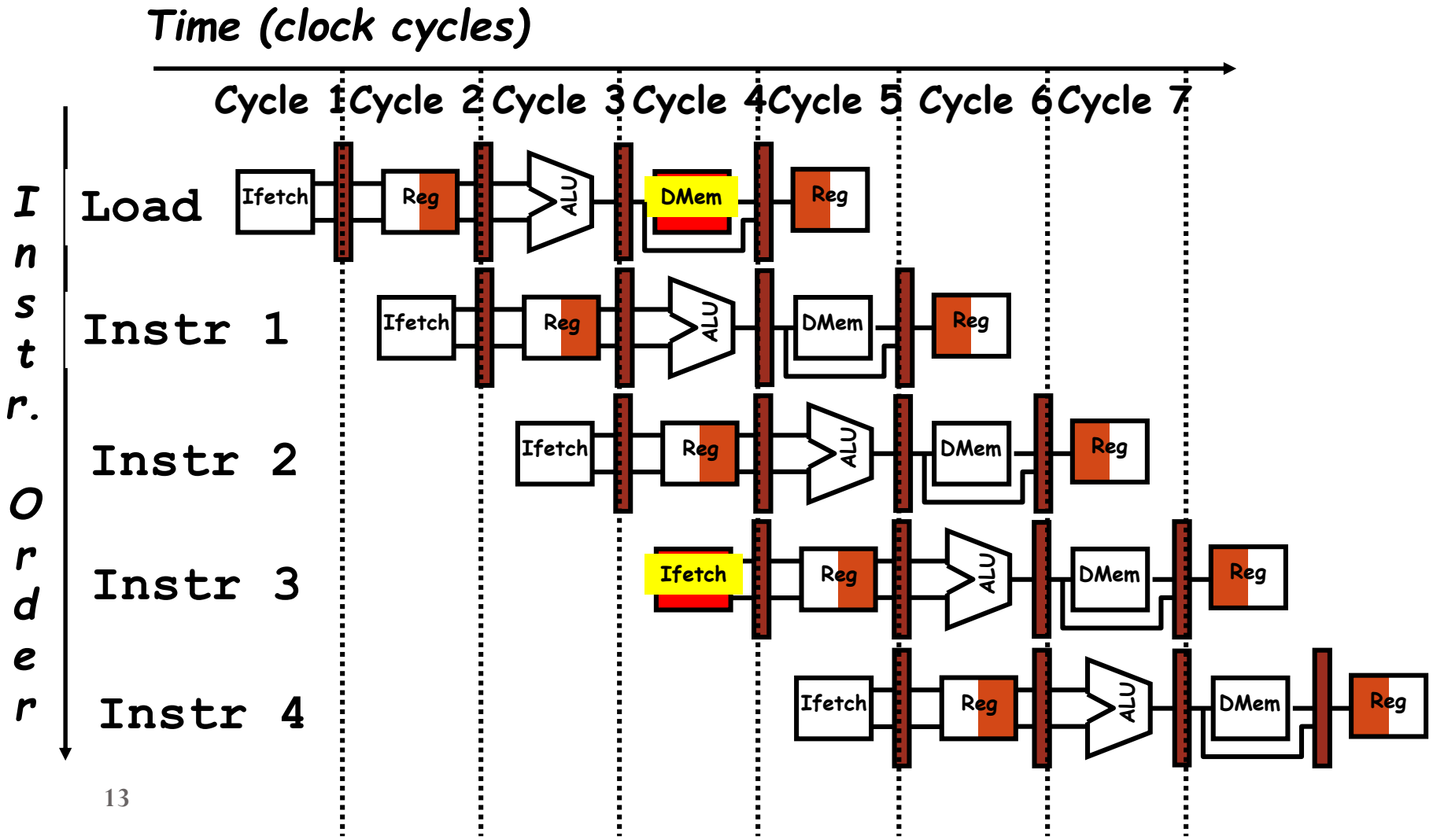

```
IR <= mem[PC];
PC <= PC + 4

Reg[IR_rd] <= Reg[IR_rs] op_IRop Reg[IR_rt]
```

5 Steps of DLX Datapath



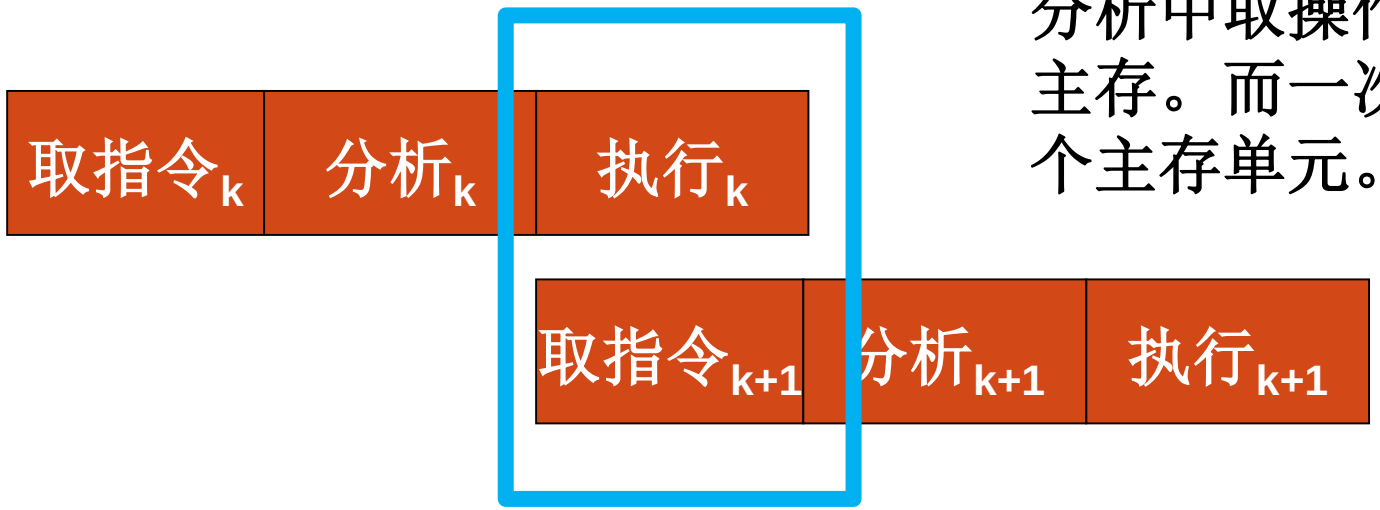
One Memory Port/Structural Hazards



重叠的实现对计算机组成有什么要求？

1、解决可能存在的访存冲突

起因：一般的机器上，操作数和指令混存于同一主存内，取指需要访主存，分析中取操作数也可能访主存。而一次只能访问一个主存单元。

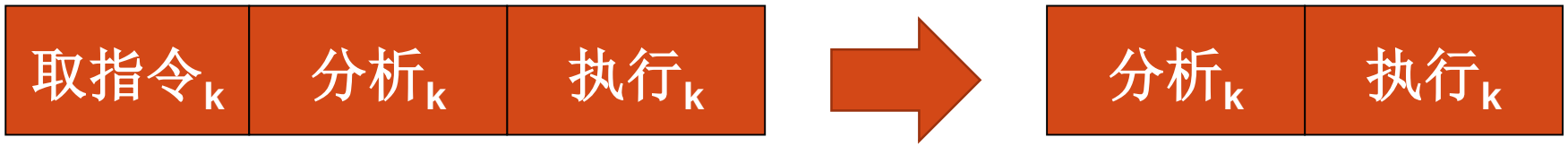


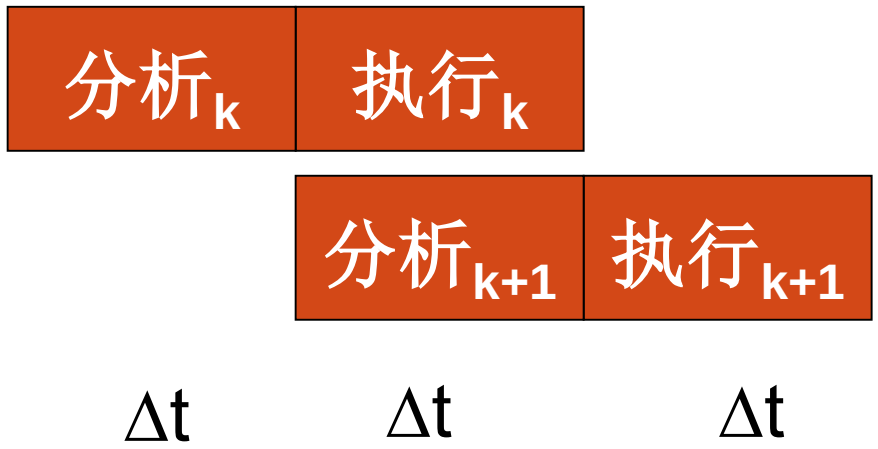
解决访存冲突的途径

- 操作数和指令分存于两个独立编址且可同时访问的存储器，有利于实现指令保护，增加总线控制和软件设计的复杂性
- 采用多体交叉主存结构

  当第k条指令所需的操作数与K+1条指令不在同一体内，仍可再一个主存周期内取得，从而实现重叠；若两者共存于一体则无法重叠
- 增设指令缓冲寄存器，则主存空闲时可预取下一条或几条指令于Cache中。

如果每次都可以从指缓中取得指令，则“取指$_{k+1}$”的时间很短，就可把这个微操作合并到“分析$_{k+1}$”内，则可将原先的重叠变成只是“分析$_{k+1}$”与“执行$_k$”的重叠





2、在硬件上保证有独立的指令分析部件和指令执行部件；解决控制的同步

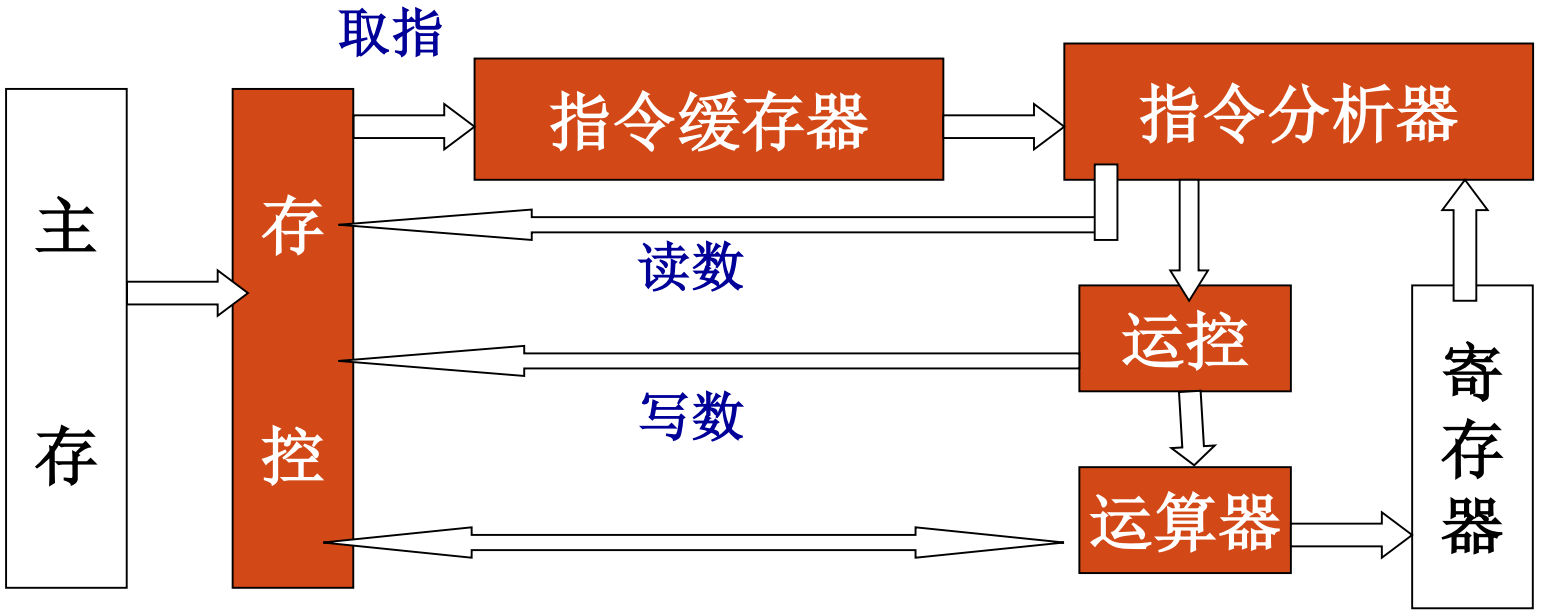
- 为了实现重叠方式，处理机的组成也应适应其需要的结构，增大一定的硬件开销。需要有独立的取指令部件、分析指令部件、执行指令部件。

  例如，分析部件要包含加法器，用以地址的计算；执行部件要包含加法器，完成操作数的想加

- 实际应用中，“分析”和“执行”所需的时间常常不一样，保证任何时候只是“分析k+1”与“执行k”的重叠。

一次重叠的基本结构：

独立的取指令部件、分析指令部件、执行指令部件；可将原来集中、统一的控制器，分解为存储控制器、指令 控制器、运算控制器　三个部分

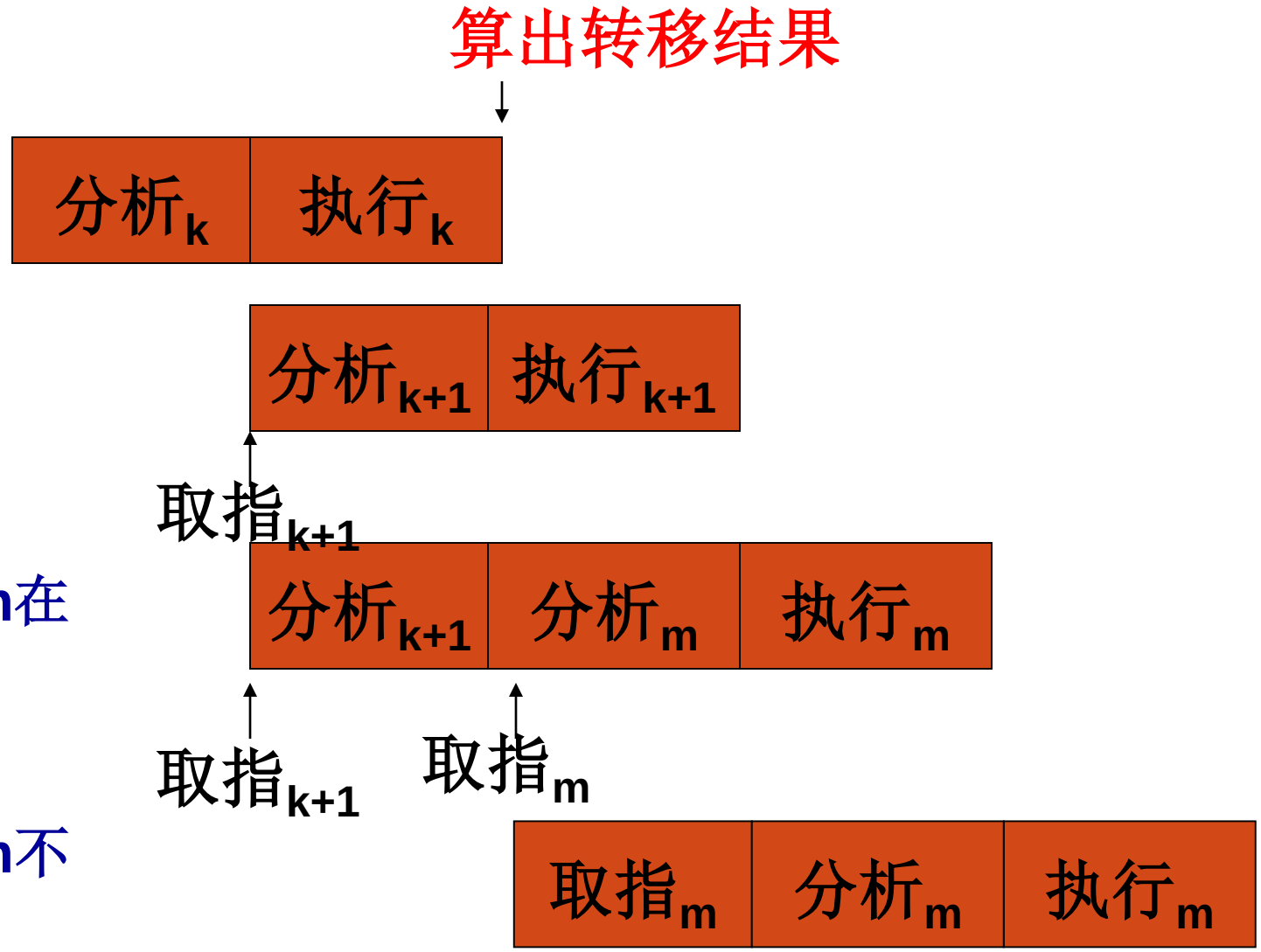


3、转移指令的处理问题

- 若第$_k$条指令是条件转移指令，当转移不成功，重叠操作有效；

- 若成功转移到*m*单元，显然下一条应执行第m条指令。由于在“执行k”的末尾才形成下一条要执行指令的地址，因此与“执行$_k$”重叠的“分析$_{k+1}$ ”需要撤销并从头分析第m条指令。

可见条件转移成功时，重叠实际变成了顺序。


算出转移结果
分析k
执行k
当转移不成功时：
分析k+1
执行k+1
取指k+1
当转移成功且指令m在指缓中时：
分析k+1
分析m
执行m
取指k+1
取指m
当转移成功且指令m不在指缓中时：
取指m
分析m
执行m
当第k条指令使条件转移时

转移指令的解决途径

- 采用重叠方式的机器中，应尽量减少使用条件转移语句。
- 若出现条件转移语句，可使用延迟转移技术等

如，将第k条转移指令与条件转移无关的第k-1条指令交换位置

4、指令相关的处理

- 因为机器语言程序中临近指令之间出现了关联，为防止出错让它们不能同时解释的现象就称为发生了“相关”。

如：当后继指令的操作数刚好是前一指令的运算结果

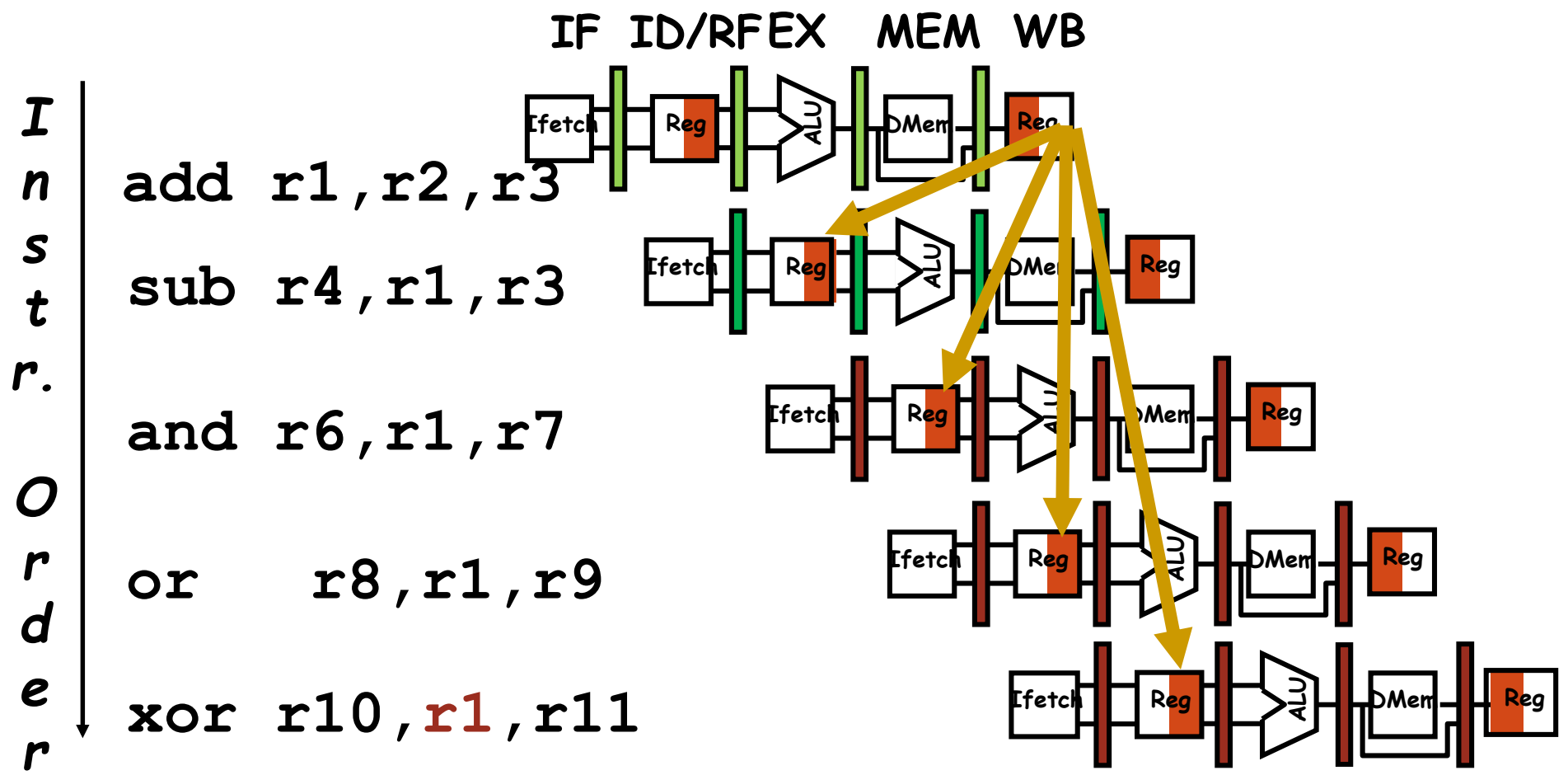
- 相关包括：指令相关、操作数相关。

- **指令相关：**后一指令的内容受前一指令的执行结果影响而产生的关联，造成两条指令不能同时解释。
  - 起因：Von Neumann型机器的指令允许修改。即可通过执行第k条指令来修改第k+1条指令。
  - 当指缓可存放n条指令时，执行到第k条指令时，与已预取进指缓的第k+1到第k+n条指令都有可能发生指令相关。
  - 指缓的容量愈大，发生指令相关的概率就愈高

- **操作数相关：** 两条指令的数据有了关联，造成两条指令不能同时解释。
  - 例如：第k+1条指令的操作数正好是第k条指令的运算结果。
  - 操作数相关不只是会发生在主存空间，还会发生在通用寄存器空间

无论何种相关，要么会使解释出错，要么会使重叠效率下降，所以必须正确处理

Data Hazard on R1



3.1.2 相关处理

1. 指令相关的处理

① 程序运行过程中不允许修改指令

- 同时可实现程序的可再入和程序的递归调用。
- 但为满足程序设计的灵活性，希望修改指令。

② 设置一条“执行”指令，将指令相关转化成操作数相关来解决

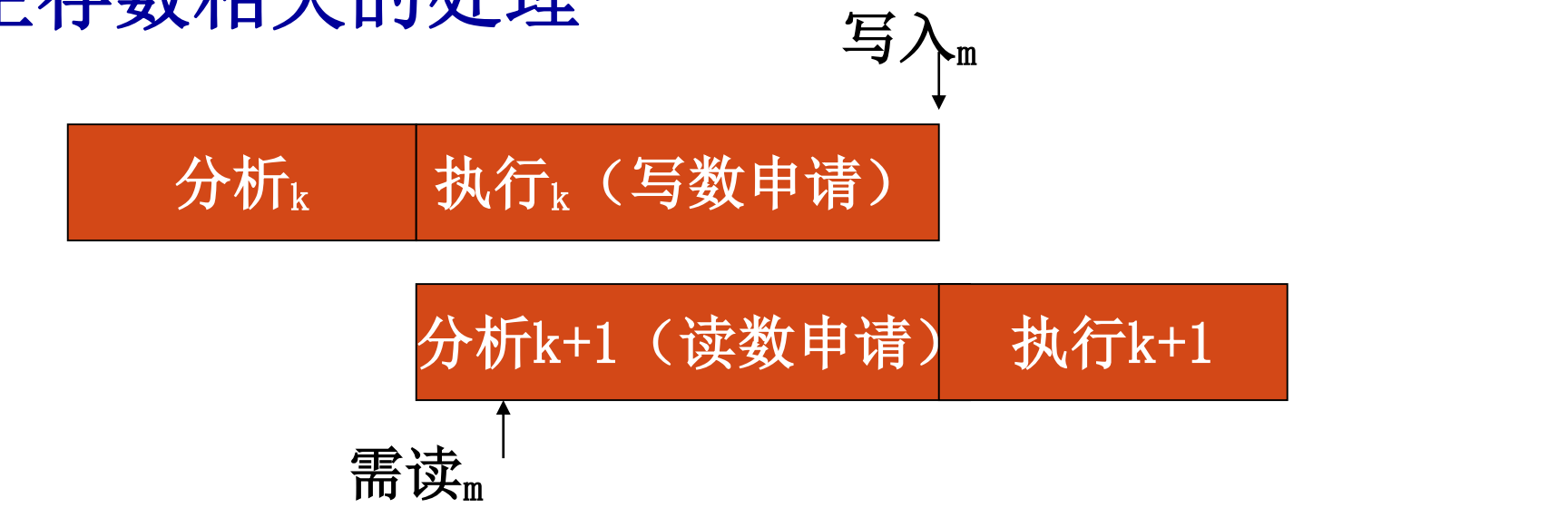
- 思想：可能被修改的指令以“执行”指令的操作数形式出现，即，指令码本身就是数据。指令相关实际上转化成操作数相关，统一用操作数相关的处理方法来解决。

2.主存空间数相关的处理

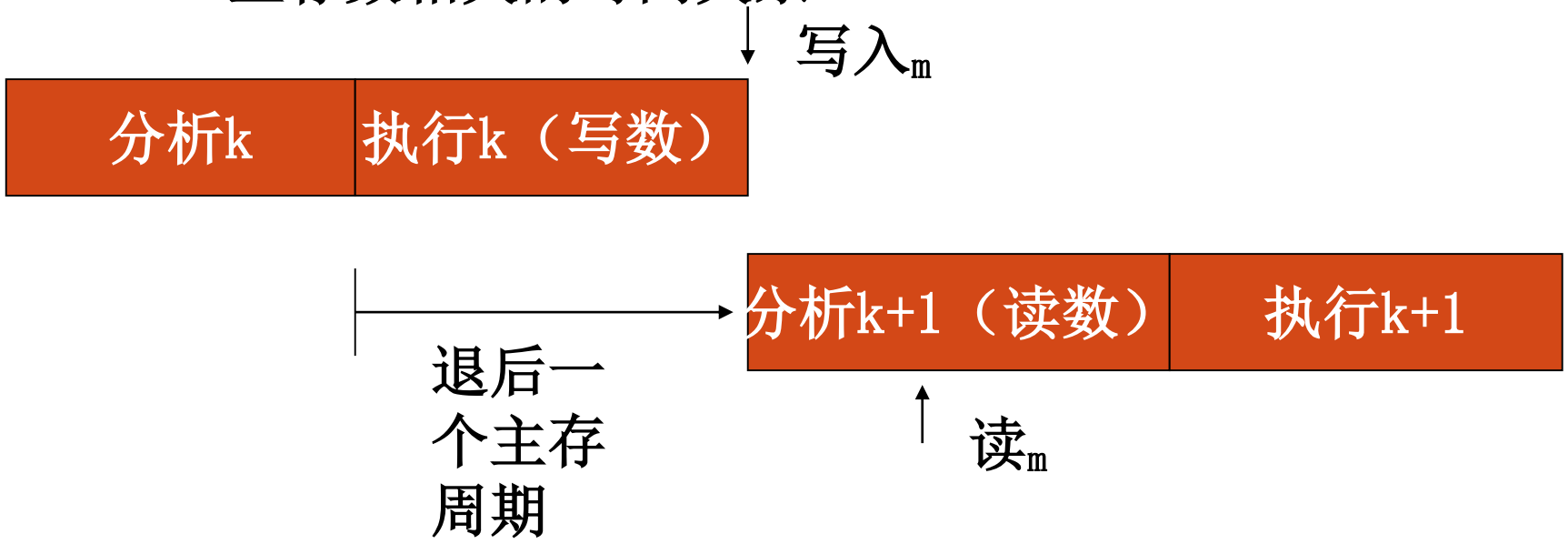
- 主存空间数相关：相邻两条指令之间出现对主存同一单元要求先写入后读出的关联。
- 如果让“执行k”与“分析k+1”在时间上重叠，就会使 “分析k+1”读出的操作数错误。

- 处理方法：推后读
  - 推后第k+1条指令的读操作数.
  - 具体方法：由存控给读数、写数申请安排不同的访存优先级来解决。
  - 通常很多机器都将访存优先级依次定为通道申请、写数、读数、取指。
  - 只要将写数级别安排成高于读数级别，则自动实现了推后读

主存数相关的处理



主存数相关的时间关系



3.通用寄存器空间数相关的处理

- 通用寄存器存放：操作数、运算结果、变址或基址
  - 存放于通用寄存器中的基址或变址值一般总是在“分析”周期的前半段就取出来用；
  - 操作数是在“分析”周期的后半段取出，到“执行”周期的前半段才用得上；
  - 运算结果是在“执行”周期的末尾才形成，并送入通用寄存器中

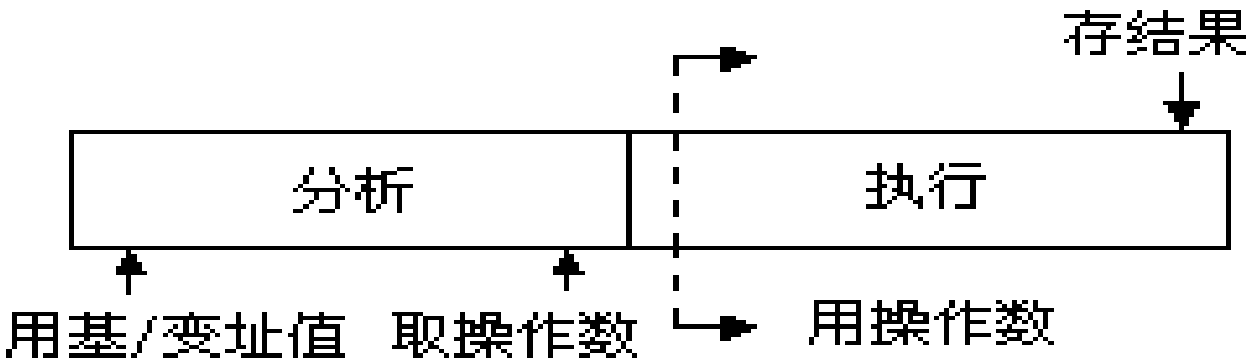
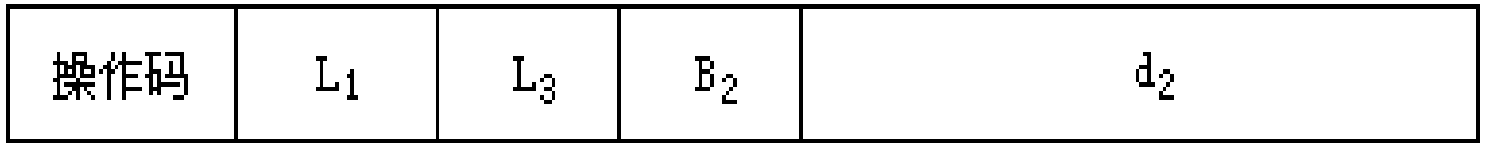


图4.8　与通用寄存器内容有关的微操作

- 机器的基本指令格式如下图所示。
  - L_1、L_3：存放第一操作数和运算结果的通用寄存器号，
  - B_2为形成第二操作数地址的基址值所在通用寄存器号，
  - d_2为相对位移量。



或

- 通用寄存器空间数相关的情况
  - 假设某台机器正常情况下，“分析”和“执行”的周期与主存周期一样都是4拍。
  - 有些指令需从通用寄存器组中取两个操作数（L_1）和（L_2），若通用寄存器组做在一个片子上，每次只能读出一个数，则在“分析k+1”期间，操作数（L_1）和（L_2）就需要在不同拍时取得，分别送入运算器的B和C寄存器，以便在“执行k+1”时用。

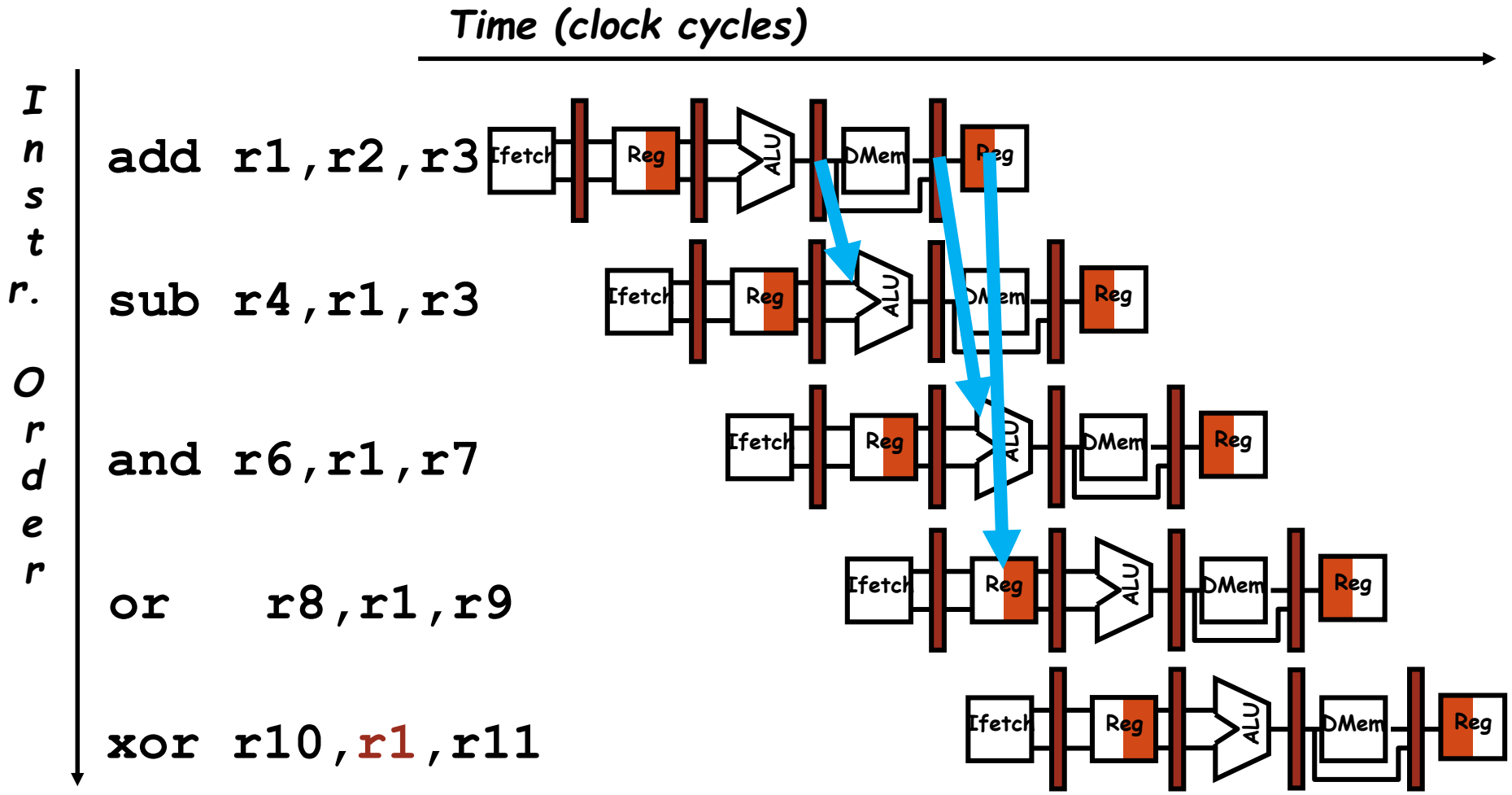
- 当程序执行过程中出现$L_{1\ (k+1)}=L_{3\ (k)}$时就发生了L_1相关；而当$L_{2\ (k+1)}=L_{3\ (k)}$时就发生了L_2相关。

- 处理办法
  - 解决方法1：推后读。具体有两种方法：
    - 把“分析k+1”推后到“执行k”结束时开始
      - 只要发生数相关就使一次重叠变成了完全的顺序串行，速度明显下降；
    - 把“分析k+1”只推后到“执行k”把结果送入L_3时，保证“分析k+1”在取（L_1）或（L_2）时能取到就可。
      - 相邻两条指令的解释仍有部分重叠，可以减少速度损失，但控制要稍微复杂一些。
  - 推后读方法靠牺牲速度来避免相关出错

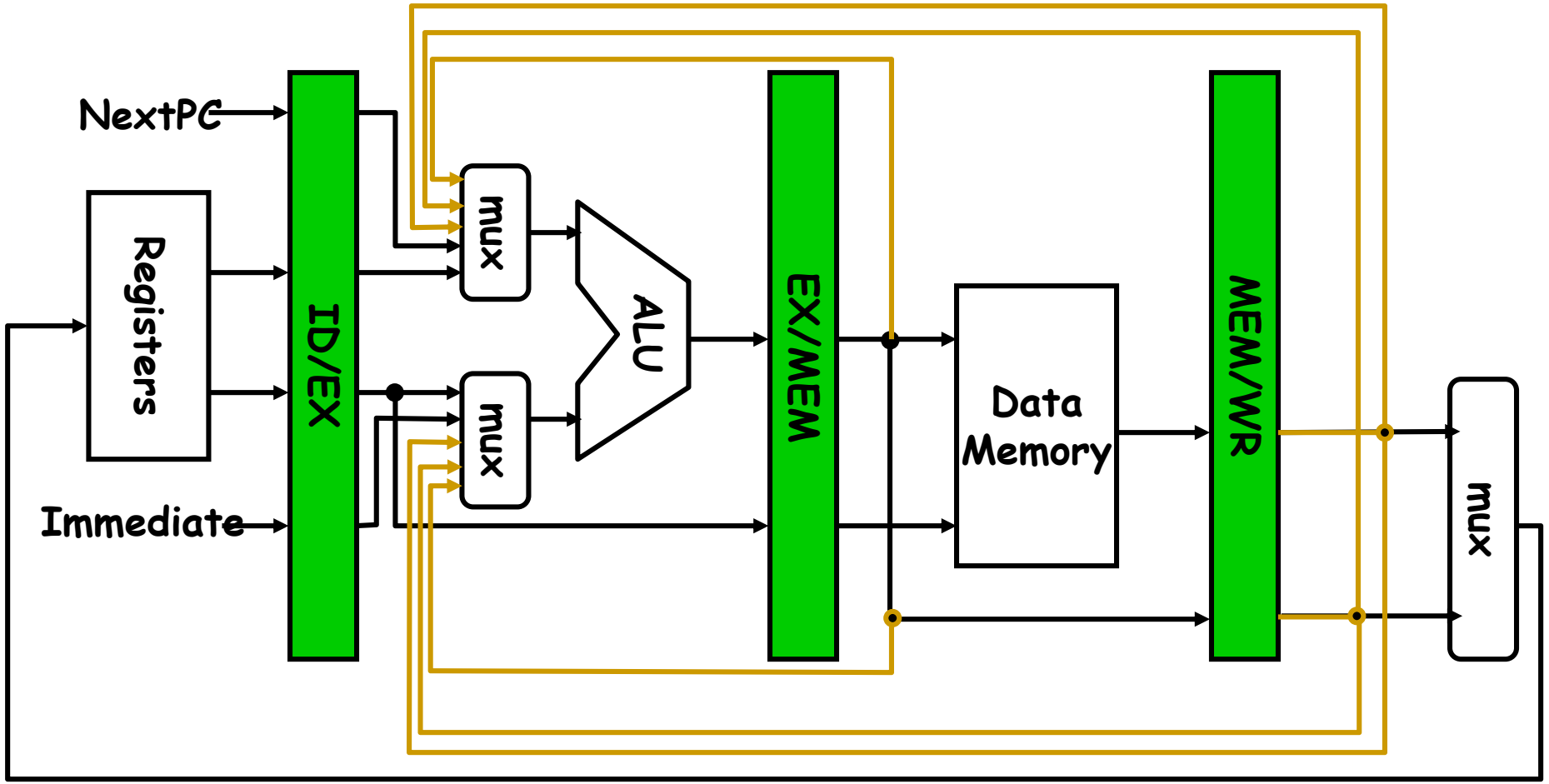
？问题：能否在不降低速度的情况下保证数相关时的正确处理呢？

- 解决方法2：增设相关专用通路
  - 在运算器的输出到操作数寄存器B或C输入之间增设一条“相关专用通路”
  - 在发生L_1或L_2相关时，让相关专用通路接通，在“执行k”时将运算结果送入通用寄存器，同时直接将运算结果回送到B或C寄存器，
  - 可大大缩短传送时间，并能保证当“执行k+1”需要用此操作数时，它已在B或C寄存器中准备好了

Forwarding to Avoid Data Hazard



HW Change for Forwarding



What circuit detects and resolves this hazard?

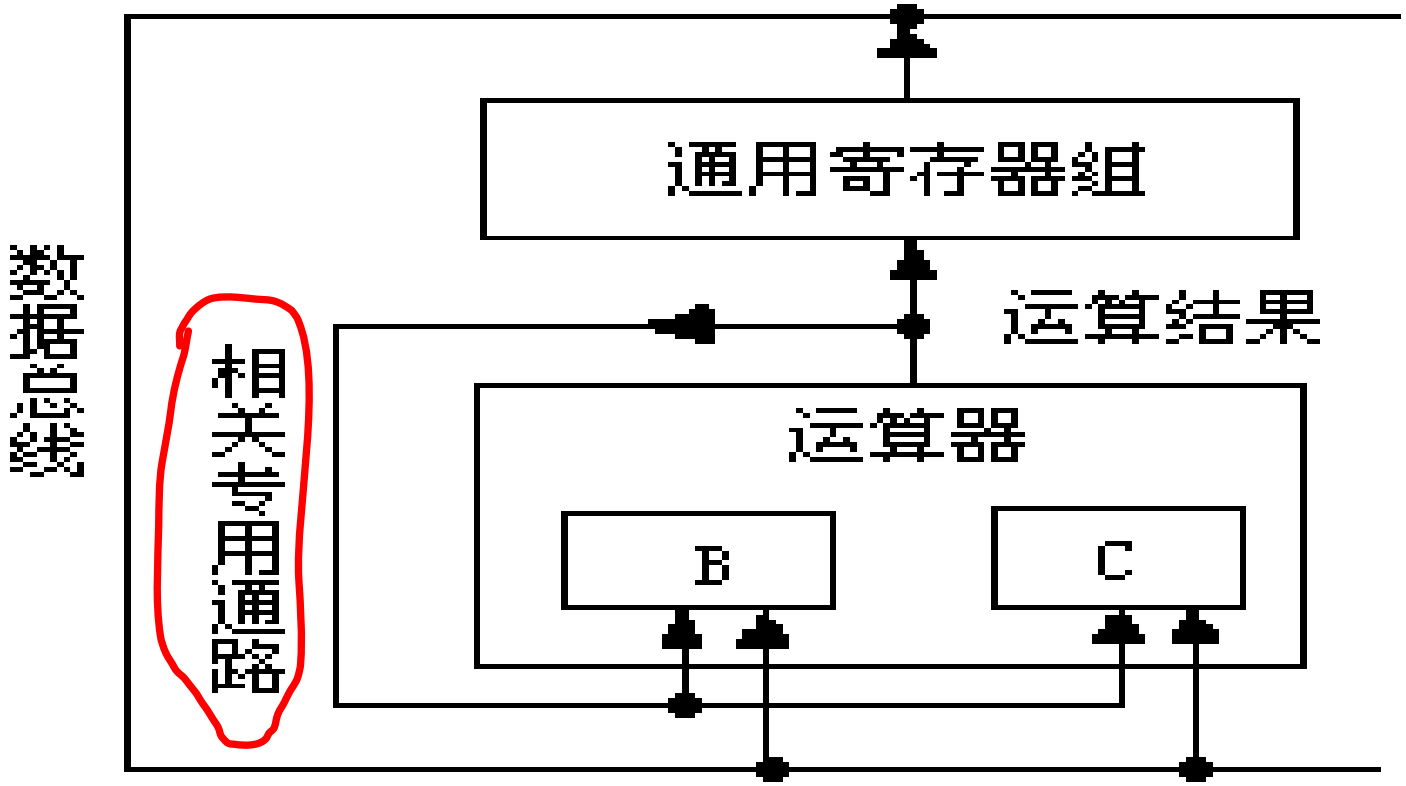


图4.11　相关专用通路

推后“分析$_{k+1}$” VS. 设置“相关专用通路”

- 解决重叠方式相关处理的两种基本方法。
- 推后“分析$_{k+1}$”以降低速度为代价，设备基本不增加；设置“相关专用通路”以增加设备为代价，重叠效率不下降。
- 设置“相关专用通路”也可用于解决主存空间数相关，但由于主存空间数相关的出现概率低很多，所以只采用推后读来解决

- 综上所述，为了实现两条指令在时间上的重叠解释

① 需要付出硬件代价

② 要处理好指令可能存在的相关。

- 相关处理的办法无非是“推后读”和设置相关专用通路两种，应当在成本和效率上加以权衡选用。

③ 合理安排好指令顺序及指令微操作的时间关系，使“分析”和“执行”所需的时间尽可能匹配，以提高重叠的效率

第 3章 标量处理机

主要内容：

- 重叠方式
- 流水方式
- 超级处理机

3.2 流水方式

3.2.1 基本概念

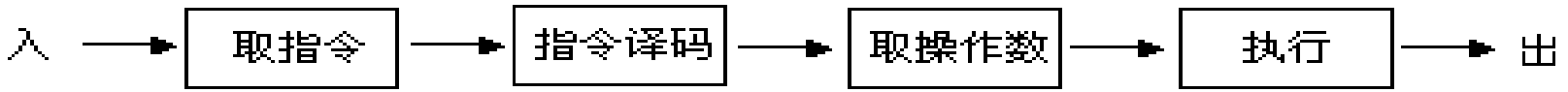
流水是重叠的引申，在一个任务完成以前就可以开始一个新的任务。

- 一次重叠中，若“分析”与“执行”子过程都需要Δt的时间
  - 需要2Δt完成一条指令的解释
  - 从机器的输出来看，每隔Δt就能完成一条指令的解释。
  - 同指令的串行解释相比，机器的最大吞吐率是原来的2倍。

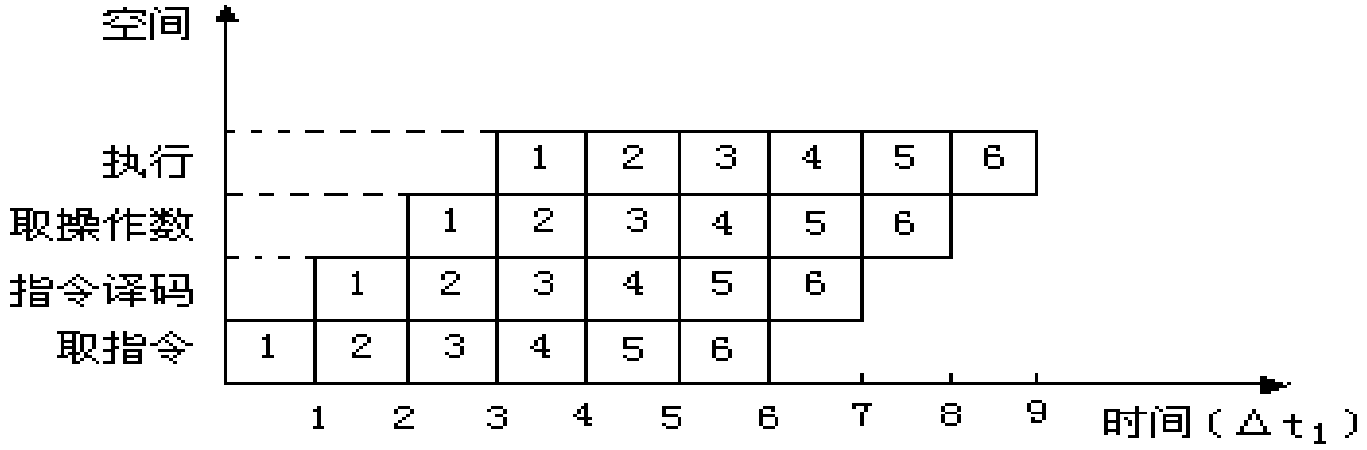
- **流水**：将指令分成更多的子过程，可同时解释多条指令，是多条指令的重叠处理，是更高程度的重叠。

流水技术特点：

- 一条流水线由多个流水段组成(多段)
- 每个流水段有专门的功能部件对指令进行某种加工(专件)
- 流水线工作阶段可分为建立、满载和排空三个阶段(三阶段)
- 在理想情况下，当流水线充满后，每隔 Δt时间将会有一个结果流出流水线。
- 流水技术适用于大量重复程序过程。只有不断提供输入，才能连续流水输出，机器效率才能充分发挥。



(a) 指令解释的流水处理



(b) 流水处理的时空图

- 如果能把一条指令的解释分解成时间相等的N个子过程，则每隔Δt=T/N就可以处理一条指令。

例：如果完成一条指令的时间为T，分别分析顺序解释、一次重叠和流水方式的执行情况。

①一次重叠分解为“分析”和“执行”两个子过程

②流水线分解为“取指令”、“指令译码”、“取操作数”和“执行”4个子过程

解：

- 顺序解释方式，每隔T “流出”一个结果
- 一次重叠方式，每隔T/2就可 “流出”一个结果，吞吐率为顺序解释的2倍
- 分为4个子过程的流水方式，每隔T/4 “流出”一个结果，即吞吐率为顺序方式的4倍

思考：子步骤是否划分的越细（越多）越好？

流水线并不是越长越好，找到一个速度与效率的平衡点才是最重要的。

流水线的分类

1、根据向下扩展和向上扩展分类

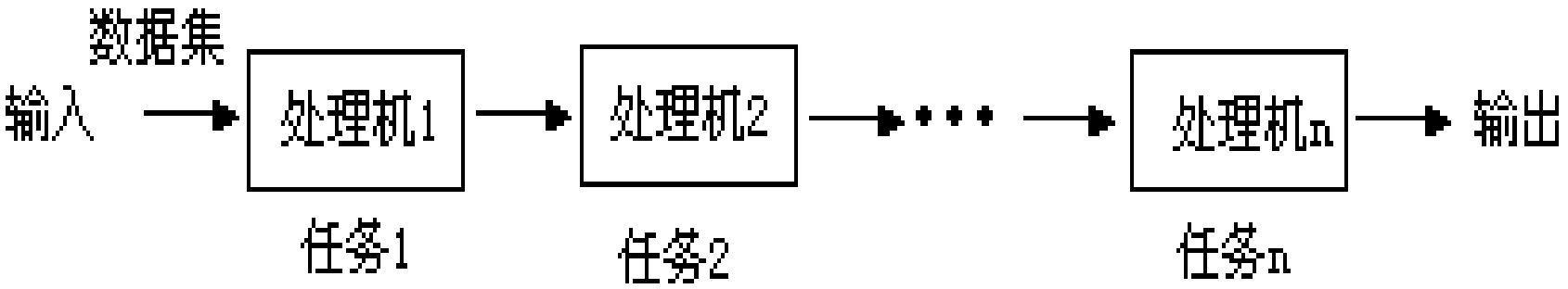
向下扩展：把子过程进一步细分，让每个子过程经过的时间都同等程度地减少，吞吐率会进一步提高。

- “执行”子过程会因指令不同，执行时间不同，而使细分的情形不同。如：浮点加法可进一步细分成为“求阶差”、“对阶”、“尾数相加”和“规格化”4个子过程

子过程的进一步细分以增加设备为代价

向上扩展：在多个处理机之间进行流水。

- 多个处理机串行地对数据集进行处理，某个处理机专门完成其中的一个任务（即专门执行特定功能的指令）
- 因为各个处理机都在同时工作，所以能对同一数据流的不同部分流水地处理，使计算机系统处理能力有较大的提高。



综上所述，**按流水处理的级别不同**，可以把流水线分为部件级、处理机级和系统级的流水线：

① **部件级流水**：构成部件内的各个子部件之间的流水，如运算器内浮点加法流水线。

② **处理机级流水**：指构成处理机的各个部件的流水，如“取指”、“分析”、“执行”间的流水。

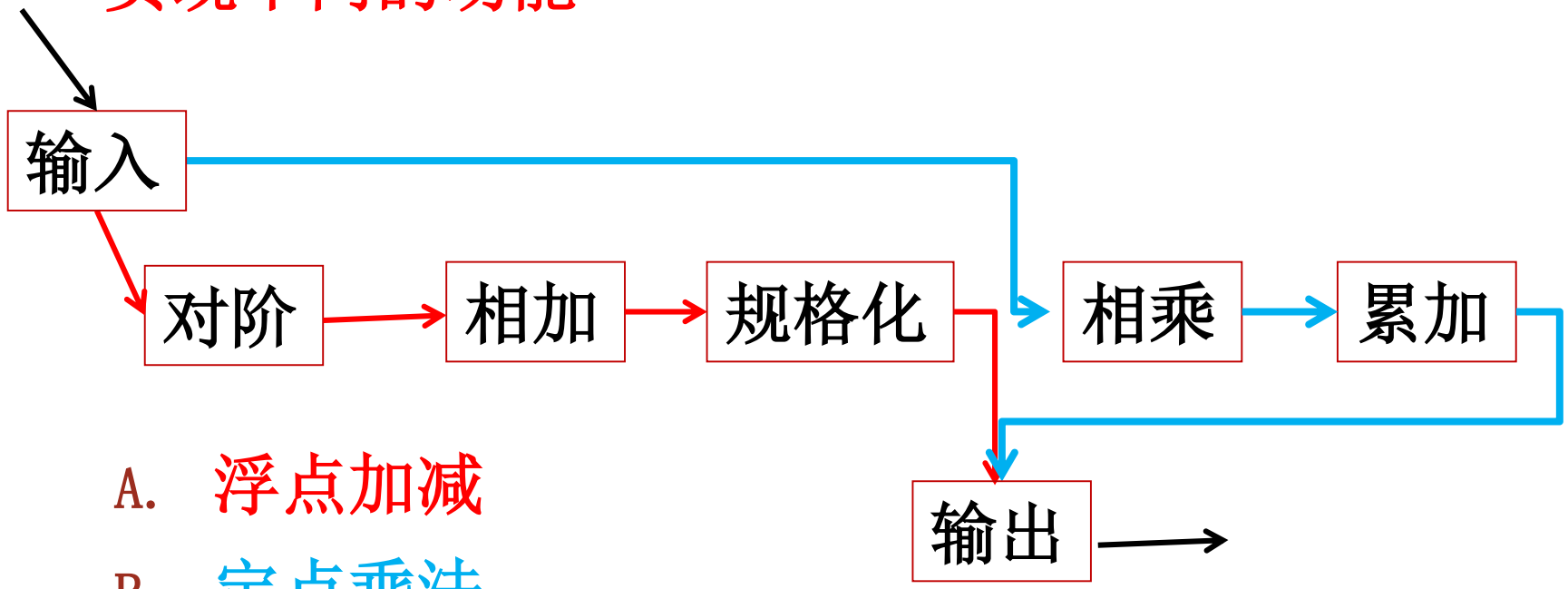
③ **系统级流水**：构成计算机系统的多个处理机之间的流水，也称为宏流水。

2、按功能分类

① 单功能流水线：只能完成一种固定功能。

要完成多个功能，可将多个单功能流水线组合。

② 多功能流水线：流水线的各段通过不同的连接实现不同的功能



A. 浮点加减

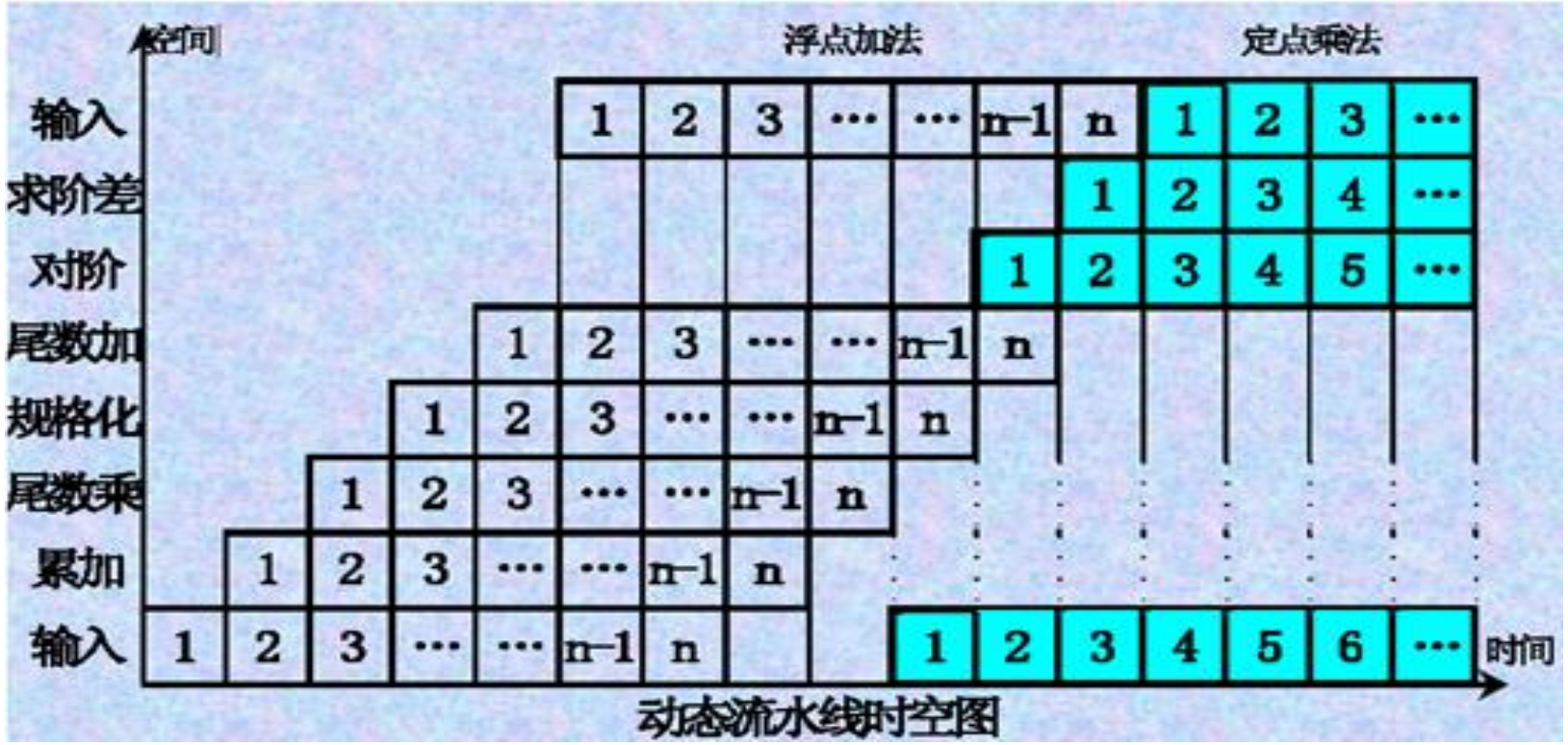
B. 定点乘法

3、按工作方式分类

① **静态流水线：**某一时间内各段只能按一种功能连接流水，只有等流水线全部流空后，才能切换成按另一种功能连接流水。

只有连续出现同一种运算时，流水线的效率才能得到充分的发挥。

② 动态流水线：允许在同一时间内各段按不同运算或功能连接。



动态流水线时空图

显然，动态流水线必是多功能流水线，而单功能流水线必是静态的。

4、按所具有的数据表示分类

① 标量流水处理机：没有向量数据表示，只能用标量循环方式来对向量、数组进行处理

② 向量流水处理机：具有向量数据表示，设置有向量指令和向量运算硬件，能对向量、数组中的各个元素流水地处理。是向量处理机和流水技术的结合

5、按所各段之间是否有反馈回路分类

① 线性流水线：各段串行连接，各段只流过一次，没有反馈回路。

② 非线性流水线：某些功能段有反馈回路，可能多次经过某个段。

3.2.2 标量流水线的主要性能

1、吞吐率

- 吞吐率 ：流水线单位时间内流出的任务数或结果数。
- 最大吞吐率：流水线正常满负荷工作时，单位时间内流出的最大结果数。
- 实际吞吐率：从启动流水线开始到流水线操作结束，单位时间内能流出的任务数或结果数。

思考：最大吞吐率和实际吞吐率的关系

- 计算吞吐率的最基本公式：$$Tp = \frac{n}{T_k}$$

思考：试分析如下情况时的 Tp Tp_{max}

n：任务数

T_k：完成n个任务所用的时间

k为流水线的段数

Δt_i为每个流水段所需时间。

① 各段执行时间相等，输入连续任务情况下，完成n个连续任务需要的总时间为：

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

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

最大吞吐率为：

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

② 若各段执行时间不相等，输入连续任务情况下，且各个子过程所需时间分别为Δt_1、Δt_2、…、Δt_k，时钟周期应为max{Δt_1、Δt_2、…、Δt_k}

- 吞吐率为：

$$Tp = \frac{n}{\sum_{i=1}^{k} \Delta t_i + (n-1)\max\{\Delta t_1, \Delta t_2, \Delta t_3, \cdots, \Delta t_k\}}$$

最大吞吐率为：

$$Tp_{max} = \frac{1}{\max\{\Delta t_1, \Delta t_2, \Delta t_3, \cdots, \Delta t_k\}}$$

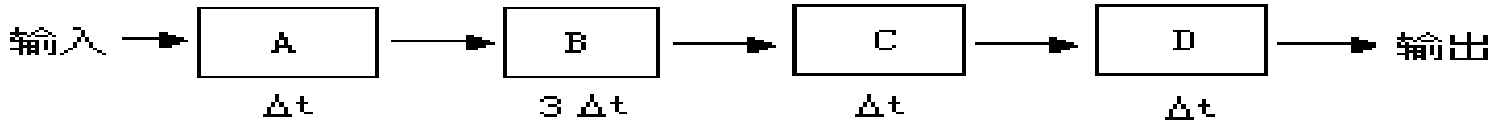
- 流水线的最大吞吐率是取决于子过程所经过的时间Δt。当各功能段时间不同时，最大吞吐率取决于最慢子过程所需时间。最慢子过程称为“瓶颈”子过程
- 实际吞吐率总是低于最大吞吐率，因为流水线从开始启动到流出第一个结果，需要经过建立阶段和排空阶段

$$Tp = \frac{n}{(k+n-1)\Delta t} = \frac{n}{(k+n-1)} \times Tp_{max}$$

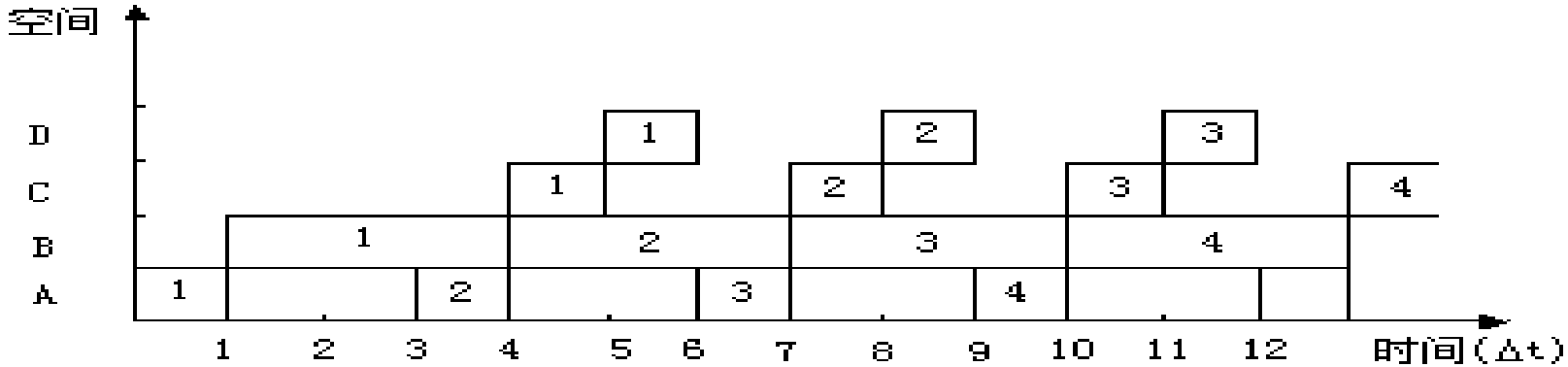
$$= Tp_{max} / (1 + \frac{k-1}{n})$$

当 k « n 时 TP ≈ TP max

- 例：有一个4段的指令流水线，其中A、C、D段的时间为Δt，B段的时间为3Δt，则B段就是该指令流水线的瓶颈。根据流水线的最大吞吐率公式，其最大吞吐率TPmax=1/(3Δt)，即流水线满负荷工作时，只能每隔3Δt才解释完一条指令，每隔3Δt才能流出一个结果



(a) 指令的流水解释

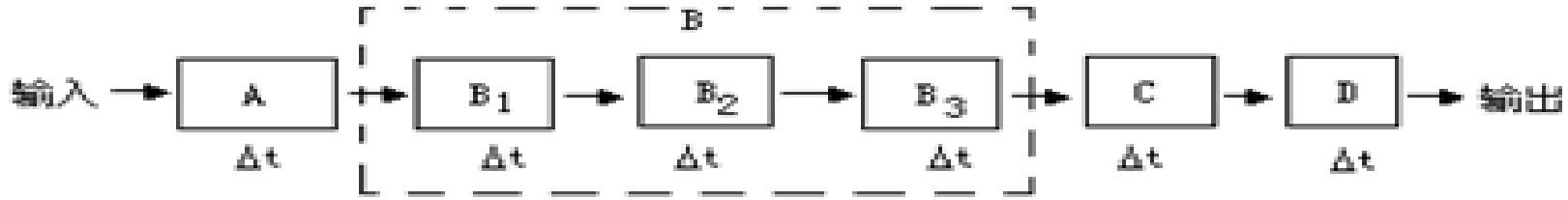


(b) 流水处理过程

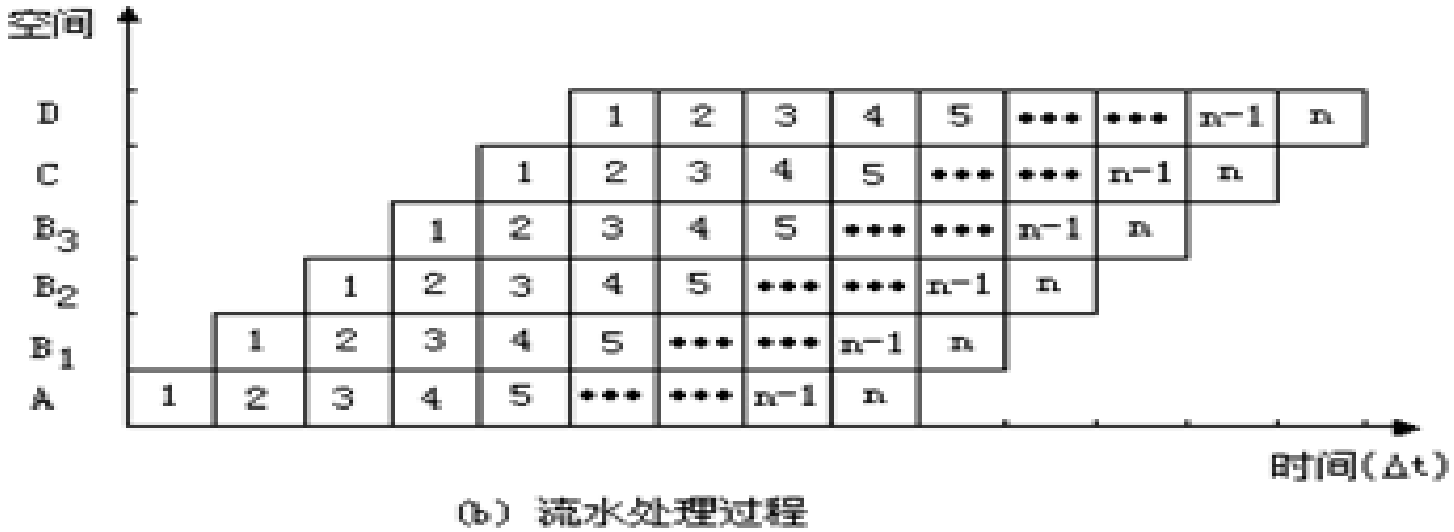
显然，要提高流水线的最大吞吐率，应当设法消除瓶颈子过程。

- 消除办法1：将瓶颈子过程再细分。

如：将B段再细分为三个段B_1、B_2和B_3，该指令流水线的最大吞吐率就变成1/Δt。



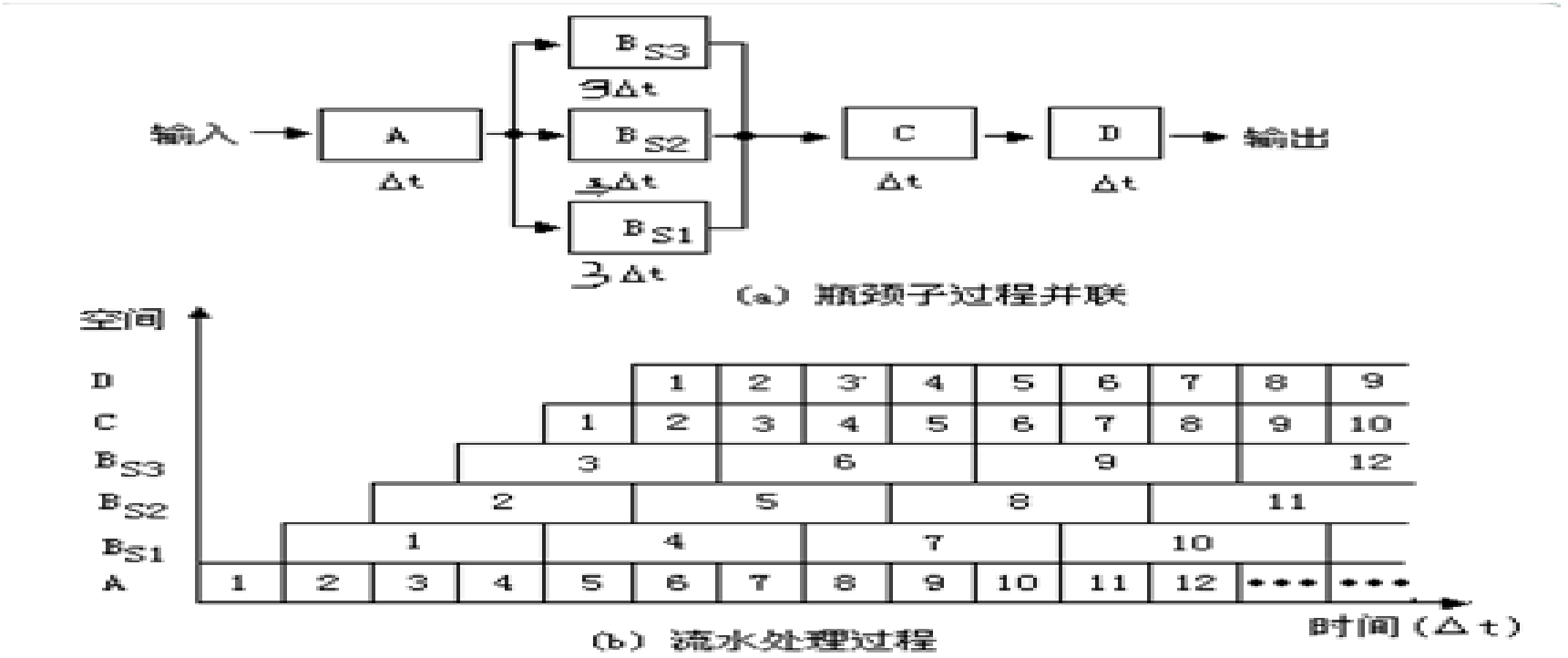
(a) 将B段再细分为3个子段



(b) 流水处理过程

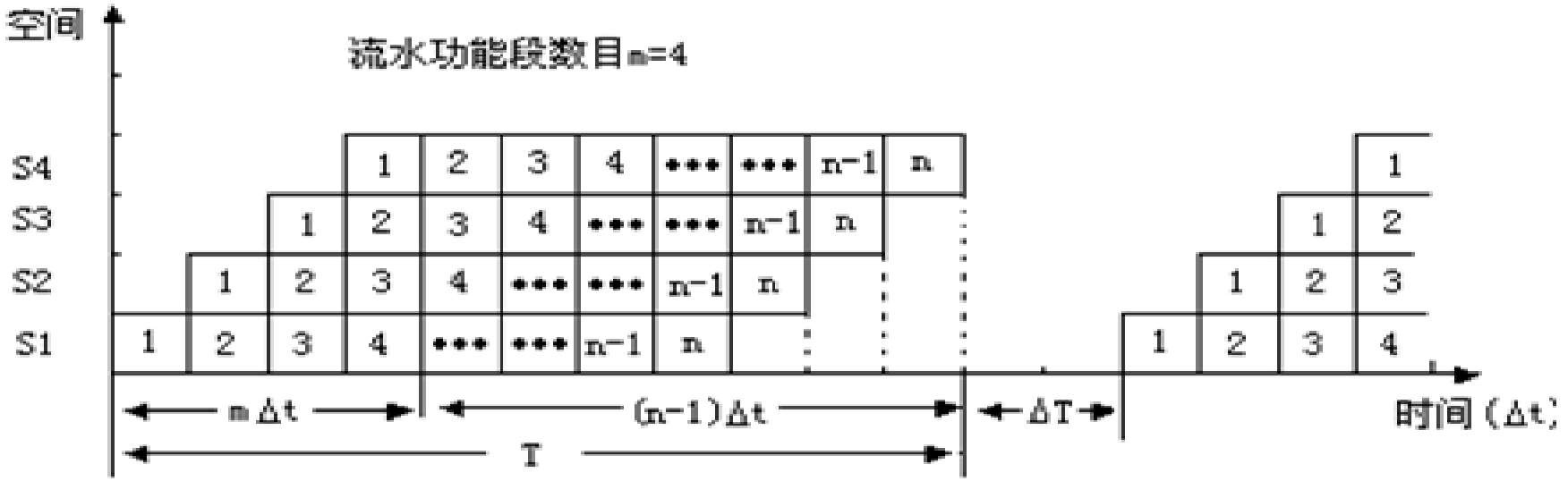
消除办法2：瓶颈子过程重复设置

如：可采用三套B段并联，每隔Δt轮流给一个瓶颈段分配任务，使它们仍可每隔Δt解释完一条指令。其流水线的最大吞吐率也是1/Δt。



- 如果流入流水线的是周期性的任务，即每流入n条指令之后，延迟ΔT的时间，再流入n条指令，延迟ΔT之后，再流入n条指令，周而复始，则这时的流水线的实际吞吐率TP就应为：

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



2、加速比

- 加速比(Speedup ratio)：流水线工作相对于顺序串行工作方式，速度提高的比值。
- 设流水线由k段组成，共完成n条指令

① 若各功能段时间相等，均为 Δt 则流水线的加速比为：

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

② 若各功能段时间不相等，记为 Δt_i 则流水线的加速比为：

$$S_p = \frac{n\sum_{i=1}^{k} \Delta t_i}{\sum_{i=1}^{k} \Delta t_i + (n-1)\Delta t_{\max}}$$

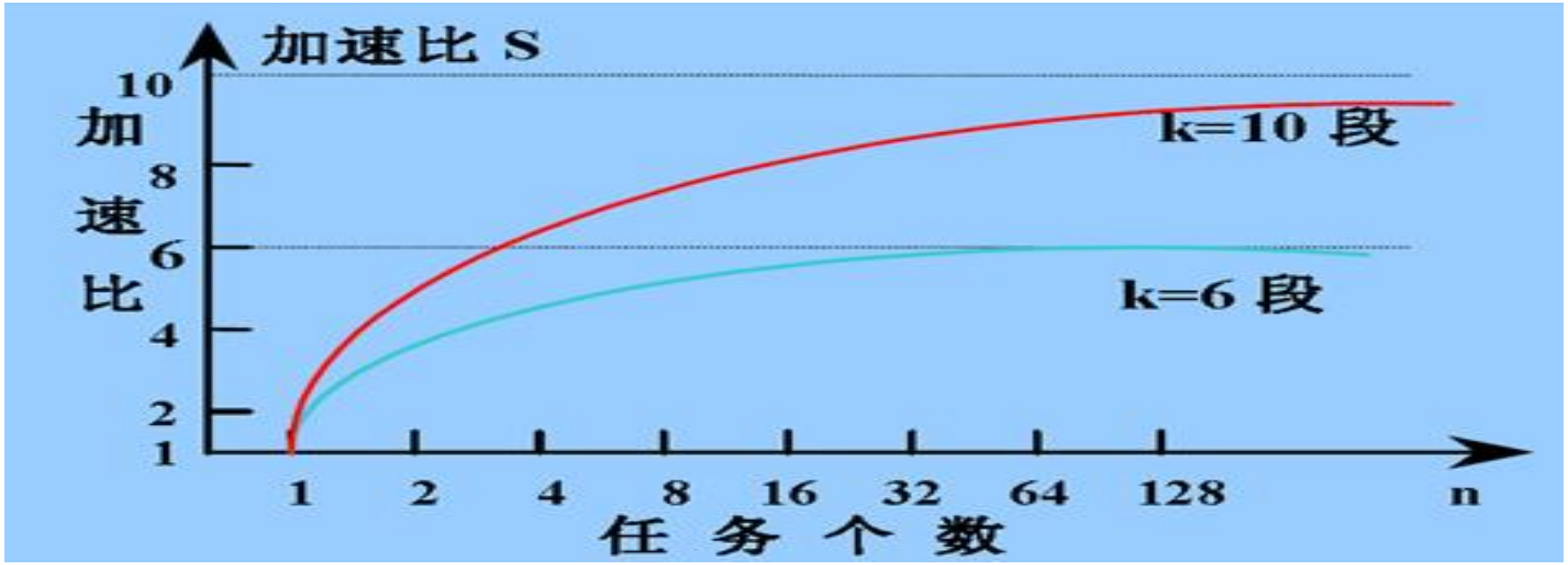
➢ 当各功能段时间相同时，最大加速比为：

$$S_{p\max} = \lim_{n\to\infty} \frac{k \cdot n}{k + n - 1} = k$$

可以看出，当n>>k时，流水线的加速比S_p才接近于流水线的段数k。因此，在n>>k的前提下，增大流水线的段数k，可以提高流水线的加速比S_p。

？问题：是否增大流水线段数k都能够提高流水线的加速比?

- 实际应用中，从软、硬两方面入手，保证在流水线中连续流动的任务数n能远远大于子过程数k，才能充分发挥流水线的效率。
  - 极端情况下，即n=1的时候，由于k的增大，锁存器数也增多，实际上增加了任务从流入到流出流水线时间，以至于其速度反而比顺序串行的还要低。

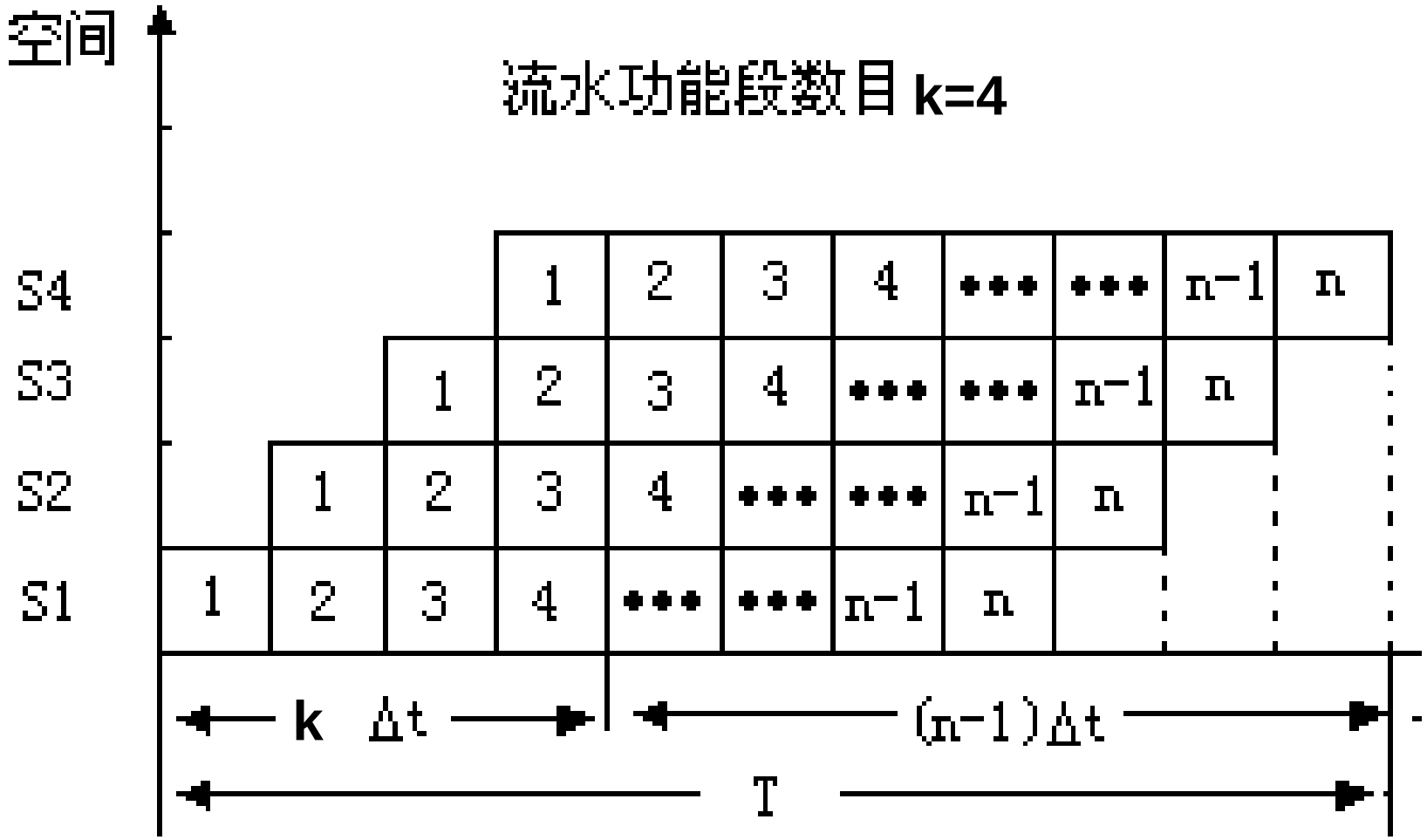


3、效率（　　，Efficiency ）

- 指流水线的设备利用率。在整个运行时间里，有多少时间流水线设备真正用于工作。是实际使用时间占整个运行时间之比
- 由于流水线有建立和排空时间，在连续完成n个任务的时间里，各段并不是满负荷工作的，因此流水线的效率一定小于1

- 如果是k段线性流水线，且各段经过时间相同，则在T时间里，流水线各段的效率都相同，均为 η_0，即

$$\eta_1 = \eta_2 = \cdots = \eta_k = \frac{n \cdot \Delta t_0}{T} = \frac{n}{k + n - 1} = \eta_0$$



$$\eta_1 = \eta_2 = \cdots = \eta_k = \frac{n}{k + n - 1} = \eta_0$$

整个流水线的效率

$$\eta = \frac{\eta_1 + \eta_2 + \cdots + \eta_k}{k} = \frac{k \cdot \eta_0}{k} = \frac{k \cdot n\Delta t_0}{k \cdot T}$$

- 几何解释：从时空图上看，效率就是

$$\eta = \frac{\text{n个任务的时空区面积}}{\text{k个段的总时空面积)}}$$

- 与吞吐率类似，只有当n>>k时， η 才趋近于1。
- 对于线性流水且每段经过时间相等时，流水线的效率正比于吞吐率， 即

$$\eta = \frac{n \cdot \Delta t_0}{T} = \frac{n}{n + (k-1)} = TP \cdot \Delta t_0$$

- 如果各段经过的时间不相等，其中“瓶颈”段时间为Δt_j，完成n条指令的解释。整个流水线的效率为：

$$\eta = \frac{n\text{个任务占用的时空区}}{k\text{个段总时空区}}$$

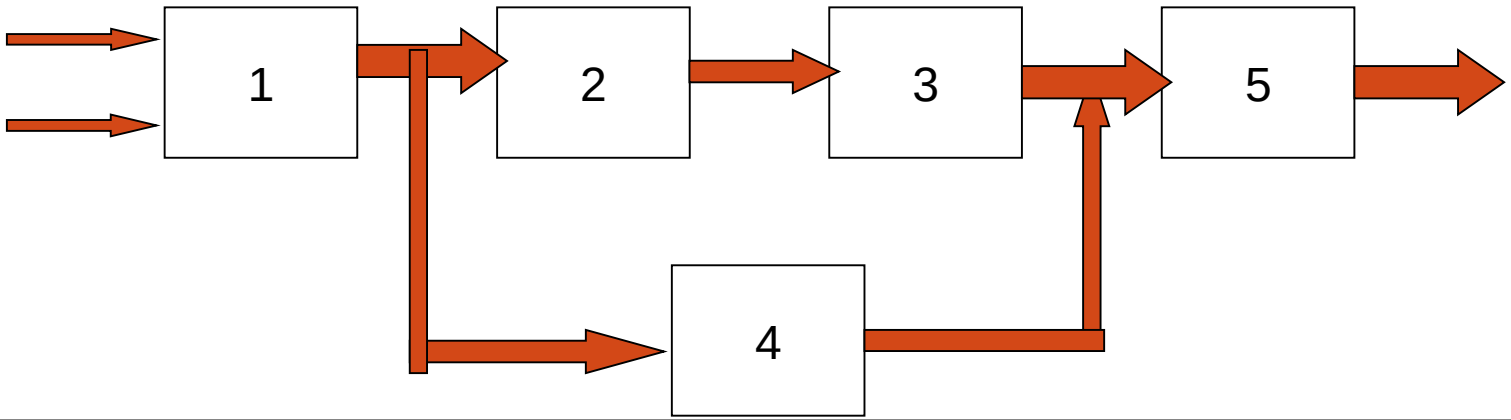
$$= \frac{n \cdot \sum_{i=1}^{k} \Delta t_i}{k \cdot \left[\sum_{i=1}^{k} \Delta t_i + (n-1)\Delta t_j \right]}$$

◆流水线工作举例1

设向量A、B各有4个元素，要在如图所示的静态双功能流水线上，计算向量点积$A \cdot B=\sum a_i \times b_i$

- S1→S2→S3→S5组成加法流水线
- S1→S4→S5组成乘法流水线

设每个流水段的时间为Δt，其延迟时间和功能切换时间忽略不计。使用合理的算法，能使完成向量点积A·B运算所用的时间最短，求出流水线在此期间**T、TP、SP、η**的值。



根据题目的要求，解题过程可分成四步：

① 把算术表达式展开

S = a1.b1 + a2.b2 + a3.b3 + a4.b4

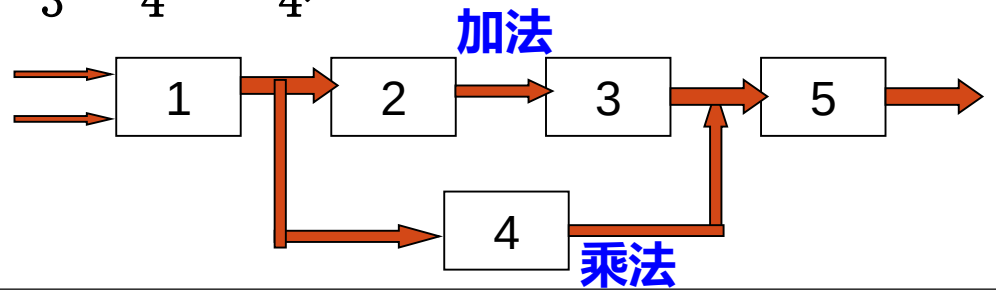
② 选择适合于静态流水线工作的算法

(1)连续计算$a_1 \times b_1$、$a_2 \times b_2$、$a_3 \times b_3$、$a_4 \times b_4$4个乘法；

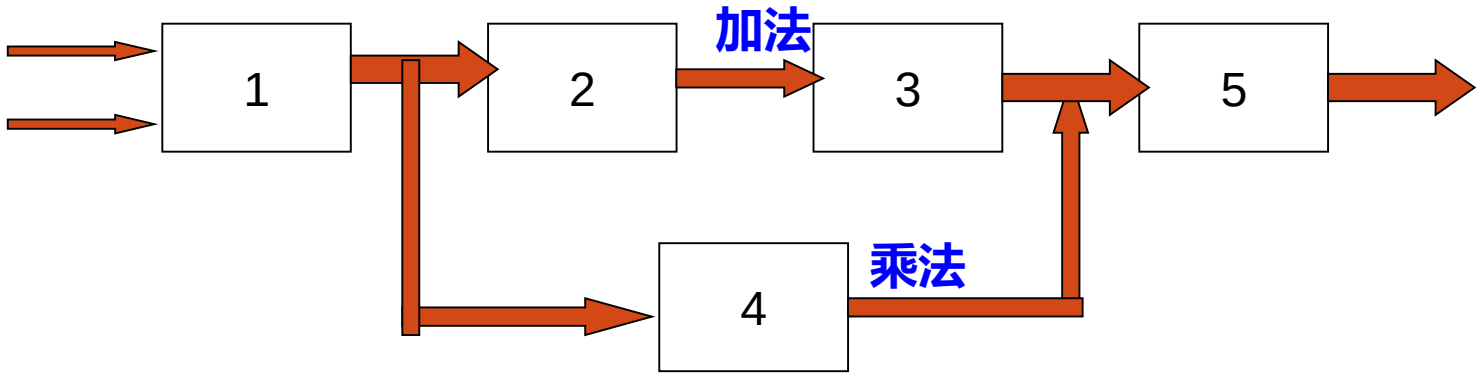
(2)等乘法流水线排空后切换功能，按加法方式联接功能部件，连续计算$(a_1 \times b_1 + a_2 \times b_2)$、$(a_3 \times b_3 + a_4 \times b_4)$；

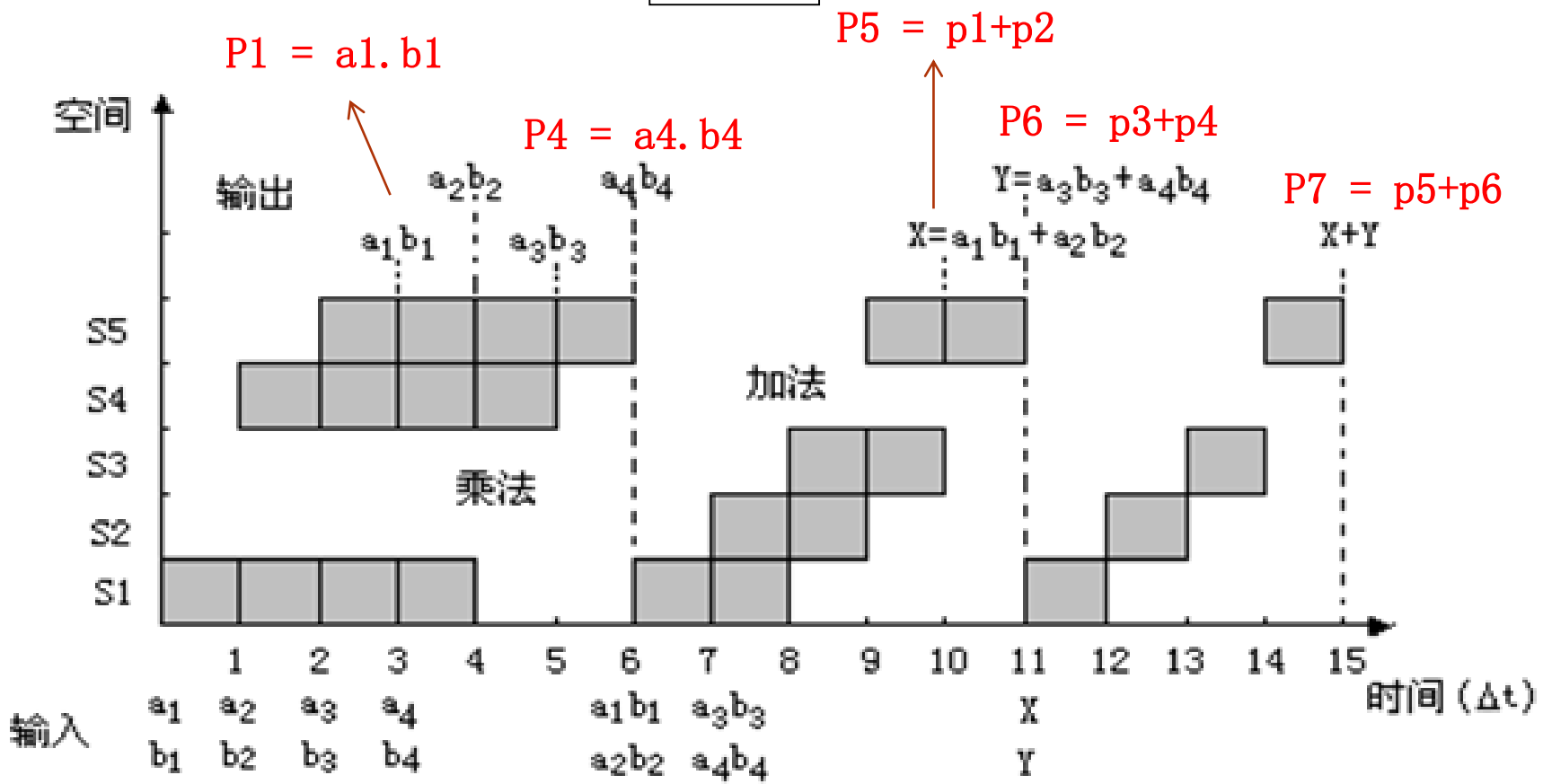
(3)产生了上述两个结果后，再最后计算：

$(a_1 \times b_1 + a_2 \times b_2) + (a_3 \times b_3 + a_4 \times b_4)$。



③ 画出时空图


1
2
加法
3
5
4
乘法


P1 = a1.b1
P4 = a4.b4
P5 = p1+p2
P6 = p3+p4
P7 = p5+p6
空间
输出
a_2b_2
a_4b_4
$Y=a_3b_3+a_4b_4$
a_1b_1
a_3b_3
$X=a_1b_1+a_2b_2$
X+Y
S5
S4
S3
S2
S1
加法
乘法
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
时间（Δt）
输入
a_1 a_2 a_3 a_4
b_1 b_2 b_3 b_4
a_1b_1 a_3b_3
a_2b_2 a_4b_4
X
Y

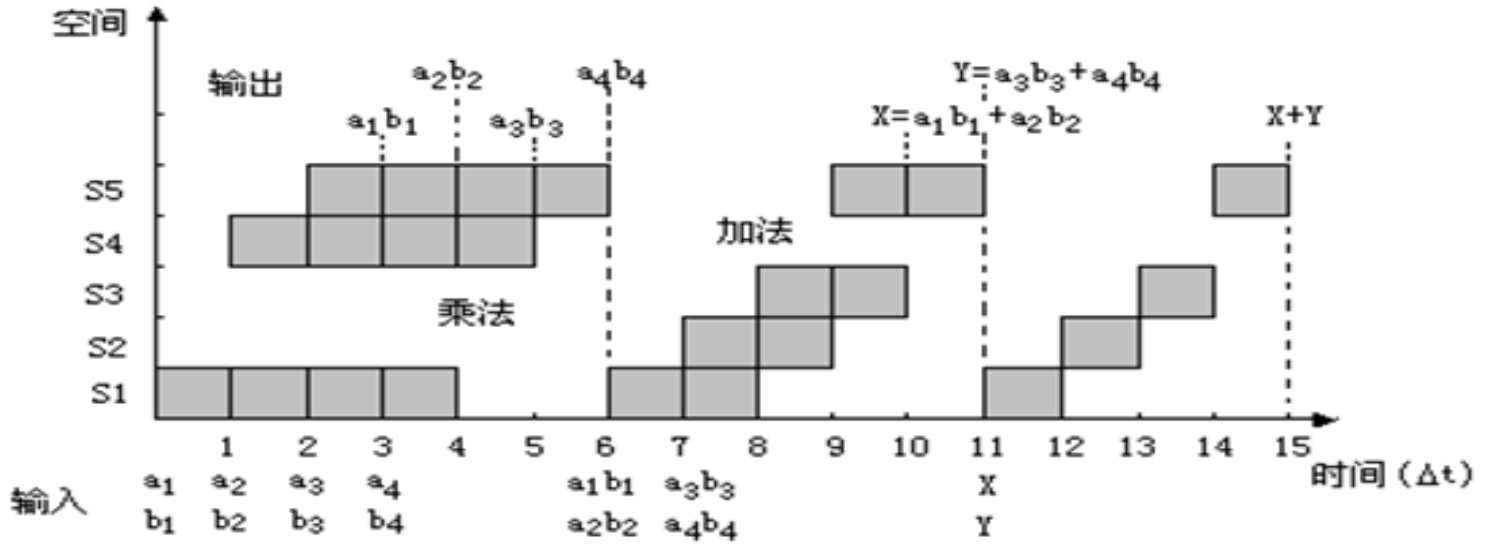
④ 计算

- 流入流水线的任务数为7，完成任务所需的时间为15Δt，而顺序完成7个任务需要的时间为
  4×3Δt+3×4Δt=24Δt
- 因此，该流水线的实际吞吐率为：

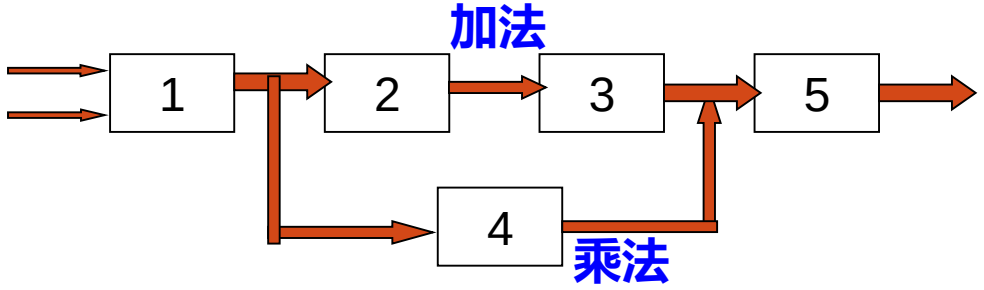
  $TP = 7/(15\Delta t)$

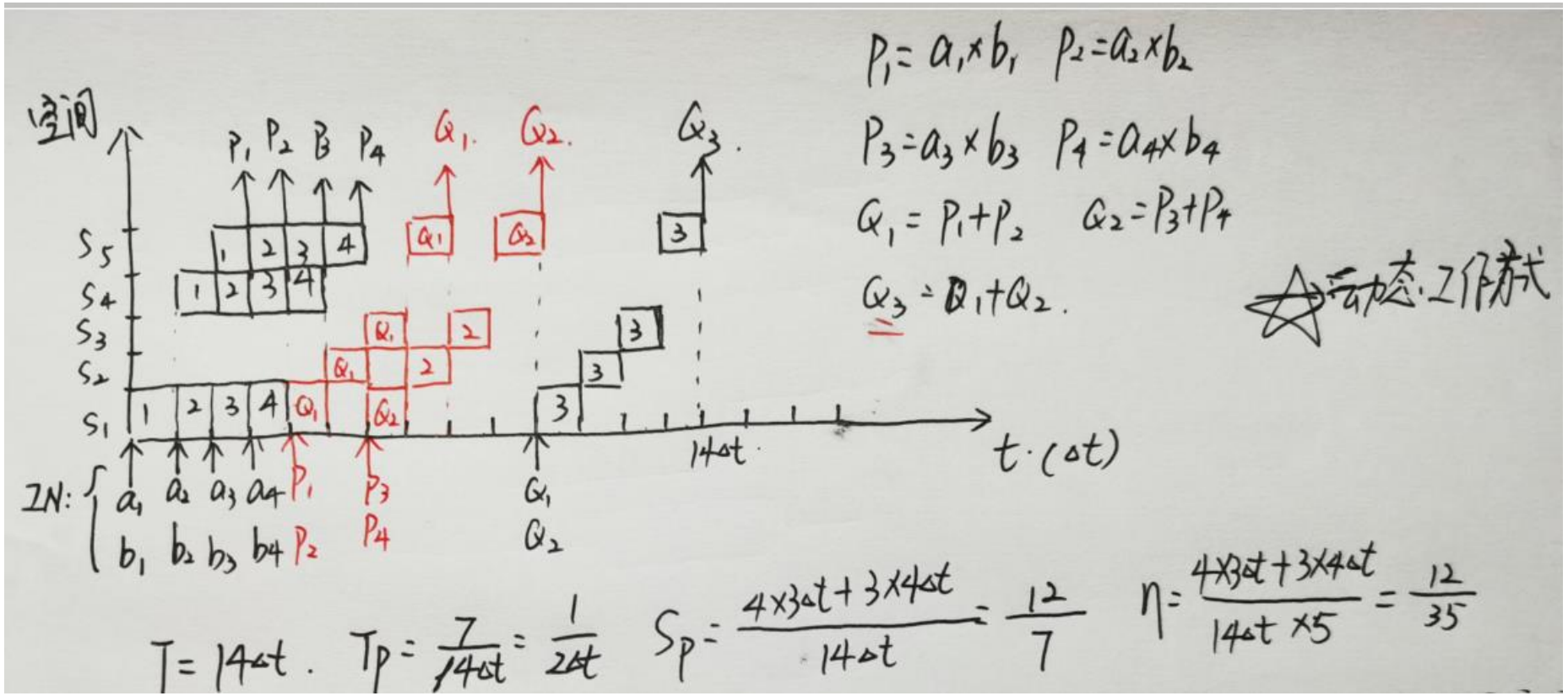
  $Sp = (3\times4\Delta t+4\times3\Delta t)/15\Delta t = 1.6$
- 效率 $\eta = (3\times4\Delta t+4\times3\Delta t)/(5\times15\Delta t)=32\%$

解题速度提升为原来的1.6倍，但效率不到1/3!



思考：若题目中其余条件不变，流水线采用动态工作方式，该如何完成计算向量点积$A \cdot B=\sum a_i \times b_i$？ 并求流水线在此期间T、TP、SP、η的值。





◆流水线工作举例2

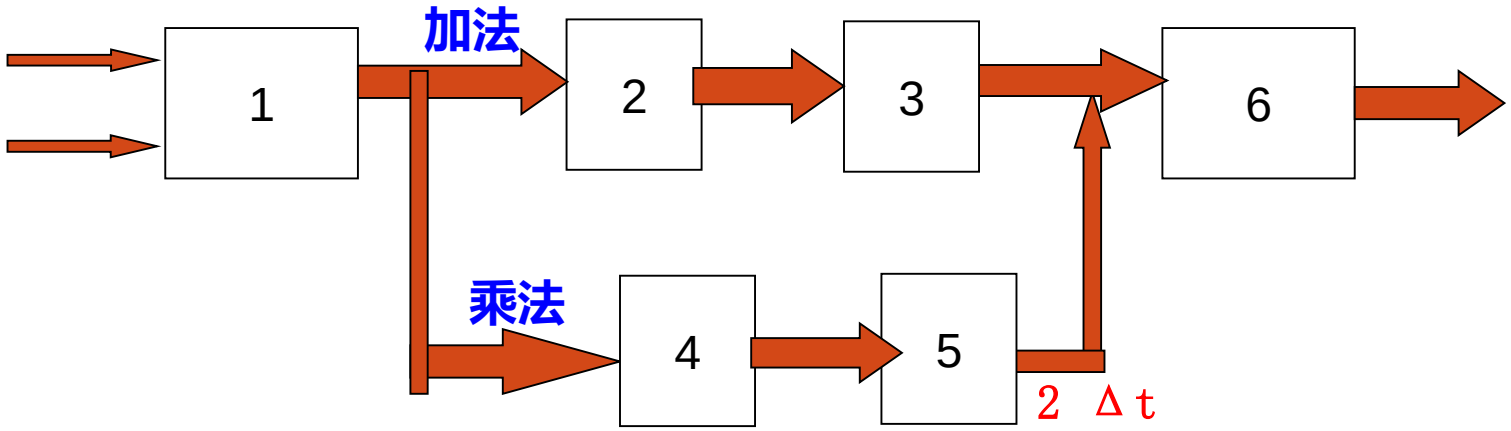
有一条双功能双输入的动态流水线，由6个功能段组成：

➢S1→S2→S3→S6连接完成加法；S1→S4→S5→S6连接完成乘法

设经过S5的时间为2 Δt，其余各段时间都为Δt，延迟时间和功能切换时间忽略不计。现要完成如下计算：

E=A *（B*C+D*（E+F*G））

试使用合理的算法，使完成运算所用的时间最短，并求出流水线在此期间TP、SP、η的值。



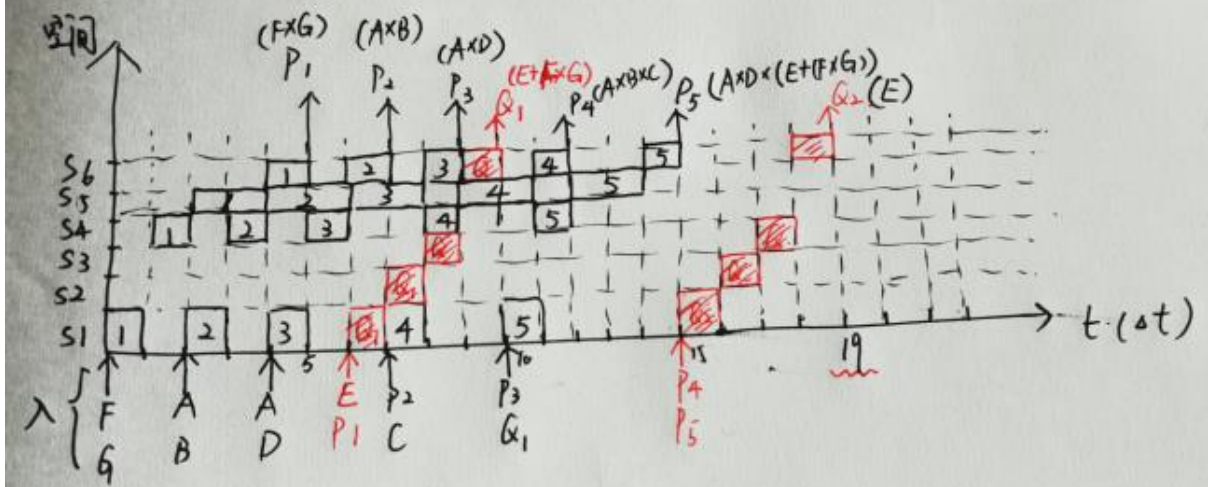
例2 (动态流水线分析)

$E = A\times(B\times C + D\times(E+F\times G))$

$= A\times B\times C + A\times D\times(E+F\times G))$

$P_1 = F\times G \quad P_2 = A\times B \quad P_3 = A\times D \quad Q_1 = E + P_1 \quad P_4 = P_2\times C$

$P_5 = P_3\times Q_1$ 和 $Q_2 = P_4 + P_5 = E$ (3次'×', 2次'+')



$T = 19\Delta t$

$Tp = \frac{n}{T} = \frac{5+2}{19\Delta t} = \frac{7}{19\Delta t}$,

$Sp = \frac{T_1}{T_p} = \frac{5\times5\Delta t + 2\times4\Delta t}{19\Delta t} = \frac{33}{19}$

$\eta = \frac{5\times5\Delta t + 2\times4\Delta t}{19\Delta t \times 6} = \frac{11}{38}$

◆流水线工作举例3

有一个双输入端乘—加双功能动态流水线，“乘”由1→2→3→4完成，“加”由1→5→4完成，各段延时均为Δt，输出可直接返回输入或存入缓冲器缓冲。若用流水线按最快的处理方式计算长度均为8的A、B两个向量逐对元素求和的连乘积

$$S=\prod_{i=1}^{8}(A_i+B_i)$$

(1) 画出流水线完成此运算的时空图；

(2) 完成全部运算所需多少Δt?此期间流水线的效率是多少?

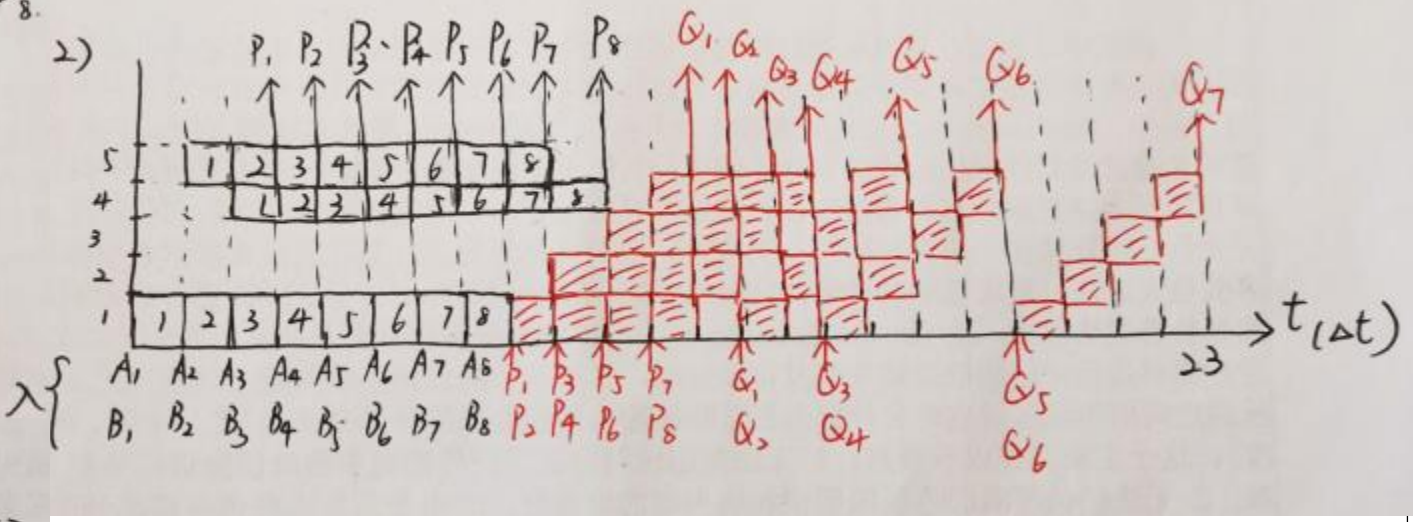
(1) $P_1 \sim P_8$: A_i+B_i $(i=1,2,\cdots 8)$

$Q_1=P_1*P_2$ $Q_2=P_3*P_4$ $Q_3=P_5\times P_6$ $Q_4=P_7\times P_8$

$Q_5=Q_1\times Q_2$ $Q_6=Q_3\times Q_4$ $Q_7=Q_5\times Q_6$

(共计8个'+', 7个'×')

2)



(3) $T=23\Delta t$

$T_P=\frac{n}{T}=\frac{15}{23\Delta t}$ $S_P=\frac{8\times 3\Delta t+7\times 4\Delta t}{23\Delta t}=\frac{52\Delta t}{23\Delta t}=\frac{52}{23}$

$\eta=\frac{8\times 3\Delta t+7\times 4\Delta t}{23\Delta t\times 5}=\frac{52}{115}$

试着用不同算法分析，是否更快？

流水线分析小结：

- 性能指标的计算
  - 确定算法
  - 画出流水线的时空图
  - 计算T，进而计算各个指标
- 一个任务能否送入流水线，取决于
  - 是否所需要的操作数都准备好？
  - 是否引起资源冲突？
  - 是否满足工作方式的要求？

流水线效率低的原因：

- 多功能流水线，未用到的部件处于空闲
- 建立与排空时，部分设备空闲；静态工作方式中的功能切换，增加了建立与排空的时间
- 送入流水线任务数量太少
- 送入流水线的任务间存在相关问题

流水线最适合处理

同一操作类型，且输入输出间不存在相关性的一连串运算

3.2.3 标量流水机的相关处理

流水线只有连续不断地流动，即不出现“断流”，才能获得高效率。然而断流不可避免。

- 造成 “断流”的原因：

① 编译形成的目标程序不能发挥流水结构的作用

② 存储系统供不上为连续流动所需的指令和操作数

③ 转移指令的使用

④ 相关和中断的出现

流水同时解释多条指令，相关状况要比重叠机器更复杂、更严重。

指令的相互联系、转移、中断等操作，影响流水线处理机的正常工作，影响了速度，也就是产生“相关”。

- 相关的类型：

- 局部性相关：指令相关、主存数相关和寄存器组数相关由于只影响相关的二条或几条指令，而至多影响流水线某些段的推后工作，并不会改动指令缓冲器中预取到的指令内容，影响是局部的，所以被称之为局部性相关，也称数据相关。

- 全局性相关：转移指令可能会造成流水线中很多已被解释的指令作废，需要重新预取指令进入指令缓冲寄存器，它将影响整个程序的执行顺序，所以称之为全局性相关，也称转移相关。

1、局部性相关的处理

局部性相关包括指令相关、主存数相关和寄存器组数相关。

- 重叠机器的两种处理办法可以沿用：
  - 推后对相关单元的读，直至写入完成；
  - 设置相关专用通路，使得不必先把运算结果写入相关存储单元，再读出后才能使用，而是经相关专用通路直接使用运算结果

送入流水线的指令

h: R0->R1;

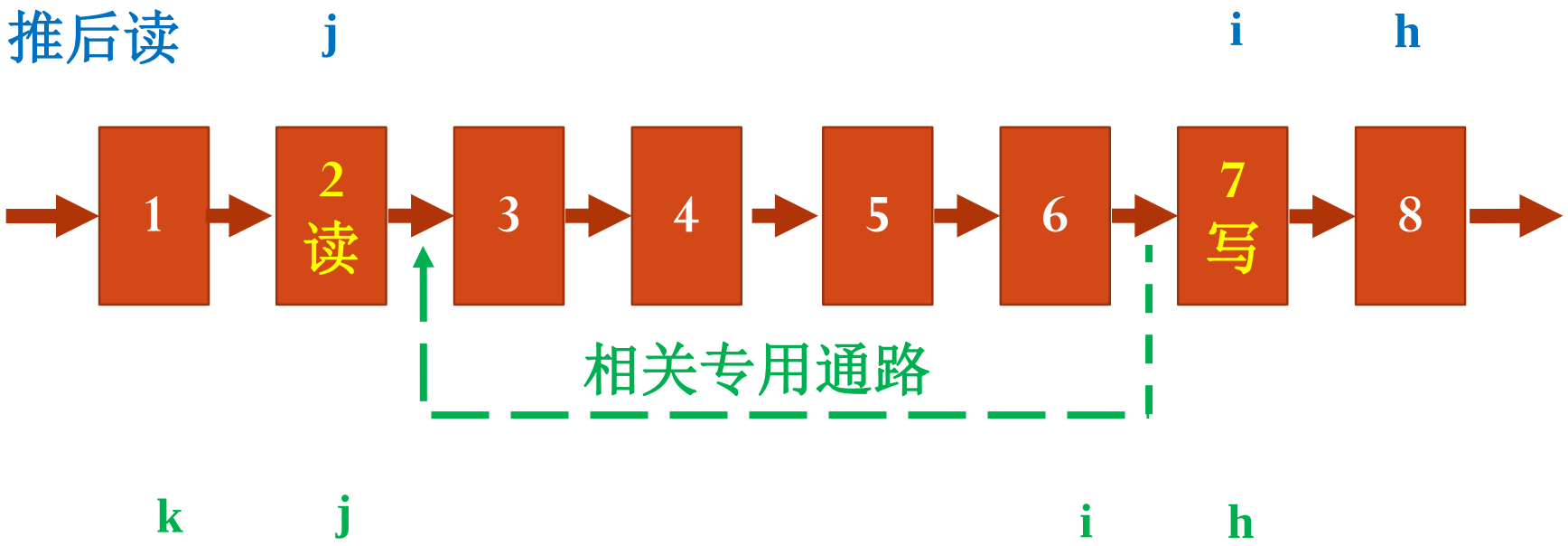
i: R3->R4;

j: R1+R2 ->R2;

k: 2->R4;

l: 2*R0->R3

m: R4+1->R0



特点：1、指令进入流水线的顺序和流出顺序相同；

2、一定程度上解决了“写后读”相关

3、吞吐率、效率仍然低

调整流水线的流动方式

① 顺序/同步流动方式

- 任务流出流水线的顺序与流入顺序一致。
  - 如：一串指令“h, i, j, k, l, m, n, …”，h和j发生数相关，j流到读段时，j及其后的指令必须停下来, 直到h到达写段并完成写入。
  - 实质：推后读
  - 控制简单，但相关后流水线的吞吐率和效率都要下降

顺序流动限制了效率的进一步提高，如何改进？

送入流水线的指令

h: R0->R1;

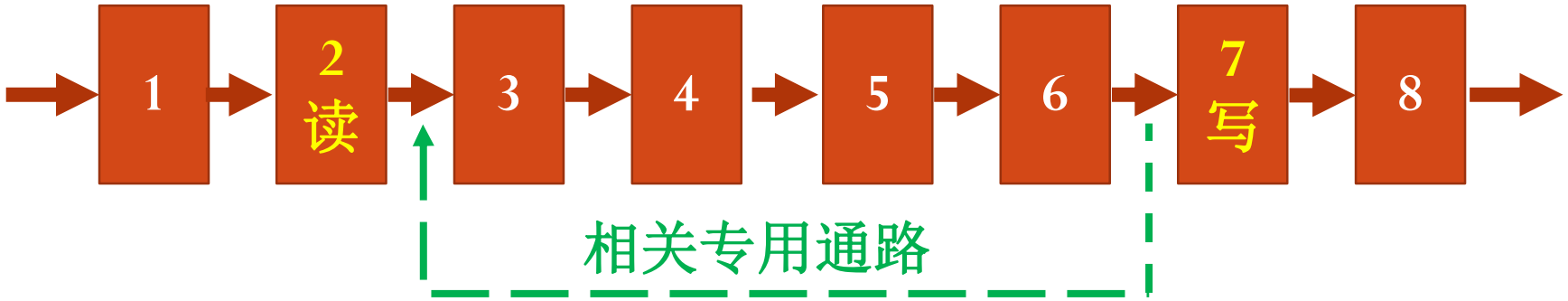
i: R3->R4;

j: R1+R2 ->R2;

k: 2->R4;

l: 2*R0->R3

m: R4+1->R0



k j i h 顺序流动

j m l k i h 不顺序流动

可行？ j i l m k h

不顺序流动特点：1、指令进入流水线的顺序和流出顺序不相同；

2、保证流水线不“断流”

3、处理好可能引起的新相关“写-写”、“读-写”

② 异步流动方式

- 任务流出流水线的顺序与流入顺序不同。
  - 如上例：j之后与j不相关的指令可越过j继续流动。
  - 流水线的吞吐率和效率都未下降。

  - 解决好可能会发生新的相关
    - “写-写”相关，如：i，k指令都要写入同一单元，避免最终出现k先写入而i后写入。
      - 控制机构必须保证发生“写-写”相关后，写入的顺序不变。
    - “先读后写”相关，如：避免最终出现l的读操作落后于m的写操作。

解决局部性相关，应在控制机构上需解决好如下问题：

- 如何判定流水线的多条指令之间是否相关
- 如何控制推后读
- 如何设置相关专用通路并控制其连通和断开
- 如何协调好异步流动时的三种相关

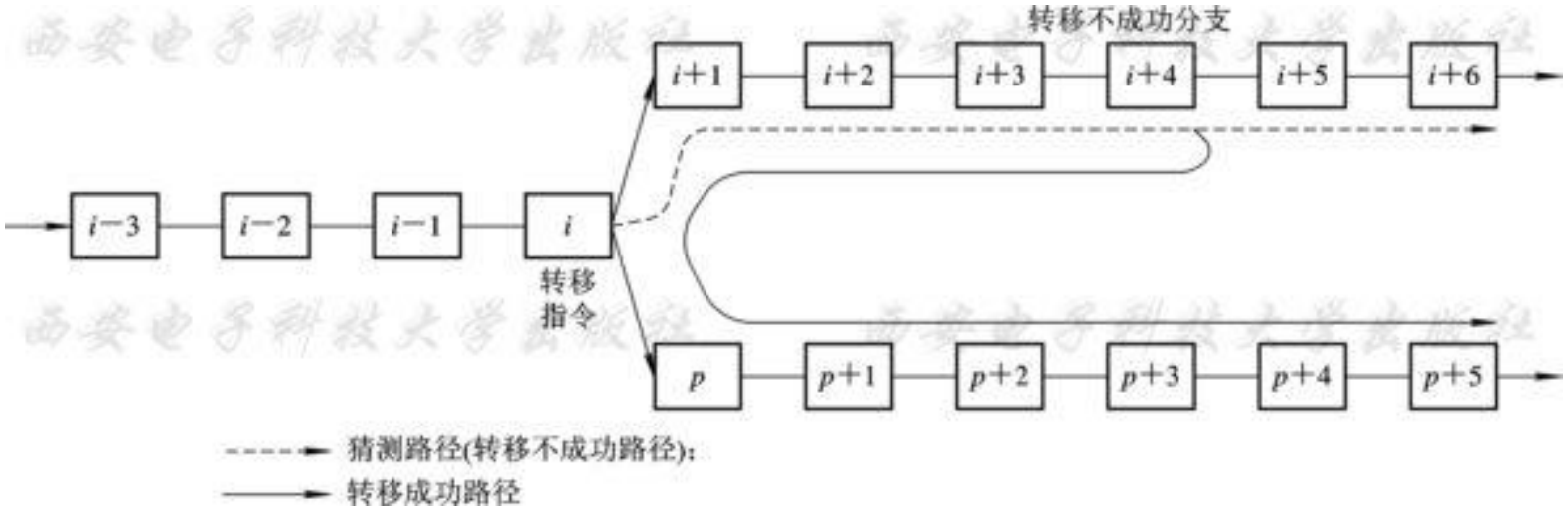
* 采用总线式分布式控制管理，进一步优化处理标量流水处理机的局部相关问题

2、全局性相关的处理

① 猜测法

猜测法又称分支预测技术，猜取第i+1条指令或第p条指令所在分支继续向前流动。

- 猜测原则：
  - 猜概率高者
  - 两者概率相近时，宜选不成功转移分支，因为它已预取进指缓。



猜测时应保证在猜错时可恢复分支点原来的现场。

- 恢复现场的三种方法：
  - 对猜测指令的解释只完成译码和准备好操作数，在转移条件码出现前不执行运算；
  - 对猜测指令的解释可完成到运算完毕，但不送回运算结果；

这两种办法不方便，因为若猜对后还要让这些指令继续完成余留的操作。

  - 对猜测指令不加区别地全部解释完，但需把可能被破坏的原始状态都用后援寄存器保存起来，一旦猜错就取出后援寄存器的内容来恢复分支点的现场。

采用后援寄存器法比前两种方法的实现效率会更高一些

② 加快和提前形成条件码

猜测法是为了提前取得指令码，执行相应指令。而加快和提前形成条件码，是先取得状态的标志。

- 加快单条指令内部条件码的形成

  - 特别适合于转移条件码是由上一条运算型指令产生的情形。
  - 由于一条运算型指令在其执行完毕之前，就能形成反映运算结果的部分条件码，如零标志、符号标志等，因此可以在取得操作数之后、开始运算之前提前形成条件码。

- 在一段程序内提前形成条件码

这特别适合于循环程序在判断循环是否结束时的转移情况。

在循环转移中，多以 CX - 1 ，CX ≠ 0 作为循环的条件。

CX ≠0　　时，转移到循环的入口地址，

CX = 0　　时，跳出循环体。

这样 就有减 1和 判别两个操作，如果把操作提前进行就能够提前形成条件码，可以加快循环的执行。

③ 延迟转移技术

- 用软件方法将转移指令与其前面不相关的指令交换位置

④ 加快短循环程序的处理

- 将长度小于指缓的循环程序一次性放入指缓，并暂停预取指令
- 或循环出口端的条件转移指令恒猜循环分支。

4、中断的处理

中断会引起流水线断流。但出现概率比条件转移的概率要低得多，且是随机发生的。

- 流水机器处理中断的主要目的是：

  如何处理好断点现场的保存和恢复，而不是如何缩短流水线的断流时间。

现场包括两个方面：

① 提供给中断服务子程序准确的断点现场；

② 在中断处理完后恢复到断点处指令流水线的指令现场。

准确的断点现场：若在执行完第i条指令时响应中断请求，送给中断处理程序的就是对应于第i条指令的中断现场，如第i条指令的程序状态字等。

中断的处理

① “不精确中断”法： 断点不精确

不论指令*i*在流水线的哪一段发生中断，未进入流水线的后续指令不再进入，已在流水线的指令仍继续流完，然后才转入中断处理程序。这样，断点就不一定是*i*，可能是*i*+1、*i*+2或*i*+3、…，即断点是不精确的。

② “精确中断”法：断点精确。

不论指令i是在流水线中哪一段响应中断，给中断处理程序的现场全都是对应i的，i之后流入流水线的指令的原有现场都能保存和恢复。 需设置很多后援寄存器，以保证流水线内各条指令的原有现场都能保存和恢复。

3.2.4 非线性流水线调度技术

主要内容：

- 概述
- 单功能非线性流水线的调度
- 多功能非线性流水线的调度（扩展）

1、概述

- 启动距离：向一条流水线的输入端连续输入两个任务之间的时间间隔。
- 冲突：当以某一个启动距离向一条非线性流水线连续输入任务时，可能在某一个功能段，或某几个功能段中发生有几个任务同时争用同一个功能段的情况

线性流水线：功能段线性地逐级串接在一起，每个任务在各段只通过一次，因此每拍都可将一个新的任务送入流水线，这些任务不会争用同一个流水线。

问题的提出：非线性流水线由于段间设置有反馈回路，会出现多个任务争用同一功能段的冲突。

- 注意：线性的多功能动态流水线可能出现冲突

？如何解决冲突？

为了避免流水线发生冲突，一般采用延迟输入新任务的方法。

调度问题：在非线性流水线的输入端，究竟每间隔多少个时钟周期（采用何种启动距离）向流水线输入一个新任务才能使流水线的各个功能段都不发生冲突。

- **非线性流水线无冲突调度的主要目标：**

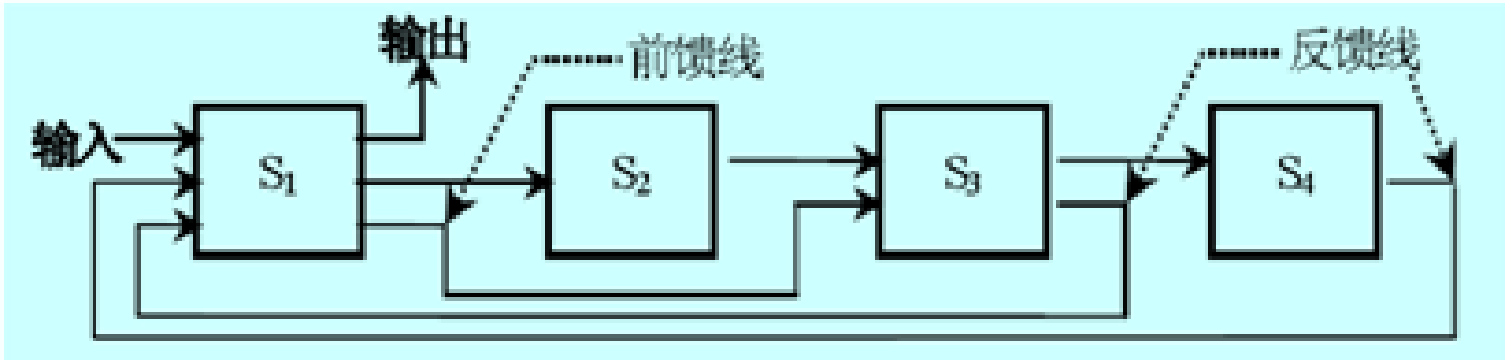
是要找出具有最小平均启动距离的启动循环（即最小的循环周期），按照这样的启动循环向非线性流水线的输入端输入任务，流水线的工作速度最快，而且所有功能段在任何时间都没有冲突。

在一般情况下，这个间隔的时钟周期数应该愈小愈好。在许多非线性流水线中，间隔的周期数往往不是一个常数，而是一串周期变化的数字。

2、单功能非线性流水线的调度

2.1 非线性流水线的表示

- 线性流水线能够用流水线连接图唯一表示
- 非线性流水线无法仅用流水线连接图唯一表示



a) S1, S2, S3, S4, S1, S3, S1

b) S1, S2/S3, S4, S1, S2, S3, S1

连接图不能唯一表示非线性流水线的工作流程，因此，引入二维流水线预约表，用来反映了一个任务使用流水线各功能段的时间关系

t S	1	2	3	4	5	6	7
S1	×			×			×
S2		×			×		
S3		×				×	
S4			×				

确定流水工作流程：S1, S2/S3, S4, S1, S2, S3, S1

预约表

横坐标:时钟周期

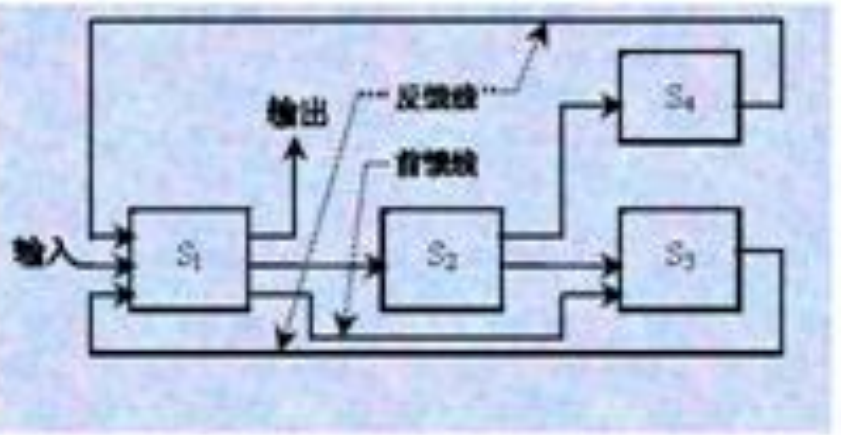
纵坐标:功能段

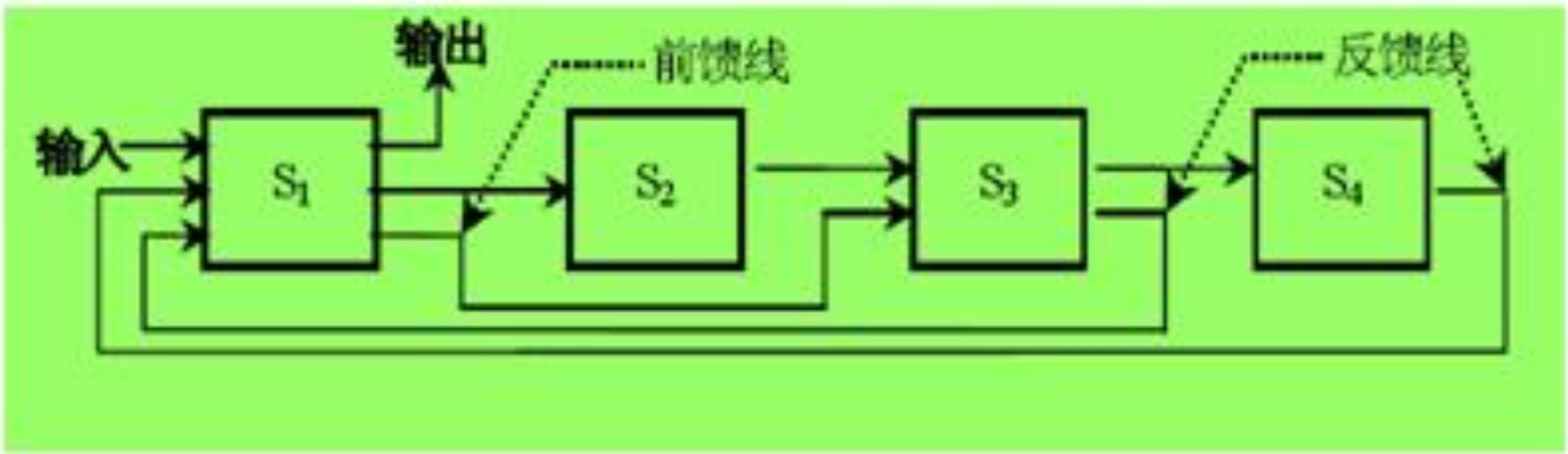
√或者×:某一功能段在某一周期处工作状态

- 预约表的行数：流水线的段数；
- 预约表的列数：一个任务从进入流水线到输出所经过的时钟周期数；
- 一行中可以有多个“×”，表示一个任务在不同时钟周期重复使用了同一流水段；
- 一列中有多个“×”，表示在同一个时钟周期同时占用了多个流水段。

一张预约表可能与多个流水线连接图相对应

功能段 \ 时间	1	2	3	4	5	6	7
S_1	×			×			×
S_2		×			×		
S_3		×				×	
S_4			×				

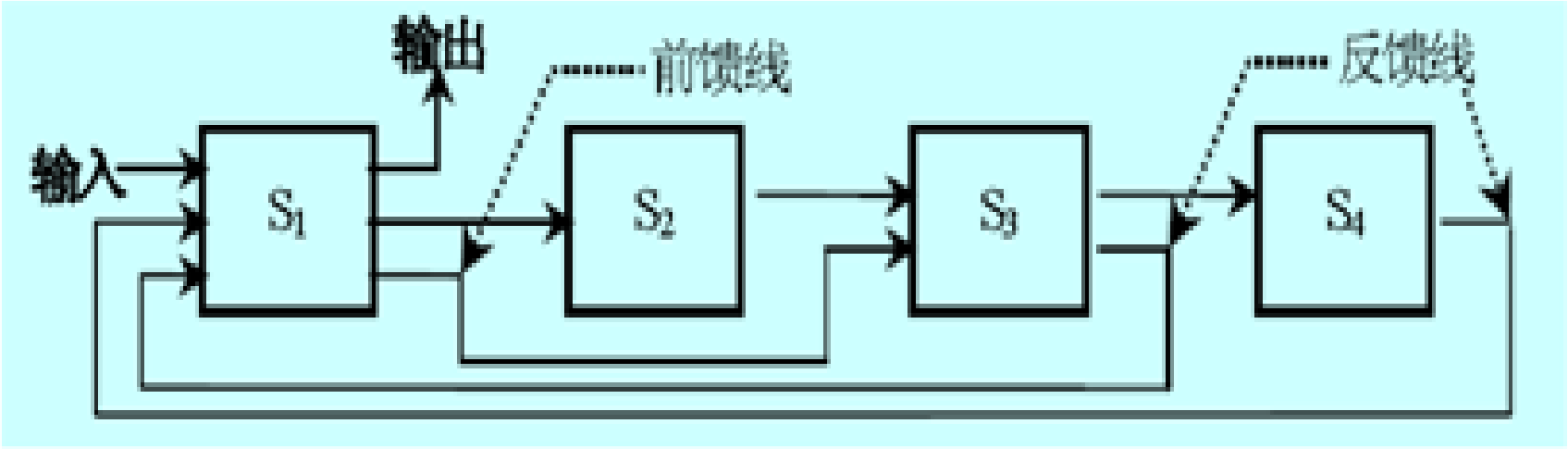




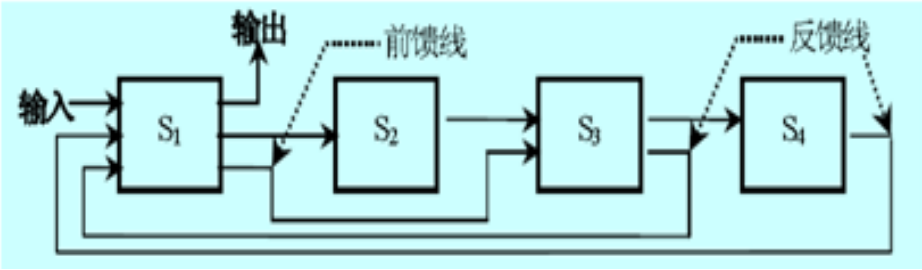
一个流水线连接图对应与多张预约表

时间 / 功能段	1	2	3	4	5	6	7
S_1	X			X			X
S_2		X			X		
S_3		X				X	
S_4			X				

时间 / 功能段	1	2	3	4	5	6	7
S_1	X				X		X
S_2		X					
S_3			X			X	
S_4				X			



2.2 冲突情况分析



功能段 \ 时间	1	2	3	4	5	6	7
S_1	×			×			×
S_2		×			×		
S_3		×				×	
S_4			×				

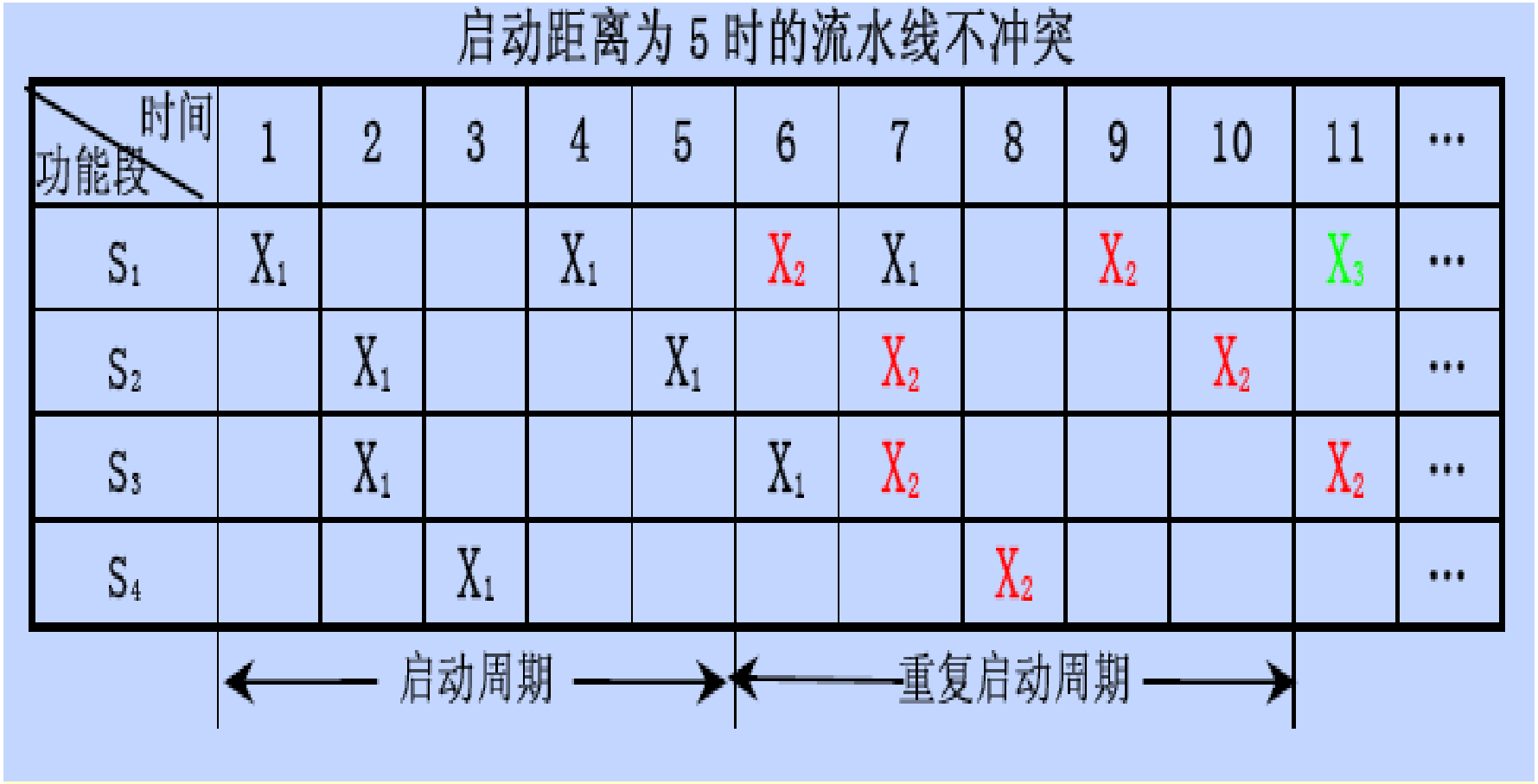
启动距离为 3 的流水线冲突情况

功能段 \ 时间	1	2	3	4	5	6	7	8	9	10	11	…
S_1	X_1			X_1X_2			$X_1X_2X_3$			$X_2X_3X_4$		…
S_2		X_1			X_1X_2			X_2X_3			X_3X_4	…
S_3		X_1			X_2	X_1		X_3	X_2		X_4	…
S_4			X_1			X_2			X_3			…

启动距离为 2 的流水线冲突情况

时间 / 功能段	1	2	3	4	5	6	7	8	9	10	11	…
S_1	X_1		X_2	X_1	X_3	X_2	X_1X_4	X_3	X_2X_5	X_4	X_3X_6	…
S_2		X_1		X_2	X_1	X_3	X_2	X_4	X_3	X_5	X_4	…
S_3		X_1		X_2		X_1X_3		X_2X_4		X_3X_5		…
S_4			X_1		X_2		X_3		X_4		X_5	…

引起非线性流水线功能段冲突的启动距离称为禁止启动距离。如上图所示的非线性流水线，启动距离2和启动距离3都是禁止启动距离。



启动距离为5，启动循环记作（5），又称恒定循环。

启动距离为（1，7）循环时的流水线预约表。图中的启动循环记作（1，7）。

2.3 无冲突调度方法

由E. S. Davidson及其学生于1971年提出

① 根据预约表写出延迟禁止表F

② 由延迟禁止表形成冲突向量C

③ 由所有的向量图画出状态图

④ 由状态图形成最佳调度方案

若，已知某一单功能非线性流水线预约表如下所示：

S \ t	**1**	2	3	4	5	6	7	8	9
1	×								×
2		×	×					×	
3				×					
4					×	×			
5							×	×	

1）由预约表确定延迟禁止表F

F={各段所需间隔的拍数}

确定方法：把预约表的每一行中任意两个“X”之间的距离都计算出来，去掉重复的，由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

S \ t	1	2	3	4	5	6	7	8	9
1	×								×
2		×	×					×	
3				×					
4					×	×			
5							×	×	

F={1，5，6，8}

意义：要想不争用流水线的功能段，相邻两个任务送入流水线的间隔拍数不能为1、5、6、8拍

2）由禁止表形成冲突向量

禁止使用的间隔拍数，可以用一个有m位的二进制向量来表示，这个向量称作冲突向量，其中m是禁止向量中的最大值。对于一张k列的预约表，有m≤k-1。

一般的冲突向量用C＝($C_mC_{m-1}\cdots C_2C_1$)来表示。如果i在禁止向量中，则C_i＝1（冲突），否则C_i=0（不冲突）。其中，C_m一定为1，因为m必定在禁止向量中。

本例：禁止向量（1,5,6,8）

初始冲突向量 C_0 =(10110001)

说明第二个任务可以在第2、3、4、7拍流入流水线

设2拍后新指令进入，则对于第3条指令而言，会产生新的冲突向量。

对第一个任务而言，经过2个时钟周期后，初始冲突向量右移2成为（00101100）；

第二个任务的初始冲突向量为（10110001）

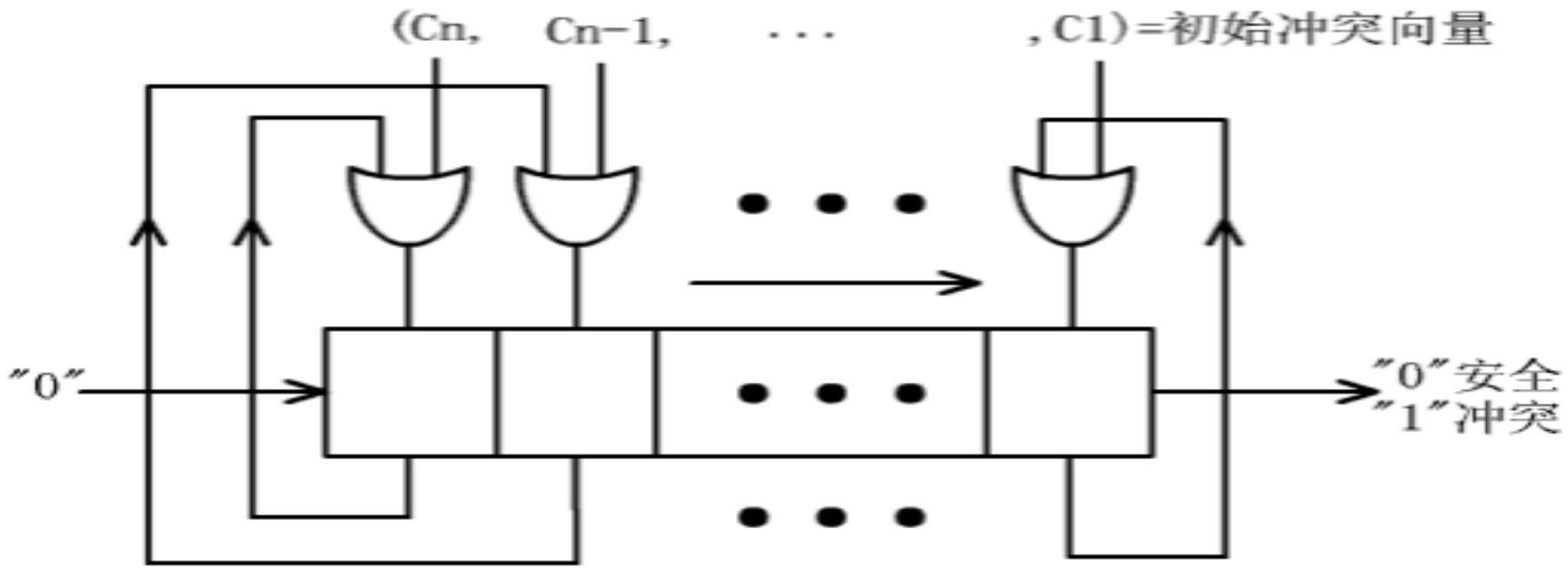
第三条指令不能与第一个和第二个任务冲突。两个冲突向量按位“或”，得到（10111101）

S \ t	1	2	3	4	5	6	7	8	9
1	×		×						×
2		×	×	×	×			×	
3				×		×			
4					×	×	×	×	
5							×	×	×

注意：冲突向量反映出，考虑到现在流水线中已存在的所有任务，下一个任务禁止进入的时刻

3）由所有冲突向量形成状态图

由冲突向量可以构造一张状态图。把初始冲突向量C_0送入一个n位逻辑右移移位器。



用n位右移寄存器实现状态变换，n是最大禁止时间

C_0每过一拍逻辑右移一位：

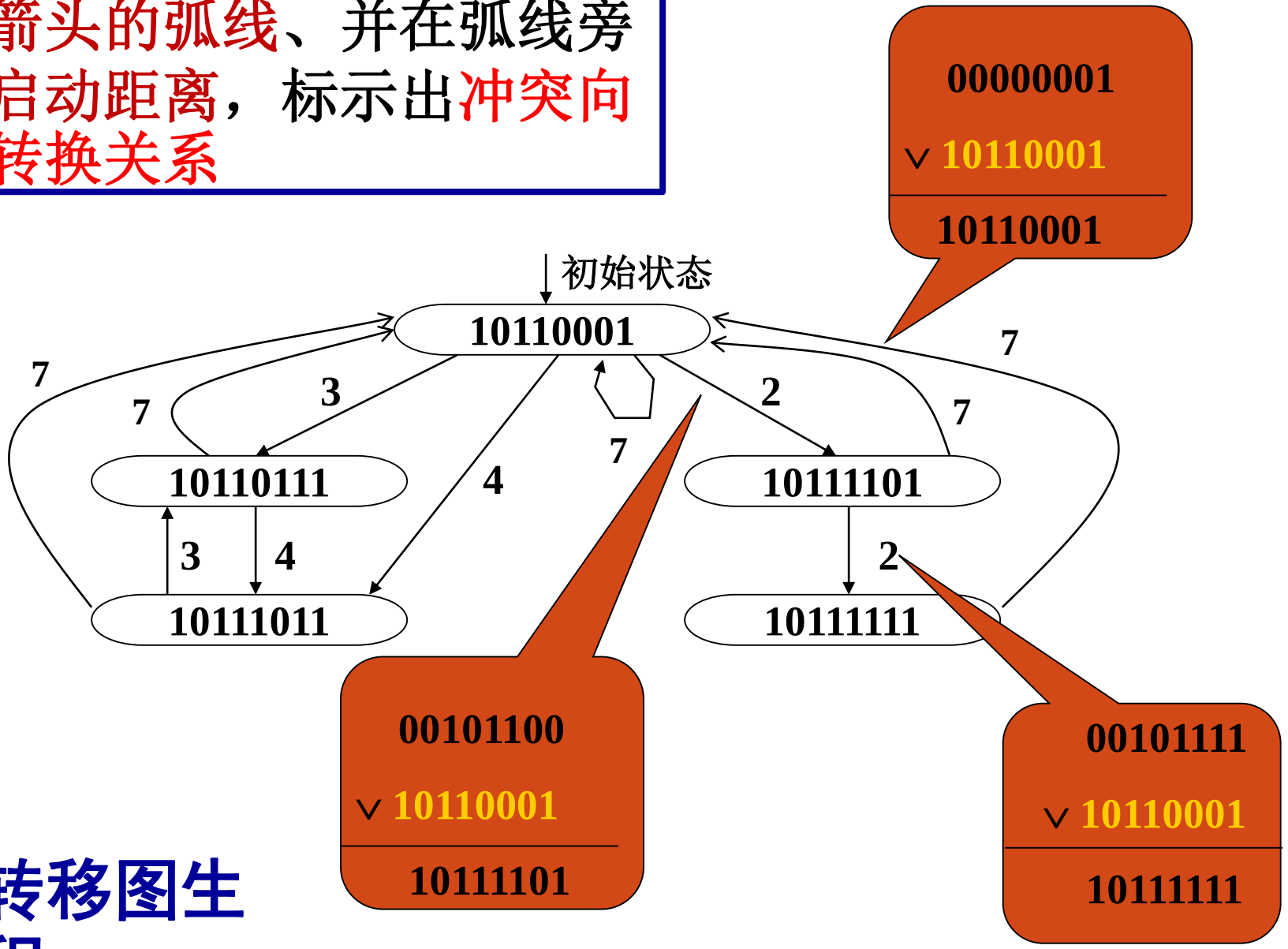
① 若移出0，则允许后续指令进入流水线，同时将移位器中的值再与初始向量C_0按位“或”，形成新的冲突向量C；

② 若移出“1”，则说明以此启动距离向流水线输入任务会发生冲突，因此，不做任何处理。

③ 重复上述步骤，直到不再生成新的冲突向量。

- 可先重复m次，得到若干新的初始冲突向量；
- 再对新生成的冲突向量依次采用与初始冲突向量相同的处理方式进行处理

用带箭头的弧线、并在弧线旁注明启动距离，标示出冲突向量的转换关系

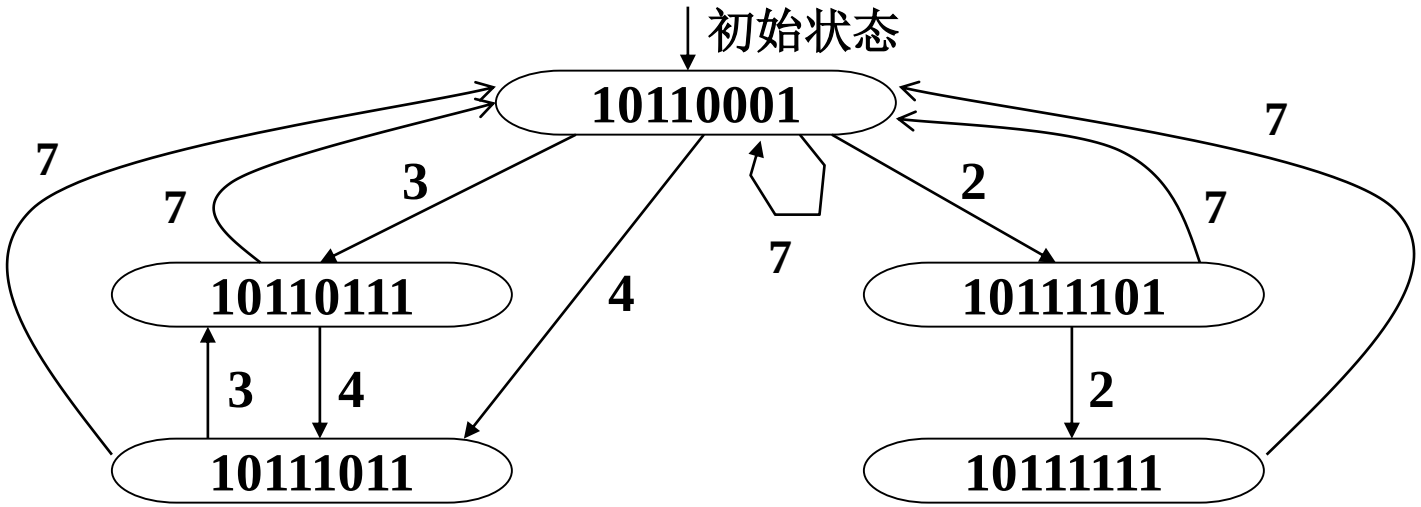


状态转移图生成过程

4）找出最佳调度方案

状态图中的任何一个闭合回路即为一个调度周期策略：

（2，7），（2，2，7），（7），（3，7），（4，7），（4，3），（3，4），（3，4，7），（3，4，3，7）



从各个闭合回路中找出平均启动距离（间隔拍数）最小的一个即最佳调度方案。

本例中调度方案如下：

调度策略	平均启动距离	调度策略	平均启动距离
(2，7)	4.50	(4，3)	3.50
(2，2，7)	3.67	(4，3，7)	4.67
(3，4)	3.50	(4，7)	5.00
(3，7)	5.00	(7)	7.00
(3，4，7)	4.67		

- 例子：一条有4个流水段的非线性流水线，每个流水段的延迟时间都相等，它的预约表如下图：

时间 / 流水段	1	2	3	4	5	6	7
S1	X						X
S2		X				X	
S3			X		X		
S4				X			

（1）写出流水线的禁止表和初始冲突向量

（2）画出调度流水线的状态图

（3）求流水线的最佳调度方案和最小平均延时

（4）求平均延时最小的等间隔调度方案。

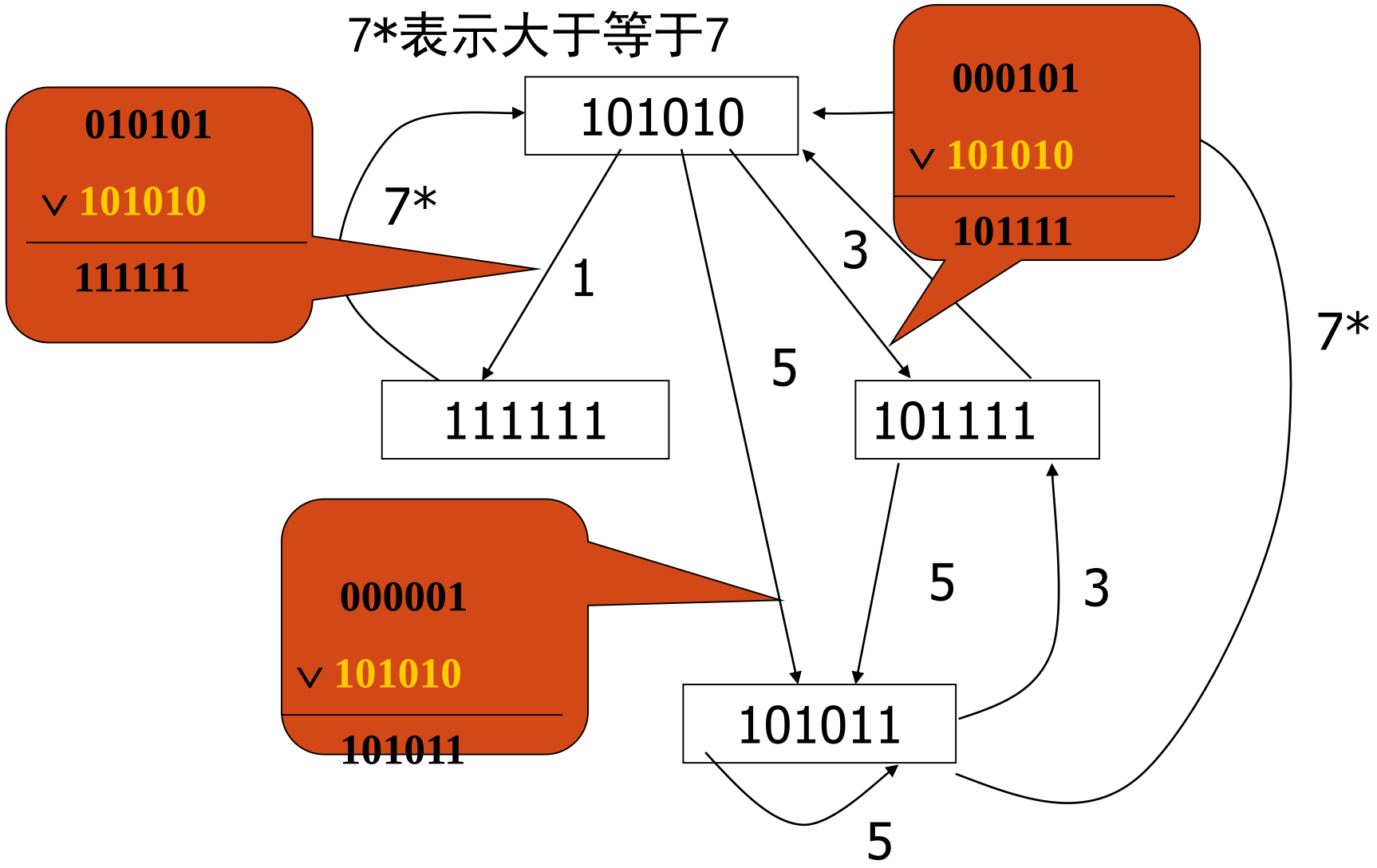
- 解：（1）禁止表为F={2，4，6}

冲突向量：用二进制表示，长度是禁止向量的最大距离。

冲突向量C=（$C_6C_5C_4C_3C_2C_1$），由禁止向量，$C_2=C_4=C_6=1$，其余位为0，冲突向量为

C=（101010）。

（2）由冲突向量构造一张图：将C放到一个6位逻辑右移移位器，当从移位器移出0，用移位器中的值与初始冲突向量做“按位或”，得到一个新的冲突向量。当移位器移出1，不做任何处理。重复这个步骤。对产生的每一个新的冲突向量做同样处理。在初始冲突向量和所有形成的冲突向量之间，箭头连接。



当右移1、3、5和大于等于7位时，移出位是0，表示用这些启动距离输入新任务不会发生冲突。

（3）从状态图中可以找到许多不发生流水段冲突的调度方案。由此确定平均延时最小的调度方案是：

（1，7）、（3，5）、和（5，3）。

（4）

简单循环（调度方案）	平均启动距离（平均延时）
（1，7）	4
（3，5）	4
（5，7）	6
（3，5，7）	5
（5，3，7）	5
（5，3）	4
（5）	5
（7）	7

（4）如果输入6个指令，则每种方案的实际吞吐率为：

（1，7）：1+7+1+7+1+7=24拍

Tp=6/24

（3，5）：3+5+3+5+3+7=26拍

Tp=6/26

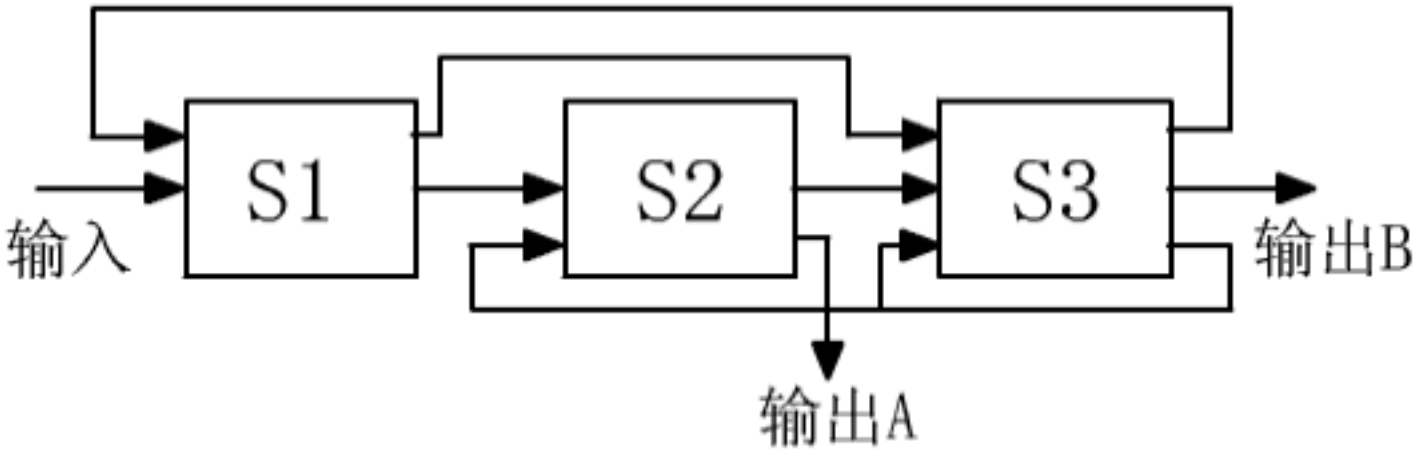
（5，3）：5+3+5+3+5+7=28拍

Tp=6/28

所以，最佳调度方案应为（1，7）

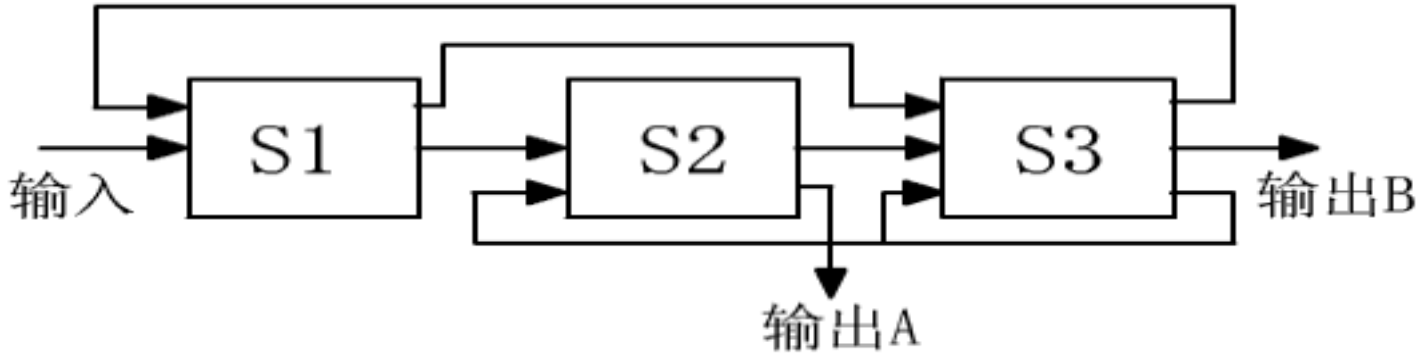
3、多功能流水线的调度(扩展学习)

下图给出一条多功能流水线，该流水线有S1、S2、S3三段，既有S1到S2到S3的线性流水线连接，又有S1到S3的前馈连接，还有S3到S1，S3到S2及S3到S3的反馈连接。



一条多功能流水线

预约表



一条多功能流水线

	t1	t2	t3	t4	t5
S1	A			A	
S2		A			
S3			A		A

A功能

	t1	t2	t3	t4	t5
S1		B			B
S2				B	
S3	B		B		

B功能

A、B双功能预约表

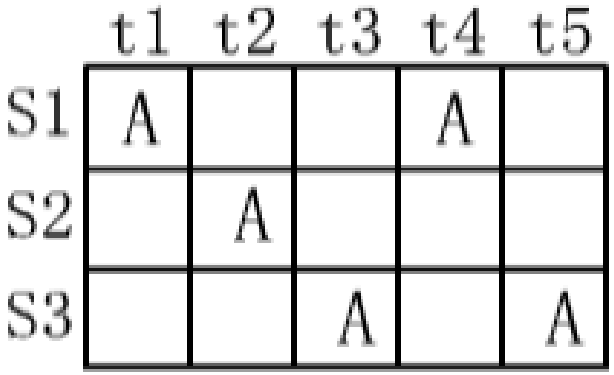
无冲突调度：求冲突向量

- 对于A功能装入后，在下一个时钟周期能否启动下一个A功能，或在B功能装入后，能否启动下一个B功能的单功能预约表的冲突向量，可用等待时间分析法。
- 对于先装入A，再装入B，或先装入B，再装入A的多功能预约表，就不能用上述方法求多个功能相互之间的冲突向量了，可用预约表重合移动法.

无冲突调度：预约表重合移位法

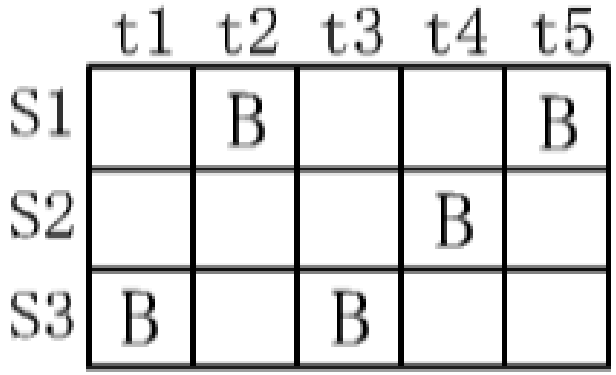
- 首先把预约表B放在预约表A上，时钟周期对齐，再把两者向相反方向移动。
- 每移动一个时钟周期，若任一个格中出现A和B两个符号，则说明在这个等待时间发生冲突，那么冲突向量中对应该等待时间的位为1；
- 若不出现两个符号在一个格中的情况，则说明在这个等待时间不发生冲突、那么冲突向量中对应该等待时间的位为0．如此下去，直至两表分离。
- 这样共移动n-1次。就得出A装入后再装入B的冲突向量为1010。

无冲突调度：预约表重合移位法



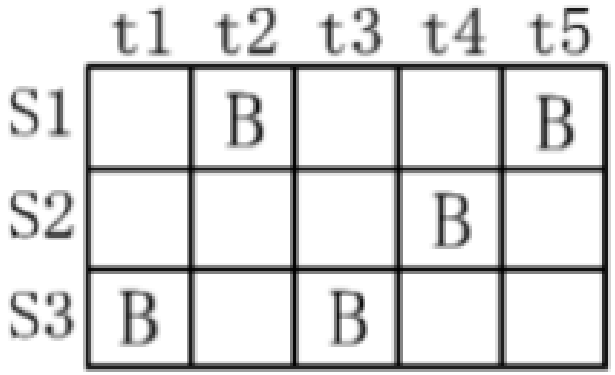
	t1	t2	t3	t4	t5
S1	A			A	
S2		A			
S3			A		A

A功能



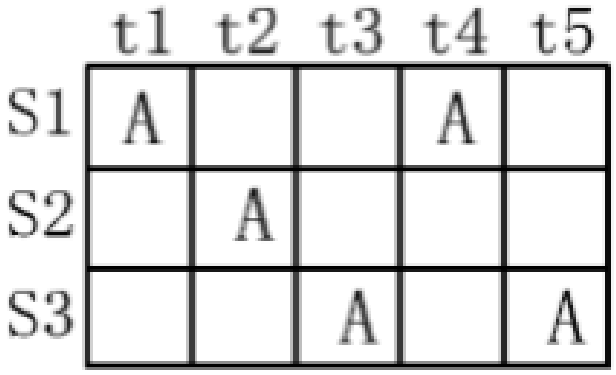
	t1	t2	t3	t4	t5
S1		B			B
S2				B	
S3	B		B		

B功能



	t1	t2	t3	t4	t5
S1		B			B
S2				B	
S3	B		B		

B功能



	t1	t2	t3	t4	t5
S1	A			A	
S2		A			
S3			A		A

A功能

无冲突调度：交叉冲突向量

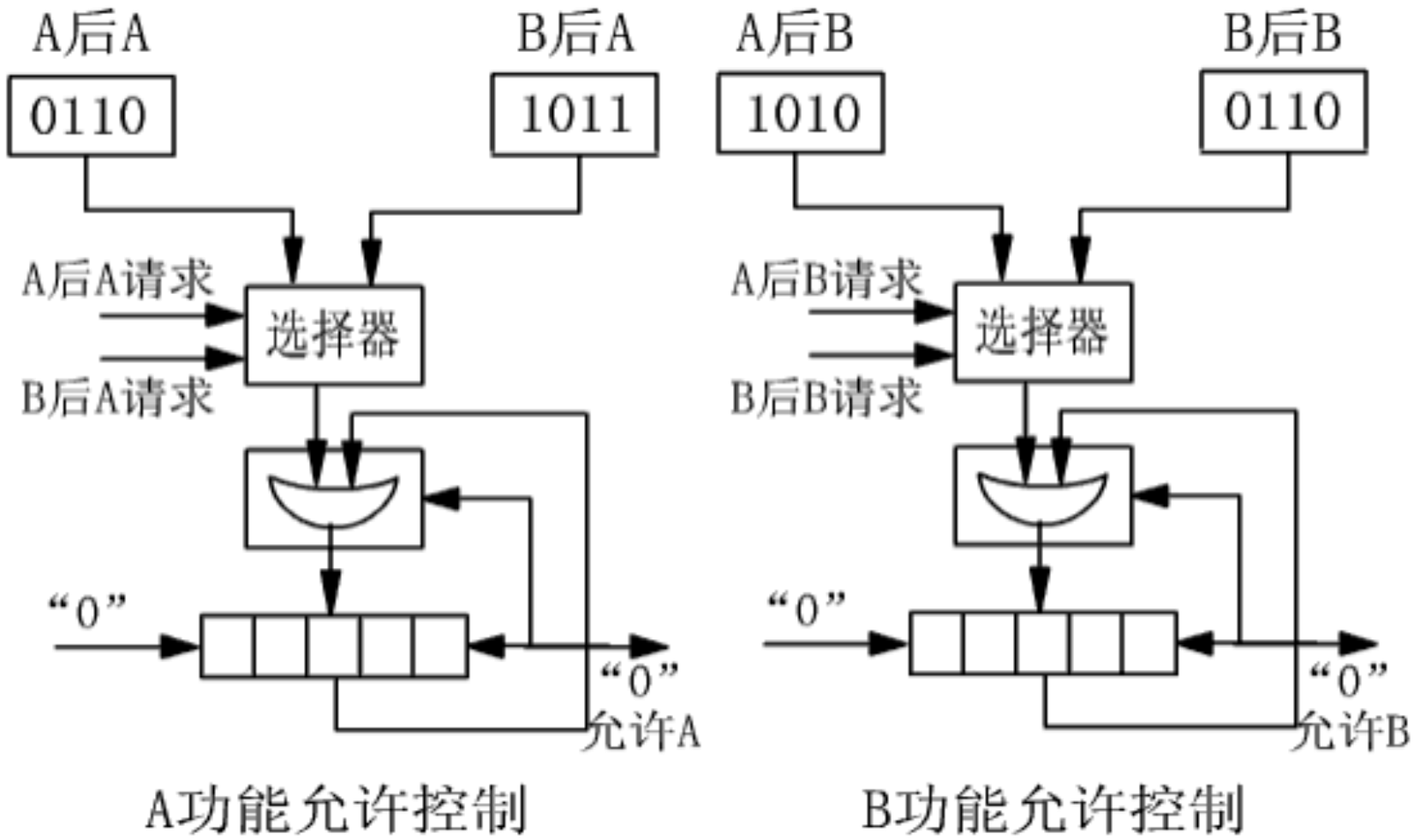
- 我们可以使用交叉冲突向量来反映具有A、B两种功能的动态流水线的禁止使用情况，这样就有4种交叉冲突向量：
- (1)先A后A：是0110
- (2)先A后B：是1010
- (3)先B后A：是1011
- (4)先B后B：是0110

无冲突调度：冲突向量矩阵

- 一般，一条有P个功能的流水线，就有P^2个交叉冲突向量，可以分别归类写成P个冲突向量矩阵Mp，其中p分别为1到P。冲突向量矩阵Mp表示流水线按p功能装入一个任务后与按功能装入后继任务所产生的全部冲突向量的集合。

$$M_A=\begin{bmatrix} 0110 \\ 1010 \end{bmatrix} \qquad M_B=\begin{bmatrix} 1011 \\ 0110 \end{bmatrix}$$

无冲突调度

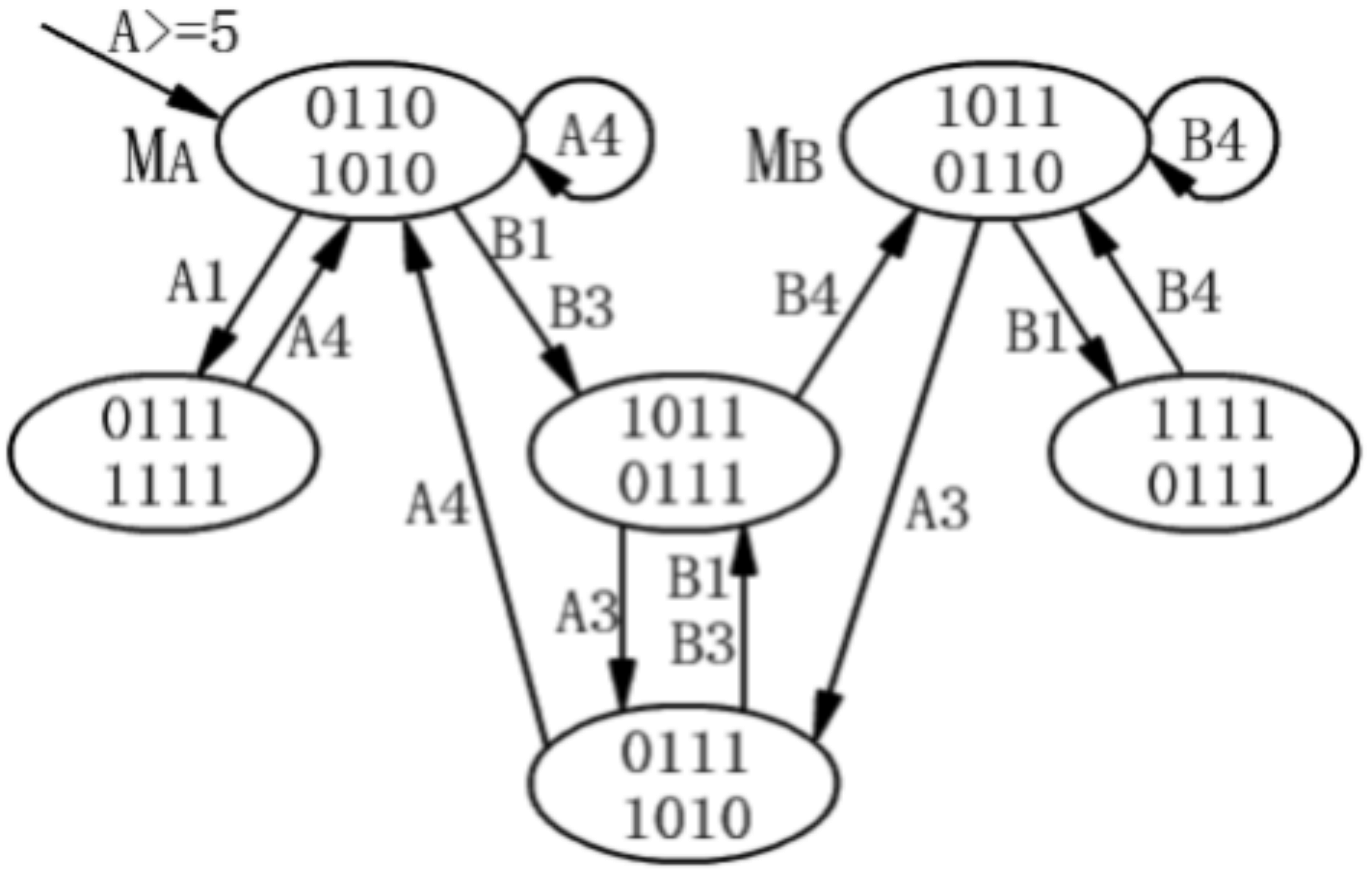


A功能允许控制　　B功能允许控制

无冲突调度

- 双功能流水线的控制和单功能流水线原理相同，不同的是用二个移位寄存器分别存放冲突矩阵的两行．一个移位寄存器控制A功能能否装入，另一个移位寄存器控制B功能能否装入，两个移位寄存器初始值，存放最早装入流水线的二功能的Mi的冲突向量矩阵内容。然后让它们每周期同时右移一位，左面补0。在移出位为"0"的时点上与后继任务相应功能的冲突向量矩阵中的对应行进行按位"或"，形成新的冲突向量矩阵，

无冲突调度：状态图



双功能状态图

无冲突调度：启动循环

- 预约表中一行符号最多的个数是2。

  最小平均等待时间的情况是：
- 如果流水线的关系是A·A·A…，为(1+4)/2=2.5，是最小循环；
- 若为A·B·A·B…，则为(3+1)/2=2；
- 若为B·A·B…，也为(3+1)/2=2；
- 若为B·B…，则为(1+4)/2=2.5 。

优化调度

- 对于多功能的非线性流水线的优化调度与单功能的一样。

避免冲突的措施：采用延迟输入新任务的方法 ——→每间隔多少个时钟向流水线输入一个新任务

↓

间隔的周期数不是常数，而是一串周期变化的数字

↓ 调度任务

是找出一个 最小的循环周期

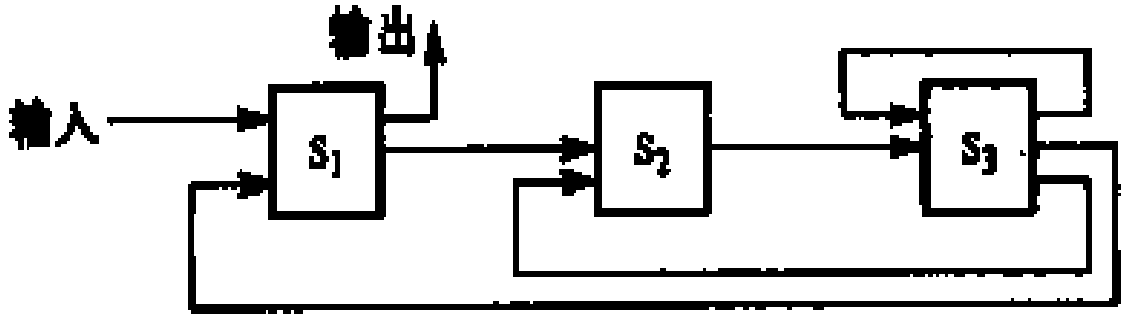
↓

按这周期向流水线输入新任务（即不产生冲突又使其流水线的吞吐率和效率最高）

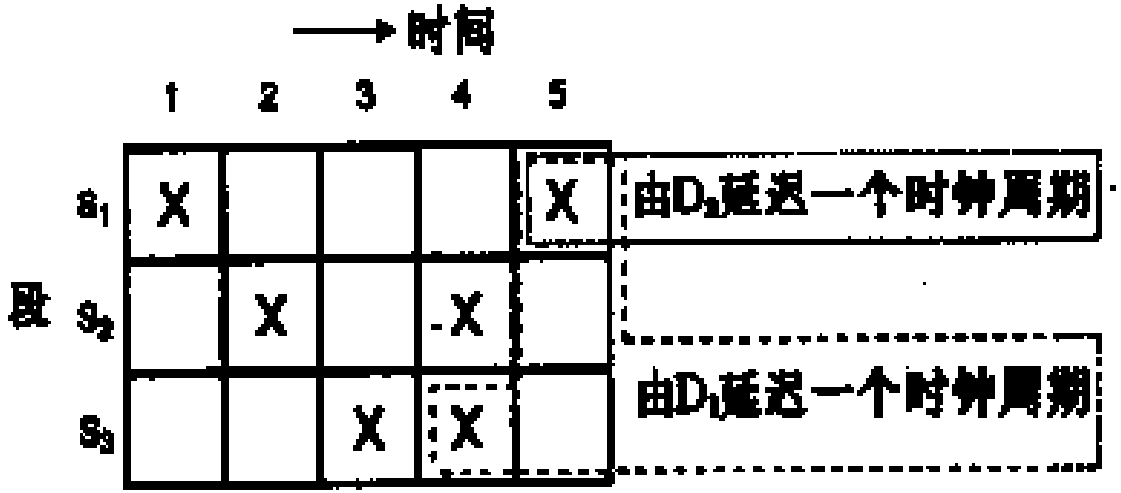
↓ 提出

基于MAL优化调度技术（主要适用于单功能非线性流水线。若将每种功能的预约表重叠，并采用单功能非线性流水线的基本调度方法可解决动态多功能非流水线的调度）

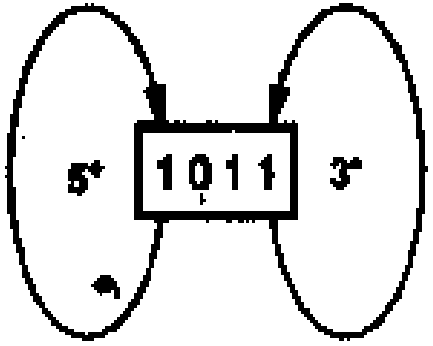
非线性流水线调度优化技术举例



(a) 一条 3 段流水线

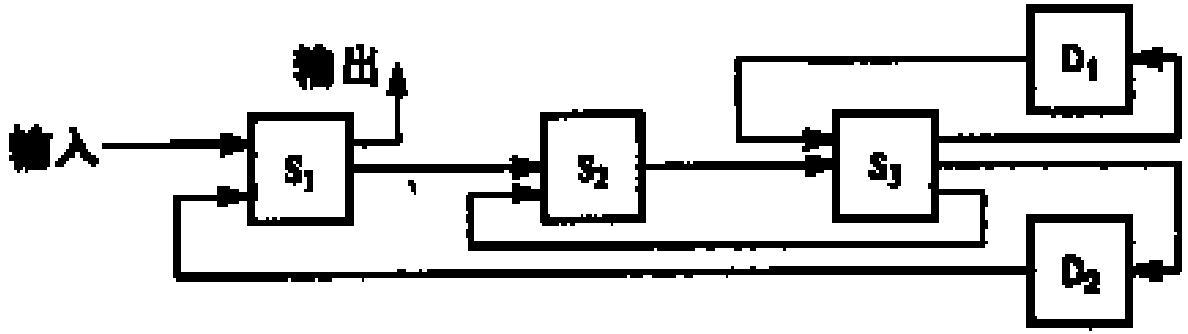


(b) 预约表以及被延迟的操作

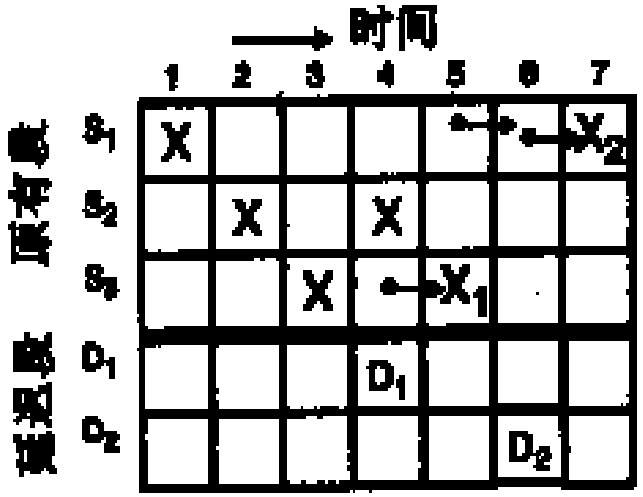


(c)MAL=3 的新状态变换图

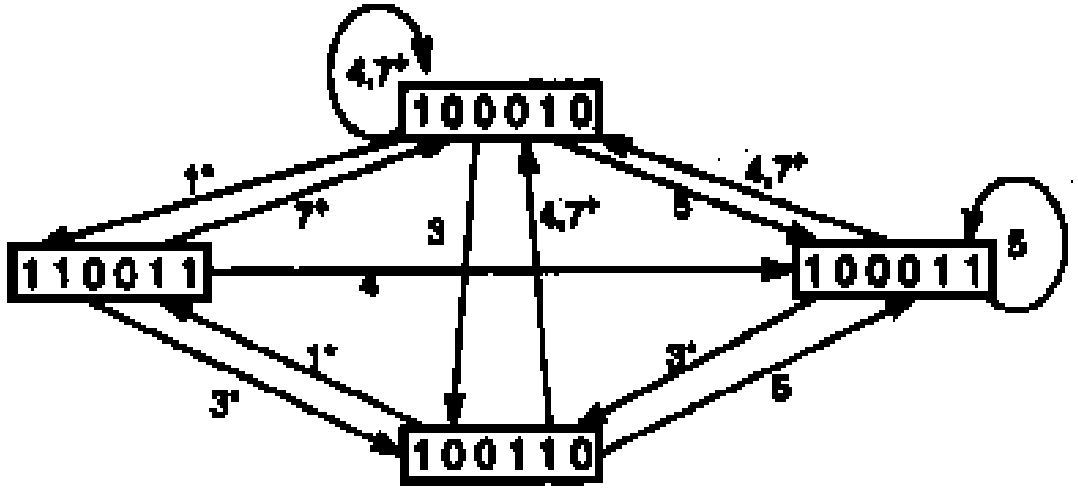
示例：插入非计算延迟以减少MAL



(a) 两个非计算延迟段的插入



(b) 修改后的预约表



(c) 修改后的状态图，减小后的 MAL=(1+3)/2=2

3.3＊指令级高度并行的超级处理机

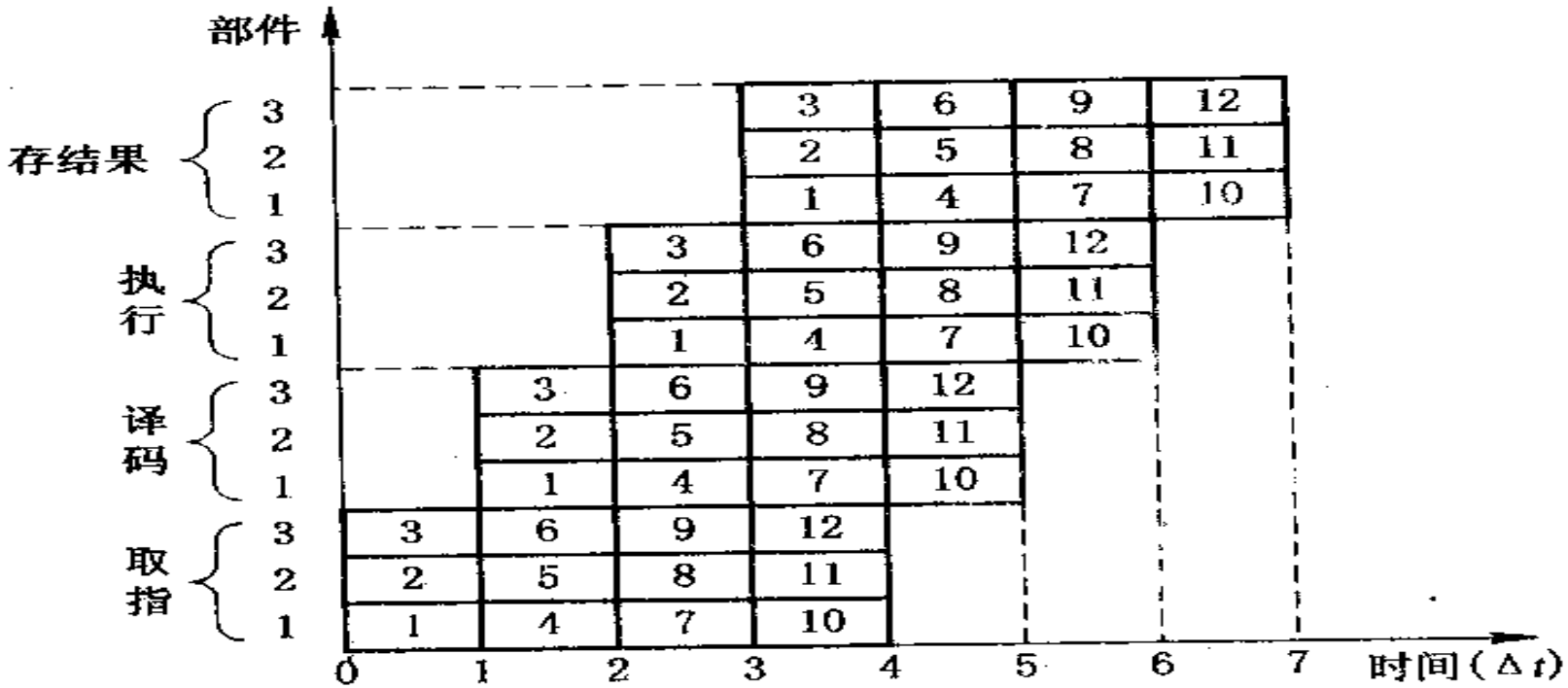
- 超标量(superscalar)处理机
- 超长指令字（VLIW）处理机
- 超流水线（Superpipelining）处理机
- 超标量超流水线处理机

超标量(superscalar)处理机

常规的标量流水线处理机：每隔△t解释完一条指令。

超标量处理机：使用多指令流水线，每个△t流出m条指令。也称度为m。

实现：配置多套部件（资源重复）



超长指令字（VLIW）处理机

编译程序找出指令间潜在并行性，将多个能并行执行的指令或无关操作先行压缩组合在一起，形成一条多个操作段的超长指令。由其来控制多个功能部件并行操作。每个操作段控制一个部件。

单指令多操作码多数据的系统结构

超流水线处理机

超级流水线：将CPU处理的指令的操作进一步细分，增加流水线级数或加大流水线长度来提高频率，又叫深度流水线。

每隔 $\Delta t'$ 仍只流出一条指令，但它的 $\Delta t'$ 值小，一台度为m的超流水线处理机的 $\Delta t'$ 只是基本机器周期 Δt的$1/m$。

着重开发时间并行性，在公共硬件上采用较短时钟周期、深度流水来提高速度。

超标量超流水线处理机

超标量超流水线处理机是超标量流水线与超流水线机的结合。

在一个 $\Delta t'$（等于 $\Delta t/n$）发射了k条指令（超标量），而每次发射时间错开 $\Delta t'$（超流水），相当于每拍 Δt 流出了 $n \cdot k$ 条指令，即并行度 $m=k \cdot n$。

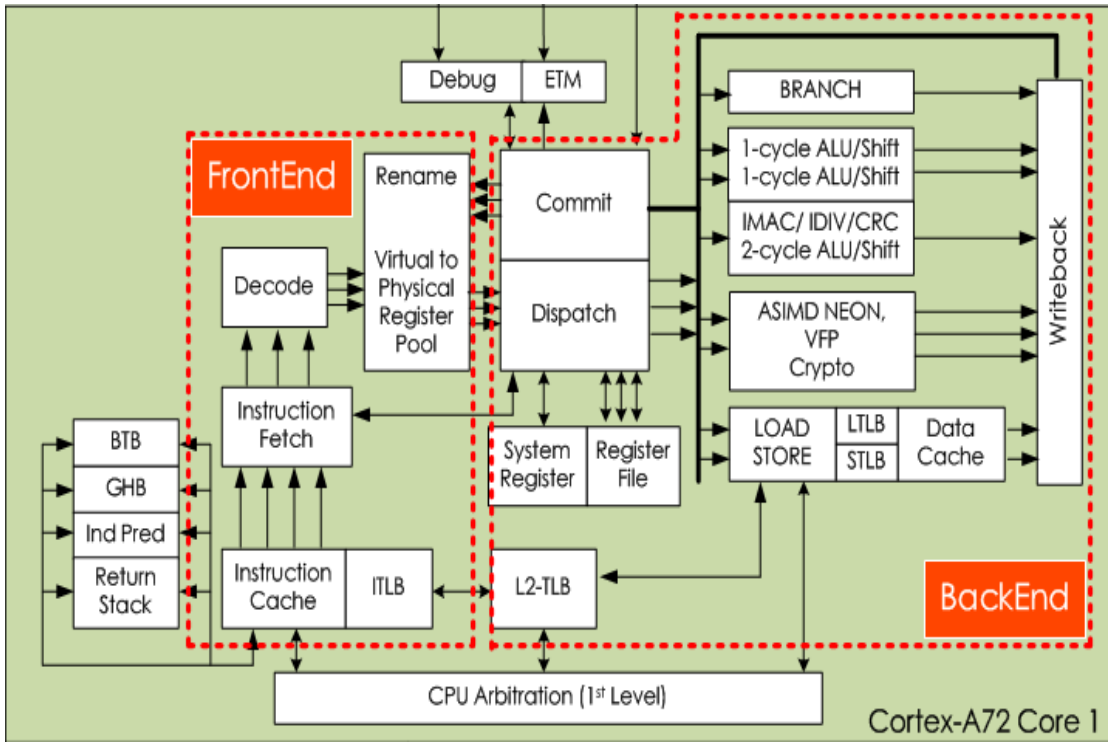
例如 $k=3, n=3$，完成12个任务时，就只需5 Δt，其并行度 $m=9$。

扩展介绍：基于ARMv8的鲲鹏流水线技术

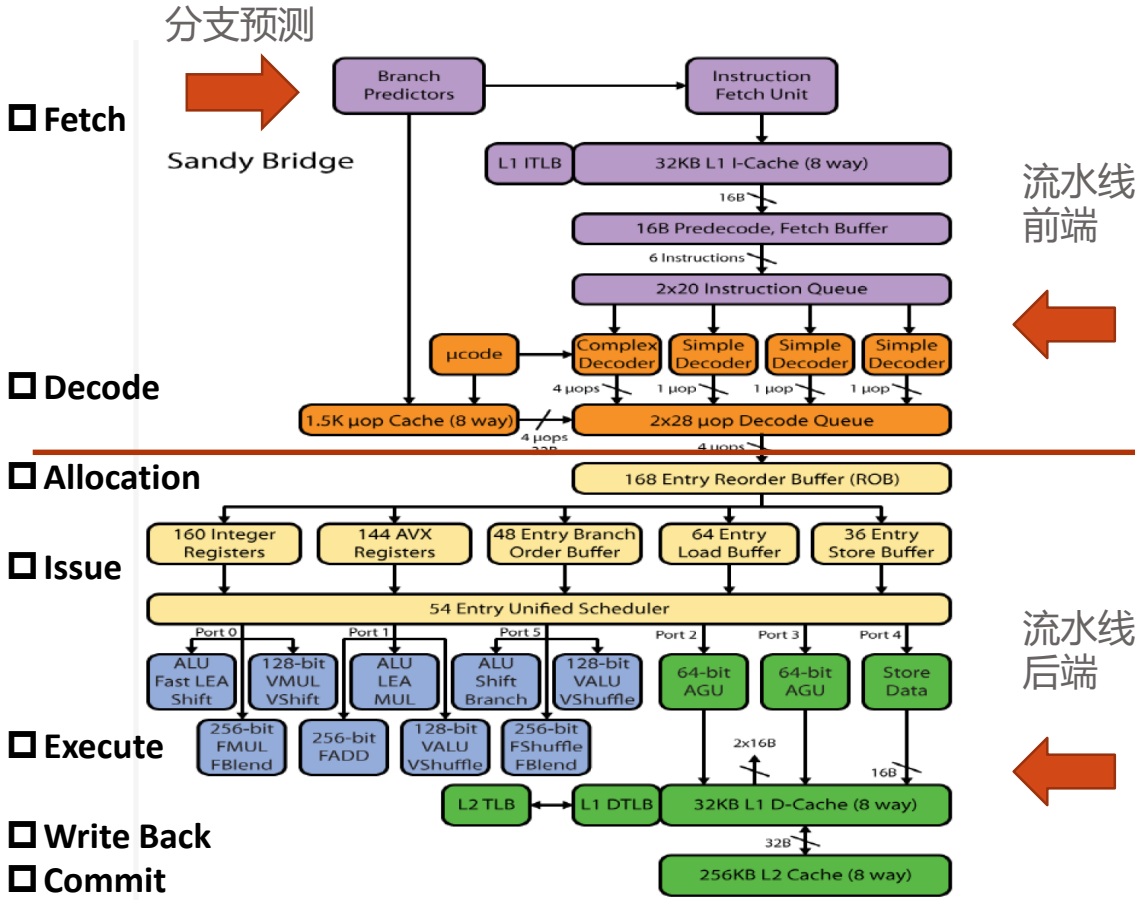
ARM流水线的基本执行顺序

- 取指令（Fetch）：从存储器读取指令；
- 译码（Decode）：译码以鉴别它是属于哪一条指令；
- 执行（Execute）：将操作数进行组合以得到结果或存储器地址；
- 缓冲/数据（Buffer/data）：如果需要，则访问存储器以存储数据；
- 回写：（Write-back）：将结果写回到寄存器组中；

CPU流水线： ARM vs X86



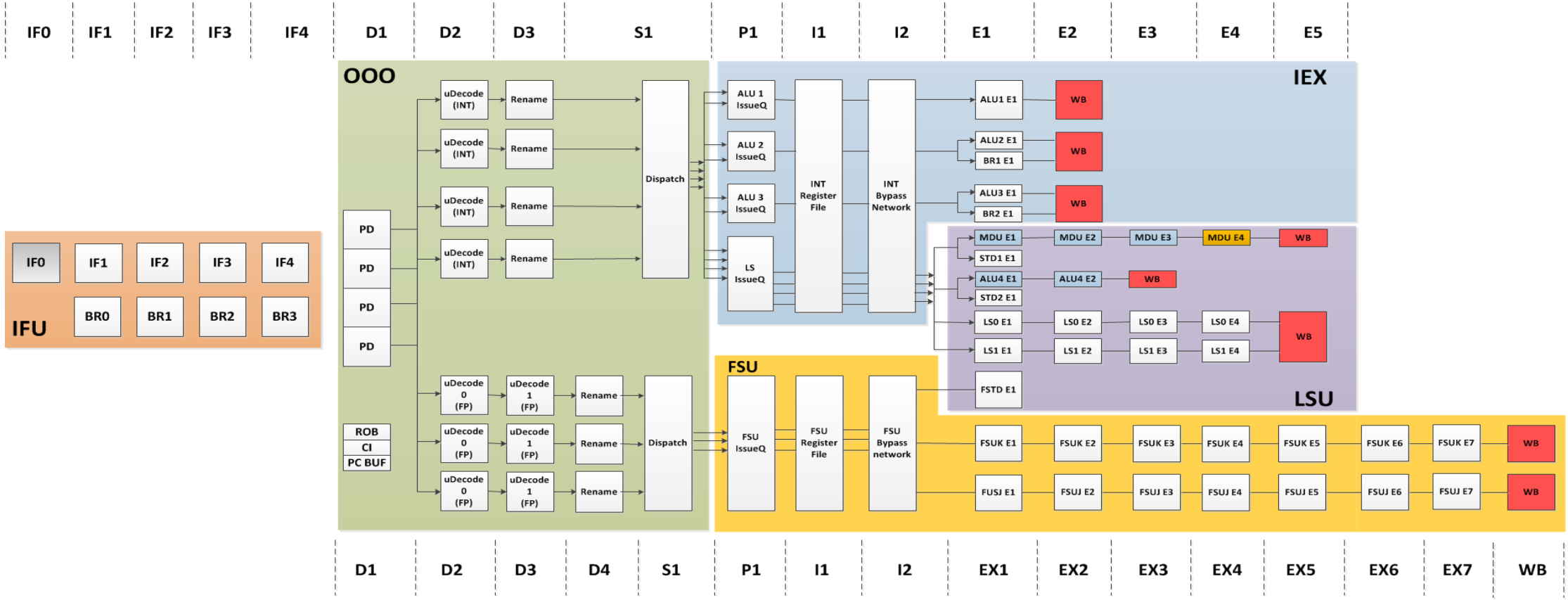
ARM64



X86 Xeon

ARM流水线与Xeon基本一致，按流水线前后端分解

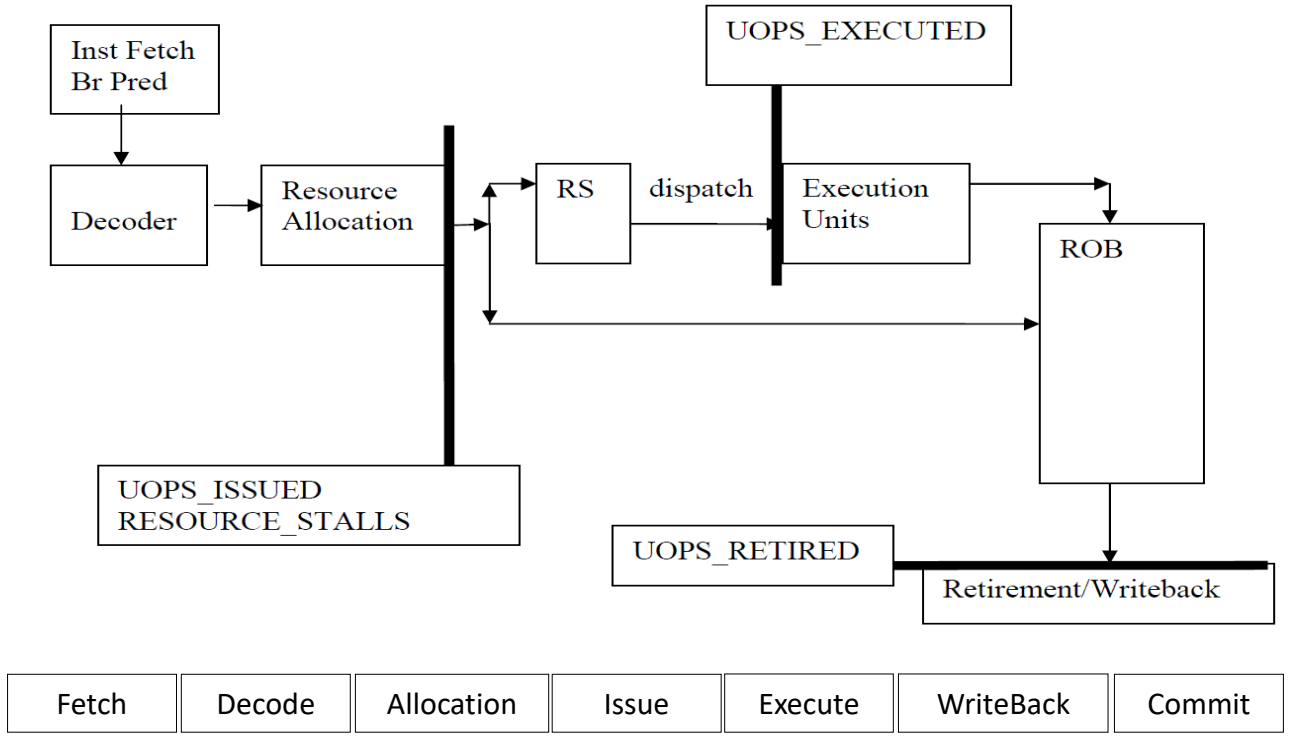
基于ARMv8的鲲鹏流水线结构



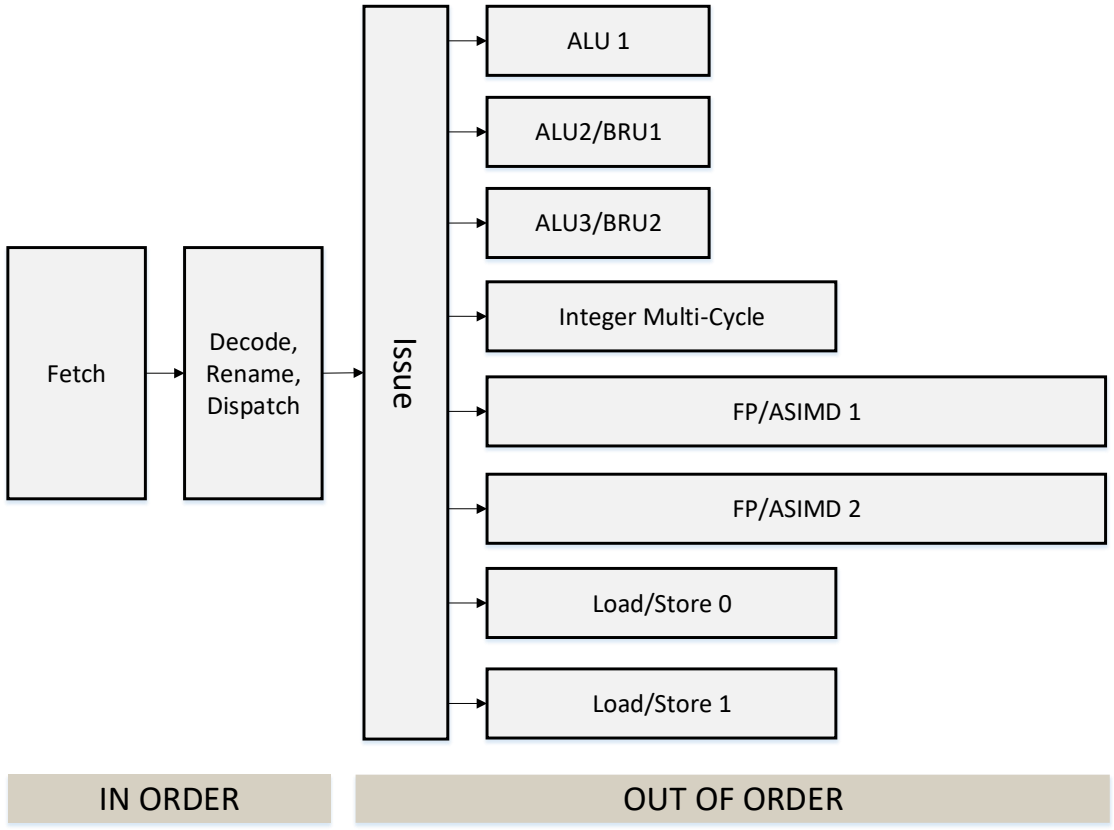
Taishan coreV110 Pipeline Architecture

鲲鹏CPU流水线主要阶段

- **Fetch**：提取指令并计算下一次Fetch的地址。包括指令缓存、Branch Prediction、Branch Target Buffer、Return Address Stack。
- **Decode**
  - 分解指令流到独立指令；
  - Translate X86指令到RISC-like Uops;
  - 理解指令语义，包括指令类型（Control、Memory、Arithmetic，等等），运算操作类型、需要什么资源(读和写需要的寄存器，等等）。
- **Allocation:**RegisterRenaming+Resources Reservation.
- **Issue**：分发指令到相应执行单元，从这儿开始进入错序执行阶段。
- **Execute**：指令执行阶段
- **Write Back**：将执行结果写入Register File、Reorder Buffer、等等
- **Commit**：重整执行结果次序、决定Speculative Execution正确性，最终输出结果。



鲲鹏流水线技术：弱保序



CPU弱保序，即乱序执行：

- 处理器不按程序规定的顺序执行指令，它根据内部功能部件的空闲状态，动态分发执行指令，但是指令结束的顺序还是按照原有程序规定的顺序。
- 处理器内部功能部件并行运转，避免了不必要的阻塞，有效提高了处理器执行指令的性能。
- 处理器分析影响执行结果的指令，避免出现有显式的数据依赖和控制依赖的乱序，但是，特殊情况下的读写乱序可能影响程序执行结果，需要软件甄别。

制约CPU效率因素

•CPU流水线前端限制：执行单元空闲，但前端不能输送充分多操作指令

•CPU流水线后端限制：执行单元繁忙或执行时等待数据，指令执行出现等待

•分支预测错误：执行错误分支浪费时间 + 处理错误分支执行消耗时间。