

第5章 标量处理机

肖梦白

xiaomb@sdu.edu.cn

<https://xiaomengbai.github.io>

5.1 先行控制技术

5.2 流水线技术

5.3 超标量处理机

5.4 超流水线处理机

5.4 超标量超流水线处理机

5.1 先行控制技术

5.1.1 指令的重叠执行方式

5.1.2 先行控制方式的原理和结构

5.1.3 数据相关

5.1.4 控制相关

5.1.1 指令的重叠执行方式

指令的三个阶段：

- **取指令**：按照指令计数器的内容访问主存储器，取出一条指令送到指令寄存器
- **指令分析**：对指令的操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数地址，并用这个地址读取操作数
- **指令执行**：根据操作码要求，完成规定的功能，将运算结果写到寄存器或主存储器



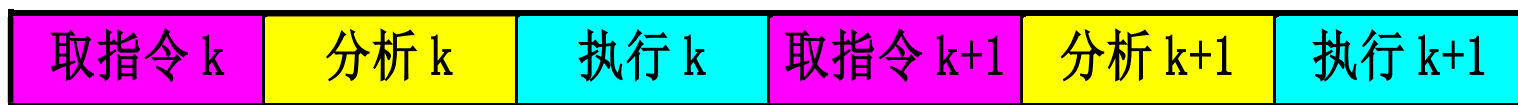
时间

1. 顺序执行方式

➤ 执行 n 条指令所用的时间为：

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

➤ 如果每段时间都为 t ，则执行 n 条指令所用的时间为： $T = 3nt$

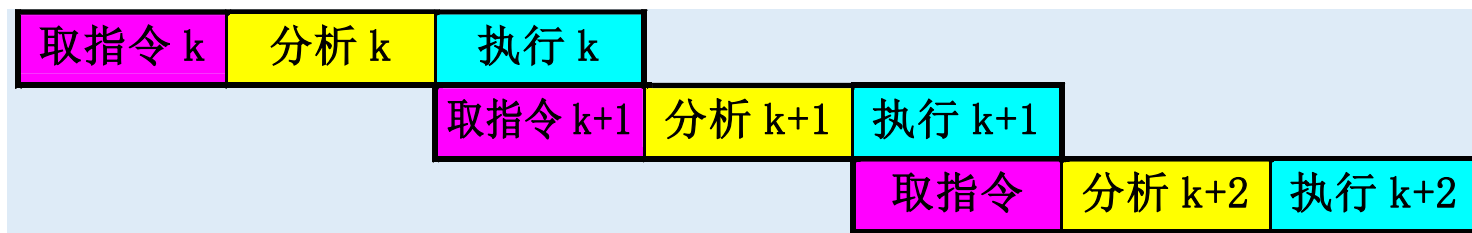


➤ 主要优点：控制简单，节省设备

➤ 主要缺点：速度慢，功能部件的利用率低

2. 一次重叠执行方式

➤ 如果两个过程的时间相等，则执行 n 条指令的时间为： $T = (1 + 2n)t$



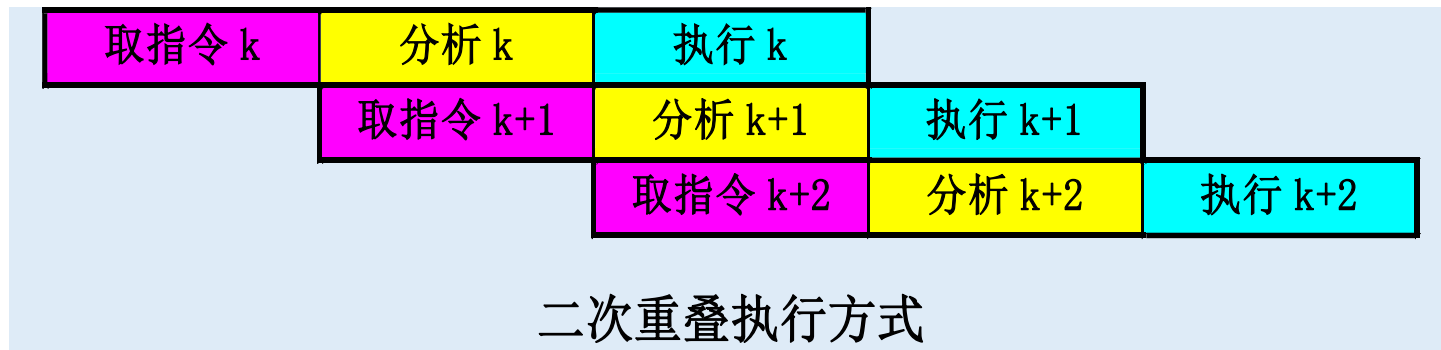
- 主要优点：指令的执行时间缩短，功能部件的利用率明显提高。
- 主要缺点：需要增加一些硬件，控制过程稍复杂。

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：

$$T = (2 + n) t$$

- 在理想情况下，处理机中同时有三条指令在执行。
- 处理机的结构要作比较大的改变，需要采用先行控制技术。



5.1.2 先行控制方式的原理

1. 采用二次重叠执行方式必须解决两个问题：

1) 有独立的取指令部件、指令分析部件和指令执行部件

- 把一个集中的指令控制器，分解成三个独立的控制器：存储控制器、指令控制器、运算控制器

2) 要解决访问主存储器的冲突问题

- 取指令、分析指令、执行指令都可能要访问存储器

5.1.2 先行控制方式的原理

2. 解决访存冲突的方法：

(1) 采用低位交叉存取方式：

- 这种方法不能根本解决冲突问题。
- 指令、读操作数、写结果。

(2) 两个独立的存储器：独立的指令存储器和数据存储器。

- 如果再规定，执行指令所需要的操作数和执行结果只写到通用寄存器，则取指令、分析指令和执行指令就可以同时进行。
- 在许多高性能处理机中，有独立的指令Cache和数据Cache。这种结构被称为哈佛结构。

5.1.2 先行控制方式的原理

2. 解决访存冲突的方法:

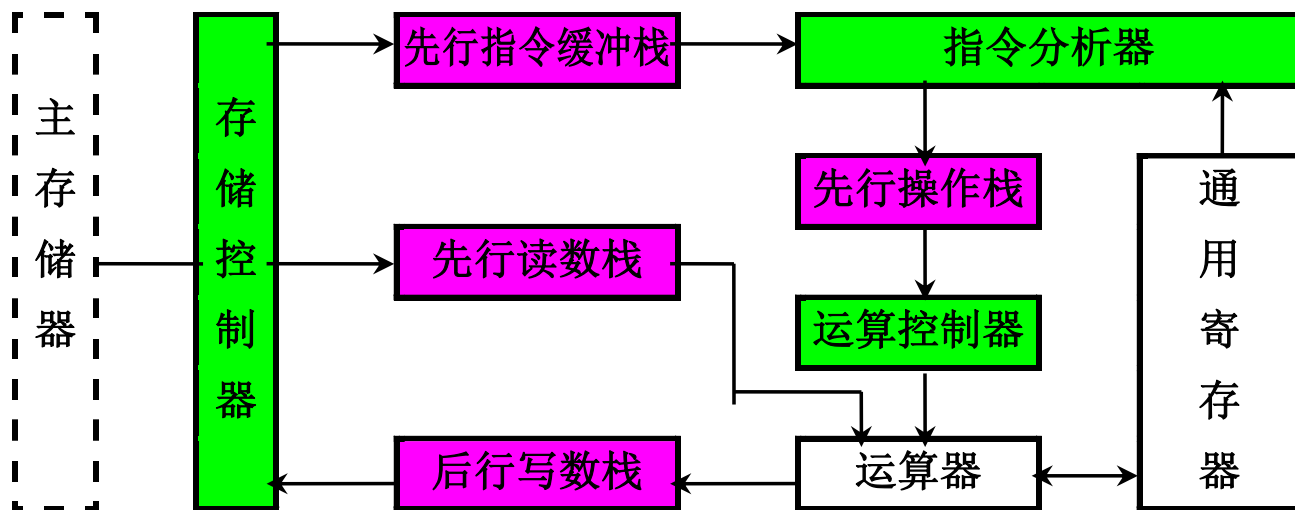
(3) 采用先行控制技术

- 采用先行控制技术的关键是缓冲技术和预处理技术。
- 缓冲技术通常用在工作速度不固定的两个功能部件之间。设置缓冲栈的目的是用来以平滑功能部件之间的工作速度。
- 在采用了缓冲技术和预处理技术之后，运算器能够专心于数据的运算，从而大幅度提高程序的执行速度。

5.1.2 先行控制方式的原理

3. 处理机结构：

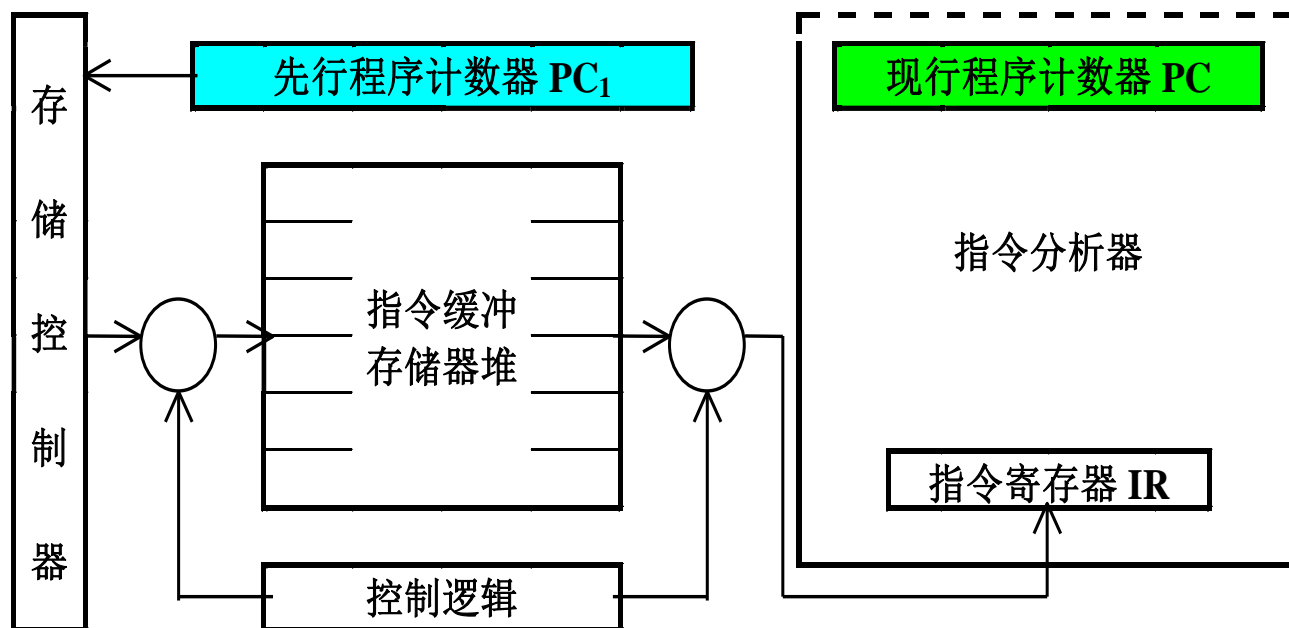
- 1) 三个独立的控制器：存储控制器、指令控制器、运算控制器
- 2) 四个缓冲栈：先行指令缓冲栈、先行读数缓冲栈、先行操作栈、后行写数栈。



➤先行指令缓冲栈的组成

- 作用：只要指令缓冲栈没有充满，就自动发出取指令的请求。
- 设置两个程序计数器：先行程序计数器 PC_1 ，用来指示取指令；现行程序计数器 PC ，记录指令分析器正在分析的指令地址。

➤存在的主要问题：各类指令“分析”和“执行”的时间相差很大、数据相关、转移或转子程序指令

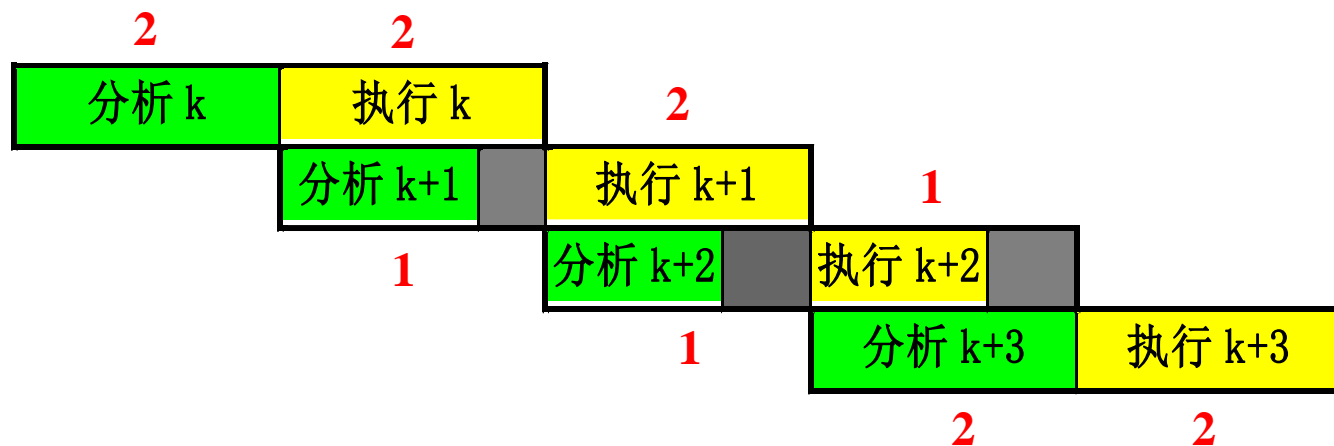


5.1.2 先行控制方式的原理

4. 指令执行时序:

- 设置了指令缓冲栈，取指令的时间就可以忽略不计。一条指令的执行可分为2个过程, 即指令分析和指令执行。

1) 分析指令和执行指令时间不相等时的情况



分析指令和执行指令时间不相等时的一次重叠执行方式

$$T = 2 \times 5 = 10$$

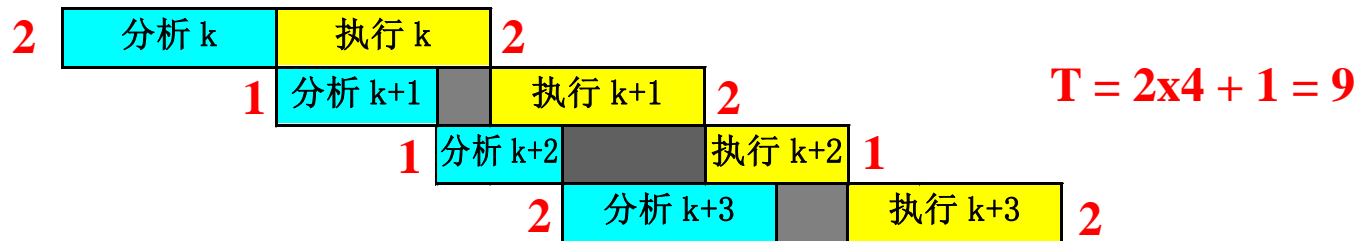
5.1.2 先行控制方式的原理

4. 指令执行时序:

➤ 设置了指令缓冲栈，取指令的时间就可以忽略不计。一条指令的执行可分为2个过程,即指令分析和指令执行。

- 1) 分析指令和执行指令时间不相等时的情况
- 2) 采用先行缓冲栈的指令执行过程

- 先行读数栈, 先行操作栈, 后行写数栈。



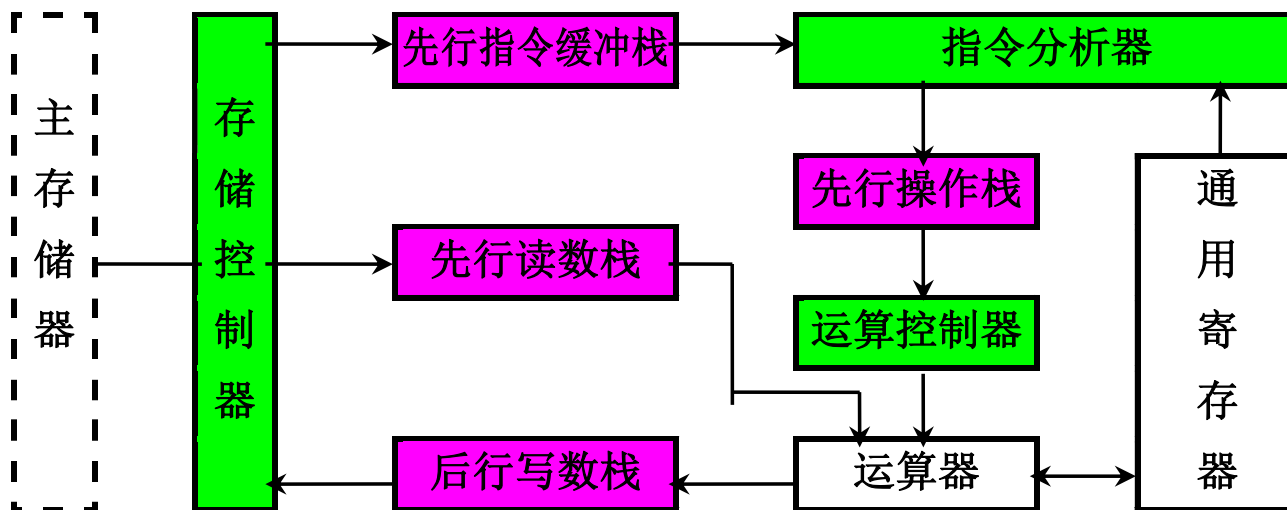
- 理想情况下，指令执行部件应该一直忙碌。
- 连续执行 n 条指令的时间为：
$$T_{\text{先行}} = t_{\text{分析}1} + \sum_{i=1}^n t_{\text{执行}i} \approx \sum_{i=1}^n t_{\text{执行}i}$$

5.1.2 先行控制方式的原理

5. 先行缓冲栈

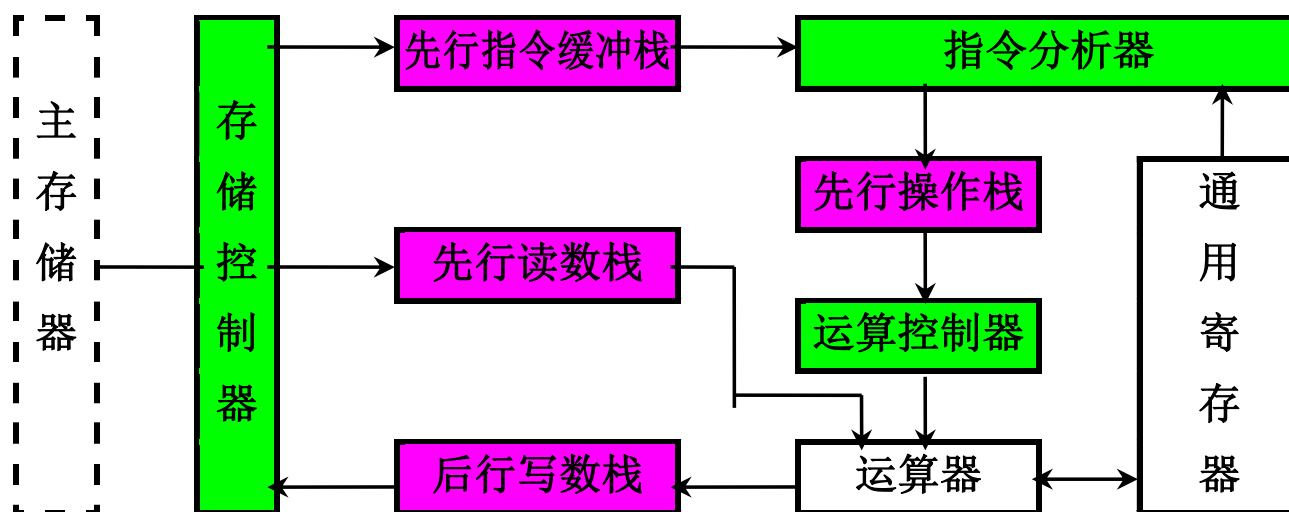
设置先行缓冲栈的目的：使指令分析器和指令执行部件能够独立工作。

1) **先行指令缓冲栈**：处于主存储器与指令分析器之间，用它来平滑主存储器取指令和指令分析器使用指令之间的速度差异



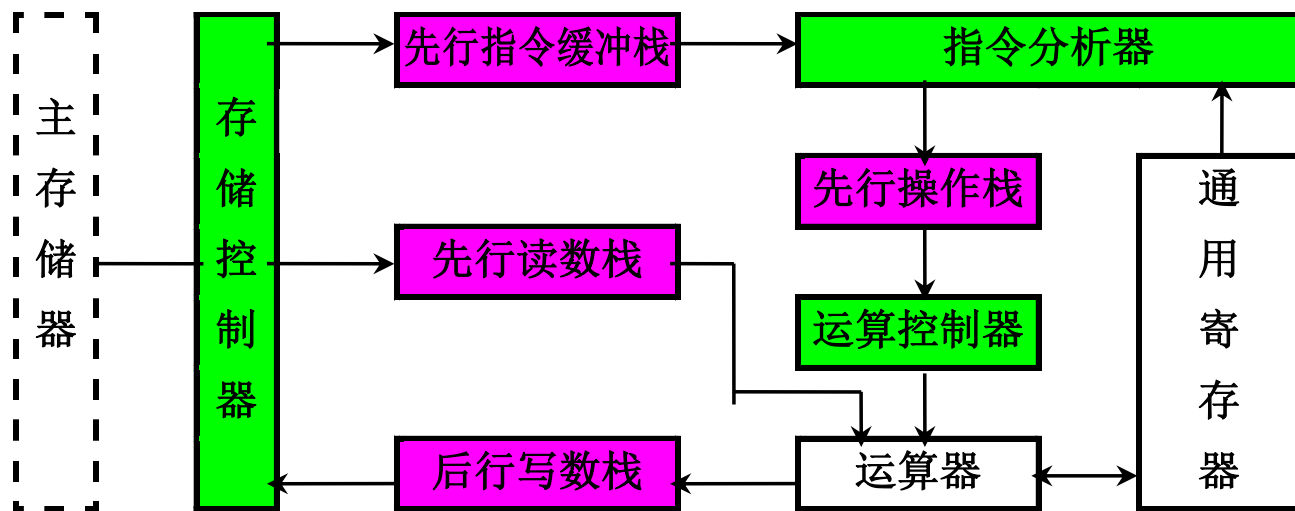
2) 先行操作栈：采用先进先出方式工作，由指令寄存器堆和控制逻辑组成

- 指令分析器对已经放在先行指令缓冲栈里的指令进行预处理，把处理之后的指令送入先行操作栈
- 处于指令分析器和运算控制器之间，使指令分析器和运算器能够各自独立工作



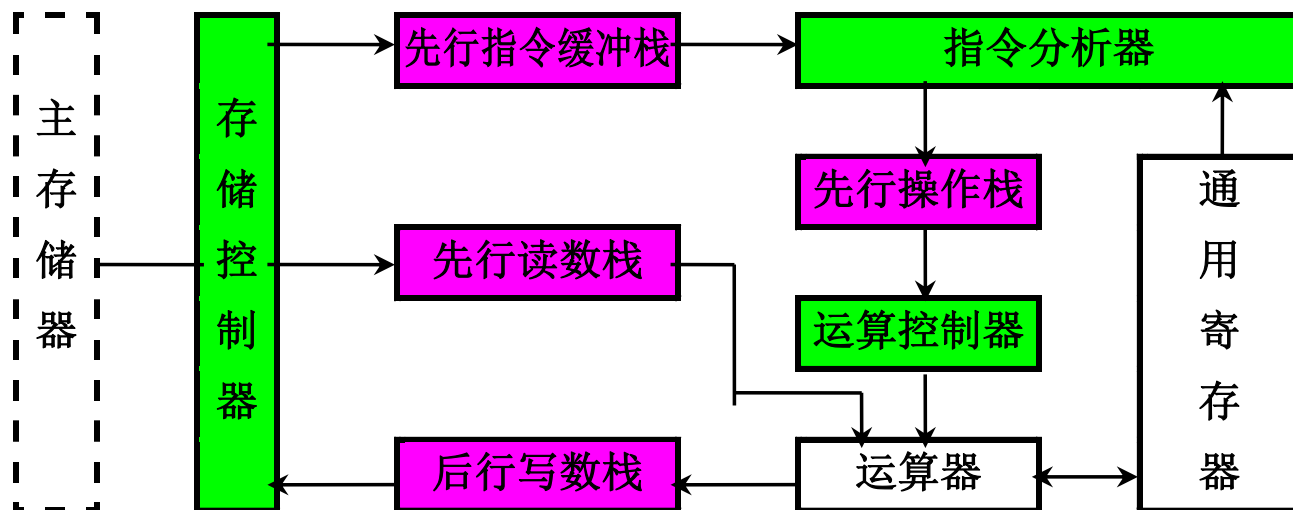
2) **先行操作栈**：采用先进先出方式工作，由指令寄存器堆和控制逻辑组成

- 对于无条件转移及条件转移等程序控制指令，一般在指令分析器中就能直接执行
- RR型指令，不必处理，直接送先行操作缓冲栈
- RS型或RX型指令，主存有效地址送先行读数栈，由先行读数栈负责到主存储器读取操作数；同时，用该先行读数栈的寄存器编号替换指令中的主存地址码部分，形成RR*指令送先行操作缓冲栈
- RI型指令，指令中的立即数送先行读数栈，用该先行读数栈的寄存器编号替换指令中的立即数部分，形成RR*指令送先行操作缓冲栈



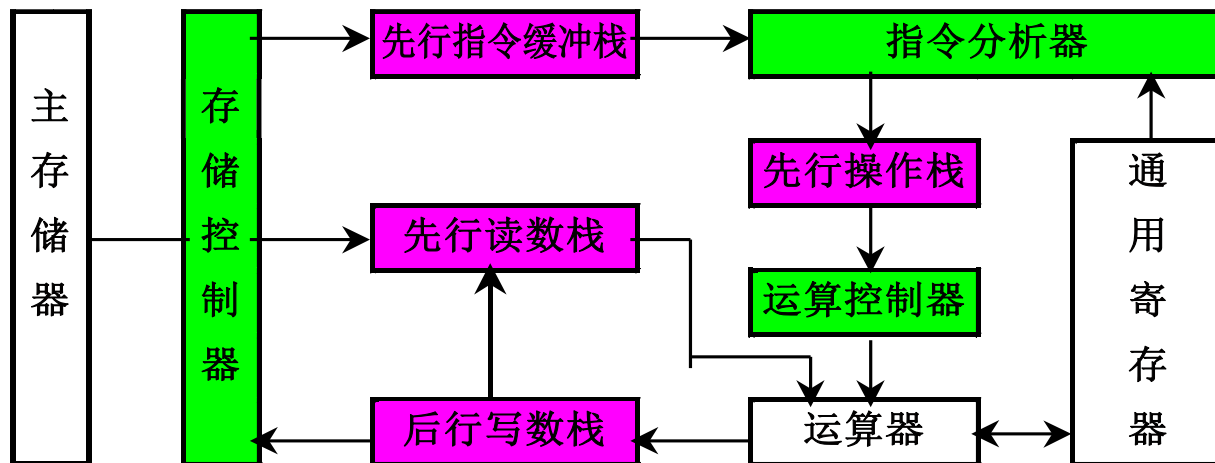
3) 先行读数栈:

- 由一组缓冲寄存器和有关控制逻辑等组成，处于主存储器与运算器之间，平滑运算器与主存储器的工作
- 每个缓冲寄存器由**地址寄存器**、**操作数寄存器**和**标志**三部分组成。也可以把地址寄存器和操作数寄存器合为一个。
- 当收到从指令分析器中送来的有效地址时，将地址的有效标志置位，向主存申请读操作数。
- 读出的操作数存放在操作数寄存器中或覆盖掉地址寄存器中的地址，置位数据有效标志



4) 后行写数栈

- 后行写数栈也由一组缓冲寄存器和有关控制逻辑等组成
- 每个后行缓冲寄存器由地址寄存器、数据寄存器和标志三部分组成。
- 指令分析器遇到向主存写结果的指令时，把形成的有效地址送入后行写数栈的地址寄存器中，并用该地址寄存器的编号替换指令的目的地址部分，形成RR*指令送入先行操作栈。
- 当运算器执行这条RR*型写数指令时，只要把写到主存的数据送到后行写数栈的数据寄存器中即可。



采用先行控制方式时一个程序的执行情况：

指令地址	指令执行情况
..... $k-i-1$	已经执行完成的指令
$k-i$ $k-1$	在后行写数栈中等待把结果写到主存储器中的指令
k	正在指令执行部件中执行的指令
$k+1$ $k+j$	已经由指令分析器预处理完成，存放在先行操作栈中的RR*型指令，指令所需要的操作数已经读到先行读数栈中
$k+j+1$ $k+j+n$	已经由指令分析器预处理完成，存放在先行操作栈中，指令所需要的操作数还没有读到先行读数栈中
$k+j+n+1$	正在指令分析器中进行分析和预处理的指令
$k+j+n+2$ $k+j+n+m$	已经从主存储器中预取到先行指令缓冲栈中的指令
$k+j+n+m+1$	还没有进入处理机的指令
.....	

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

以静态分析为主，通过模拟来确定缓冲深度。

1) 先行指令缓冲栈的设计

考虑两种极端情况：假设缓冲深度为 D_I

(1) 先行指令缓冲栈已经充满

- 指令流出的速度最快，例如连续分析RR型指令，设这种指令序列的最大长度为 L_1 ，平均分析一条这种指令的时间为 t_1 ；
- 指令流入的速度最慢，设平均取一条指令的时间为 t_2 。从主存储器中取到先行指令缓冲栈中的指令条数是 $L_1 - D_I$ 条。

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

以静态分析为主，通过模拟来确定缓冲深度。

1) 先行指令缓冲栈的设计

考虑两种极端情况：假设缓冲深度为 D_I

(1) 先行指令缓冲栈已经充满

- 应该满足如下关系： $L_1 t_1 = (L_1 - D_I) t_2$
- 计算出缓冲深度为： $D_I = \left\lceil \frac{L_1 \cdot (t_2 - t_1)}{t_2} \right\rceil$
- 如果这种指令流的连续长度超过 L_1 ，则先行指令缓冲栈失去作用。

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

以静态分析为主，通过模拟来确定缓冲深度。

1) 先行指令缓冲栈的设计

考虑两种极端情况：假设缓冲深度为 D_I

(2) 先行指令缓冲栈原来为空

- 输入端指令流入的速度最快，每次取指令的时间最短，设这种指令序列的最大长度为 L_2 ，平均取一条这种指令的时间为 t_2'
- 输出端指令流出的速度最慢，指令分析器连续分析最难分析的指令；设平均分析一条指令的时间为 t_1'

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

以静态分析为主，通过模拟来确定缓冲深度。

1) 先行指令缓冲栈的设计

考虑两种极端情况：假设缓冲深度为 D_I

(2) 先行指令缓冲栈原来为空

- 分析的指令条数是 $L_2 - D_1$ 条
- 应该满足如下关系： $(L_2 - D_1)t'_1 = L_2 t'_2$
- 计算出缓冲深度为： $D_I = \left\lceil \frac{L_2(t'_1 - t'_2)}{t'_1} \right\rceil$
- 如果这种指令流的连续长度超过 L_2 ，先行指令缓冲栈失去缓冲作用

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

一个采用先行控制方式的处理机，指令分析器分析一条指令用一个周期，到主存储器中取一条指令装入先行指令缓冲栈平均用4个周期，如果这种指令的平均长度为9。

解： 计算先行指令缓冲栈的缓冲深度为：

$$D_I = \left\lceil \frac{L_1 \cdot (t_2 - t_1)}{t_2} \right\rceil = \left\lceil \frac{9 \cdot (4 - 1)}{4} \right\rceil = 7$$

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

2) 先行指令缓冲栈的工作时间关系

工作周期	1	2	3	4	5	6	7	8	9	10
指令分析器的指令序列	k+1	k+2	k+3	K+4	K+5	K+6	K+7	K+8	k+9	空
向主存请求取指令	↑				↑				↑	
指令取到先行缓冲栈中										
缓冲栈中剩余指令条数	6	5	4	4*	3	2	1	1*	0	0

- 第1个周期，取走指令k+1，请求取指令
- 第4个周期末尾，指令k+8取到先行指令缓冲栈
- 第8个周期末尾，指令k+9取到先行指令缓冲栈
- 第9个周期，分析指令k+9，先行指令缓冲栈空
- 第10个周期，指令分析器等待

5.1.2 先行控制方式的原理

6. 缓冲深度的设计方法

3) 其余缓冲栈的设计原则

- 一般有关系： $D_I \geq D_C \geq D_R \geq D_W$

其中： D_I 是先行指令缓冲栈的缓冲深度，

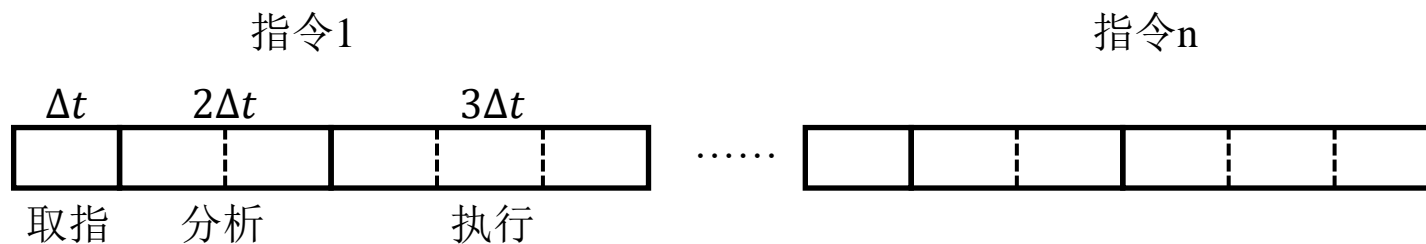
D_C 是先行操作栈的缓冲深度，

D_R 是先行读数栈的缓冲深度，

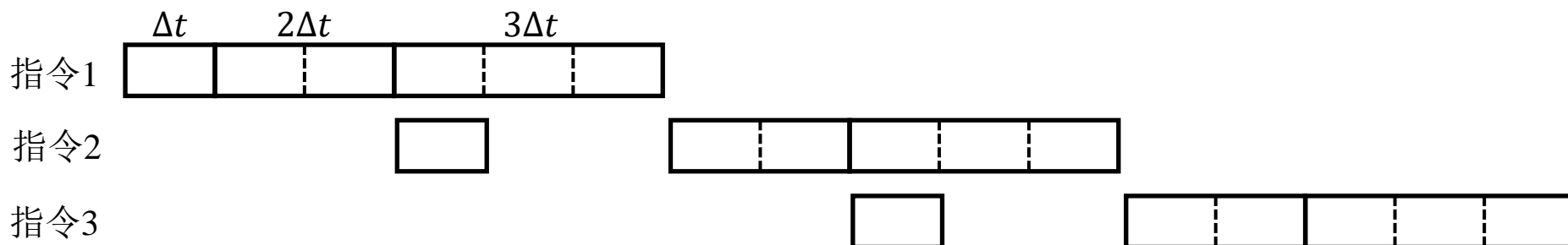
D_W 是后行写数栈的缓冲深度。

- IBM370/165机： $D_I=4$ ， $D_C=3$ ， $D_R=2$ ， $D_W=1$ 。
- 我国研制的两台大型计算机： $D_I=8$ ， $D_C=D_R=4$ ， $D_W=2$ ； $D_I=12$ ， $D_C=D_R=6$ ， $D_W=2$ 。

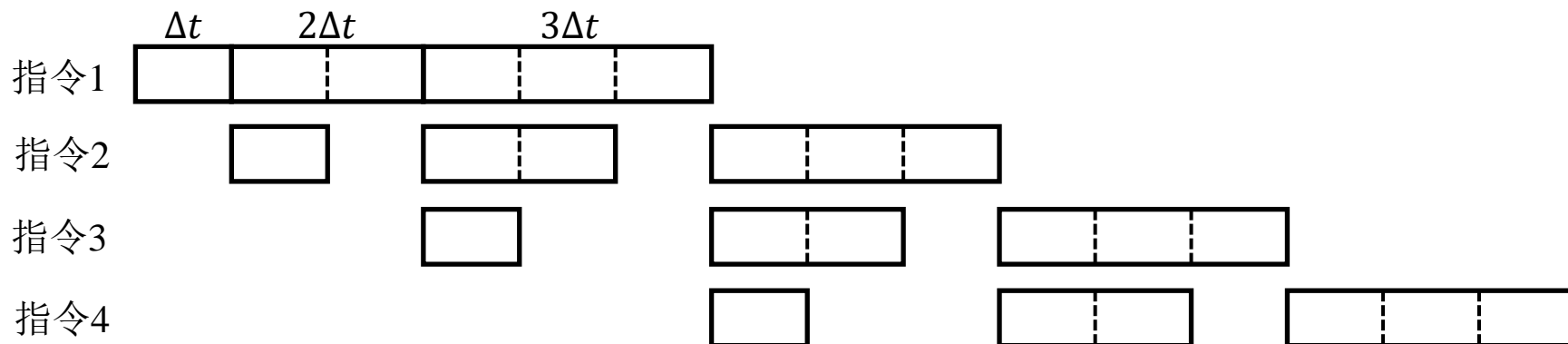
- 假设一条指令的执行过程分别为“取指令”、“分析”、“执行”三段，每一段的时间分别是 Δt 、 $2\Delta t$ 、 $3\Delta t$ 。在下列情况下，分别写出连续执行 n 条指令所需要的时间表达式
 - 1) 顺序执行方式
 - 2) 仅“取指令”和“执行”重叠
 - 3) “取指令”、“分析”和“执行”重叠
 - 4) 先行控制方式



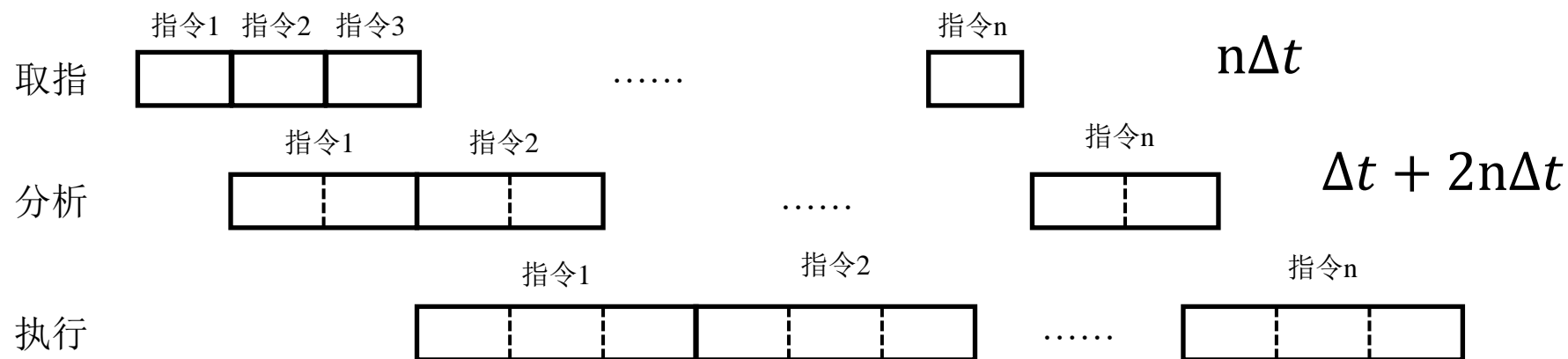
$$(1) T = n(\Delta t + 2\Delta t + 3\Delta t) = 6n\Delta t$$



$$(2) T = 6\Delta t + (n - 1)(2\Delta t + 3\Delta t) = (5n + 1)\Delta t$$



$$(3) T = 6\Delta t + (n - 1)3\Delta t = (3n + 3)\Delta t$$



$$(4) T = 3\Delta t + 3n\Delta t = (3n + 3)\Delta t$$


5.1.3 数据相关

- 数据相关：在执行本条指令的过程中，如果用到的指令、操作数、变址量等是前面指令的执行结果，这种相关称为数据相关。
- 控制相关：由条件分支指令、转子程序指令、中断等引起的相关。
- 解决数据相关的方法有两种：推后处理, 设置专用路径。


5.1.3 数据相关

1. 指令相关

➤ 发生指令相关的情况：

n : STORE  $n+1$ **ADD R2, R3**

$n+1$: **SUB R2, R3**



满足关系： 结果地址(n) = 指令地址($n+1$)

➤ 当第 n 条指令还没有把执行结果写到主存之前，取出的第 $n + 1$ 条指令显然是错误的。

5.1.3 数据相关

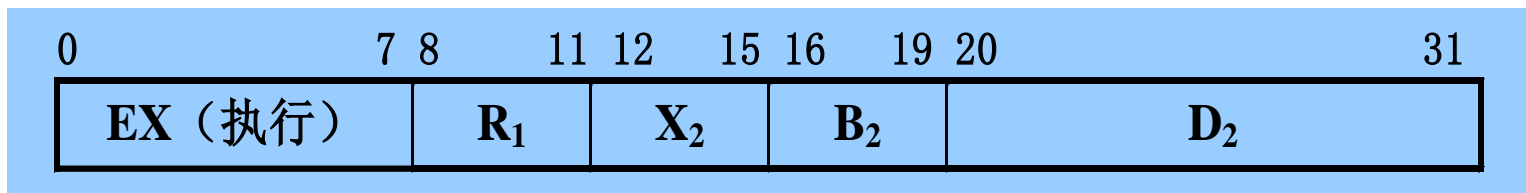
1. 指令相关

- 在采用先行控制方式的处理机中，如果执行部件正在执行第 n 条指令，与下述情况之一发生相关，都可能造成程序执行结果发生错误。
- 存放在先行操作栈中的指令
 - 正在指令分析器中分析的指令
 - 已经预取到先行指令缓冲栈中的指令
- 更严重的是：有些分支指令，可能已经在指令分析器中执行完成。

5.1.3 数据相关

1. 指令相关

- 解决指令相关的根本办法是：在程序执行过程中不允许修改指令。
- 现代程序设计方法要求程序具有再入性，可以被递归调用等，也要求不修改指令。
- 在程序执行过程中不允许修改指令也有利于程序的调试和诊断
- 在IBM370系列机中，用“执行指令”来解决：在程序执行过程中既能够修改指令，程序又具有再入性。
 - “执行指令”执行由第二地址 $((X2) + (B2) + D2)$ 决定的主存数据区中的指令。



5.1.3 数据相关

2. 主存操作数相关

➤ 发生主存操作数相关的指令序列：

$n: OP \ A_1, A_2, A_3 \quad ; \ A_1 = (A_2) OP (A_3)$

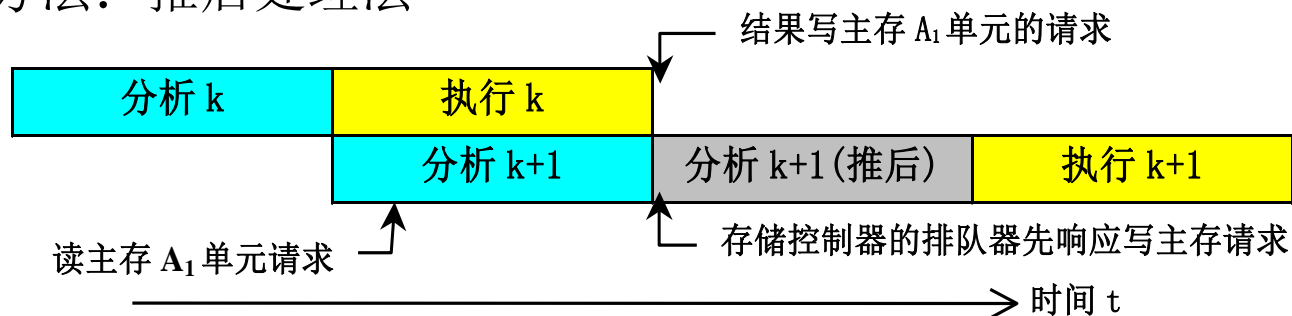
$n+1: OP \ A_4, A_1, A_5 \quad ; \ A_4 = (A_1) OP (A_5)$

➤ 出现下列情况之一，就发生主存操作数相关：

$RES(n) = OPREND\#1(n+1)$

$RES(n) = OPREND\#2(n+1)$

➤ 解决办法：推后处理法



➤ 先行操作栈

5.1.3 数据相关

3. 通用寄存器数据相关

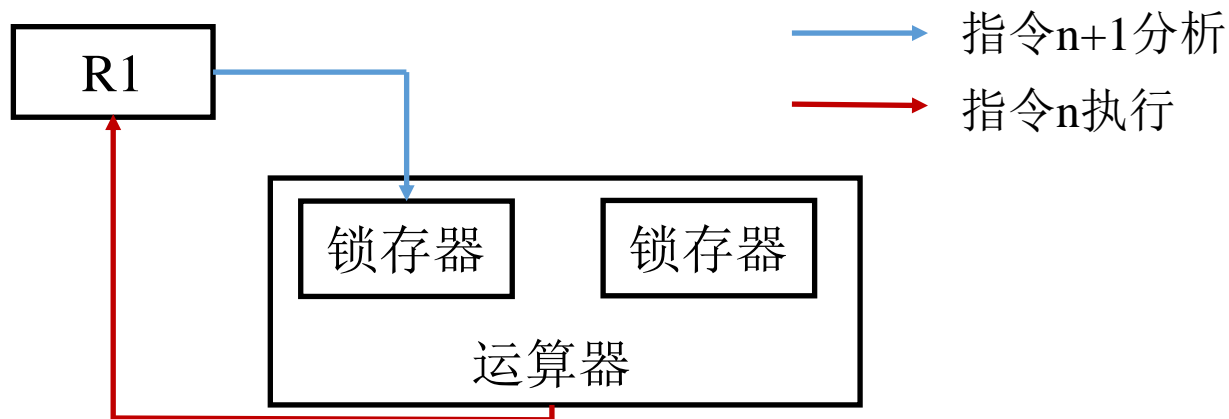
➤ 发生寄存器数据相关的可能性很大，影响面也很大

n: OP R_1 , A_2 ; $R_1 = (R_1)$ OP (A_2)

n+1: OP R_1 , R_2 ; $R_1 = (R_1)$ OP (R_2)

➤ 发生 $R_1(n) = R_1(n+1)$ 称为 R_1 数据相关。

➤ 发生 $R_1(n) = R_2(n+1)$ 称为 R_2 数据相关。

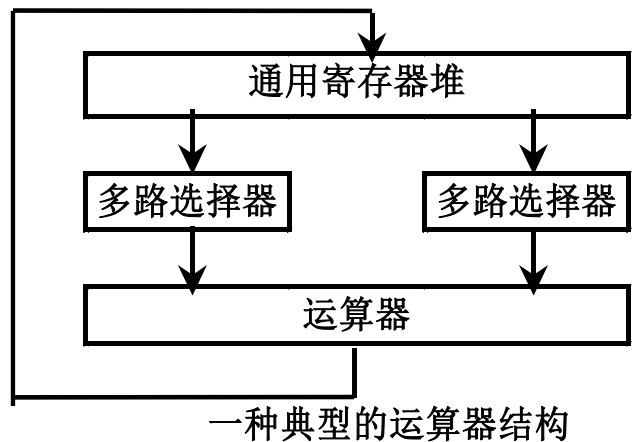


5.1.3 数据相关

3. 通用寄存器数据相关

➤ 解决通用寄存器数据相关的方法：

方法一：如果通用寄存器用D型触发器构成，而且在通用寄存器到运算器之间建立有直接数据通路（即不设置缓冲寄存器或锁存器），则不会发生通用寄存器数据相关，因为D型触发器允许在同一节拍中实现寄存器之间的循环传送

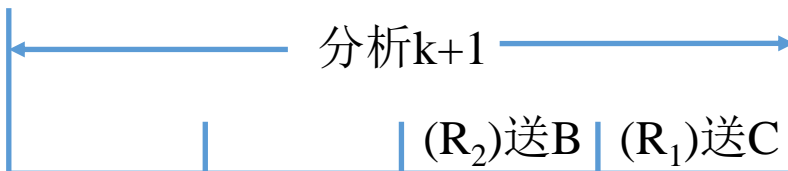
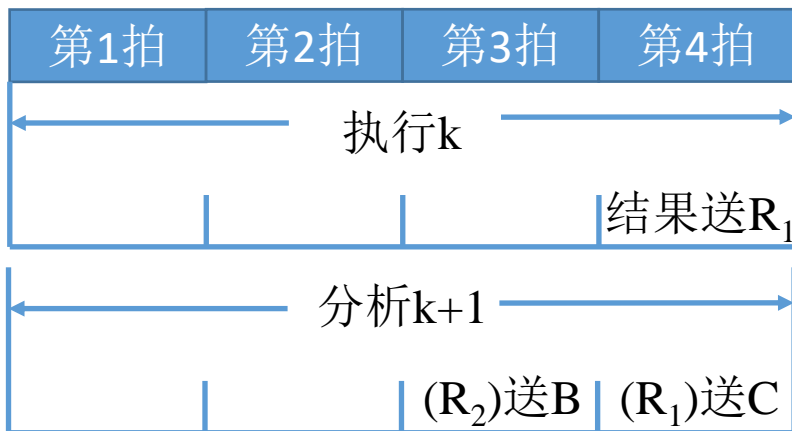


5.1.3 数据相关

3. 通用寄存器数据相关

➤ 解决通用寄存器数据相关的方法：

方法二：分析指令推后一个周期执行

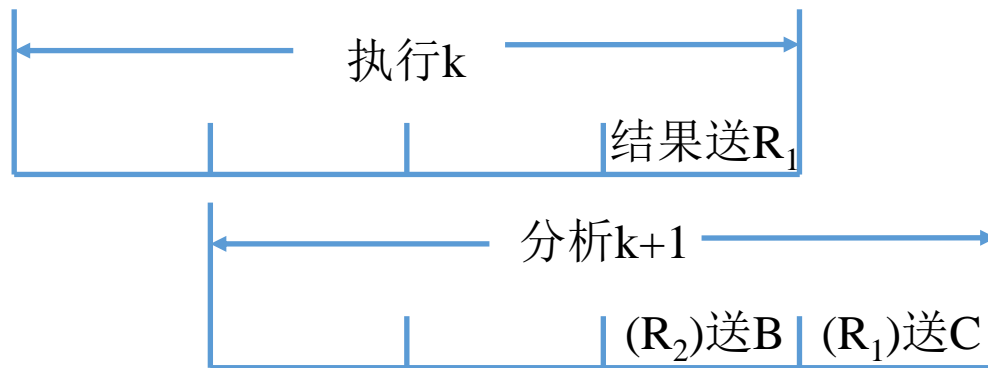


5.1.3 数据相关

3. 通用寄存器数据相关

➤ 解决通用寄存器数据相关的方法：

方法三：分析指令推后一个节拍

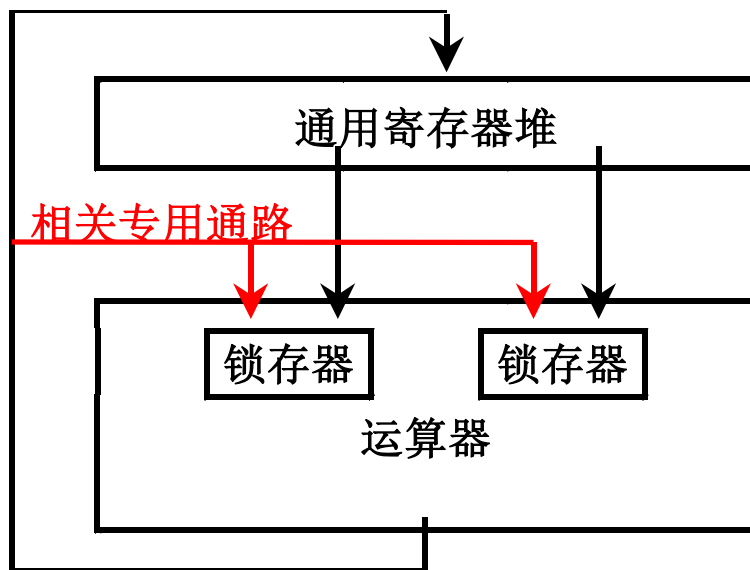


5.1.3 数据相关

3. 通用寄存器数据相关

➤ 解决通用寄存器数据相关的方法：

方法四：设置专用通路



设置专用数据通路解决通用寄存器数据相关

5.1.3 数据相关

4. 变址相关

- 在采用变址寻址方式的处理机中，由于变址量放在寄存器中，因此，可能发生与通用寄存器数据相关类似变址相关

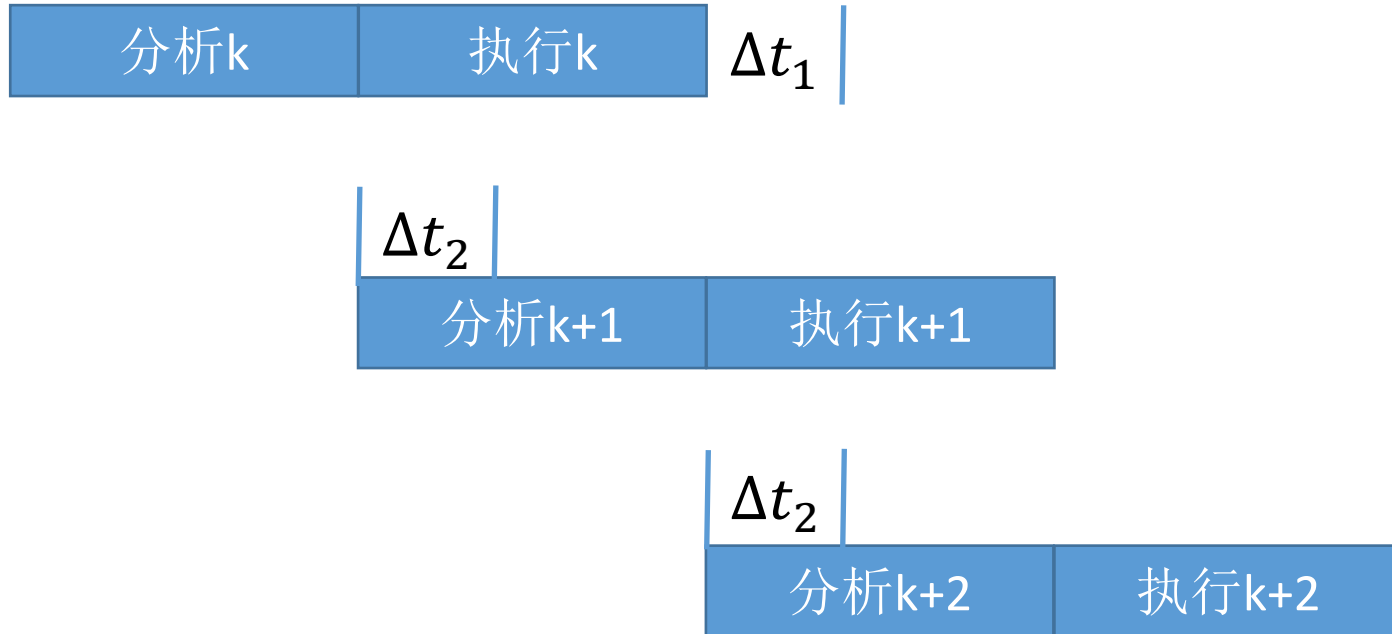
$$k: \quad \text{OP } R_1, R_2 \quad ; \quad R_1 = (R_1) \text{ OP } (R_2)$$

$$k+1: \quad \text{OP } R_1, A_2(X_2); \quad R_1 = (R_1) \text{ OP } ((A_2) + (X_2))$$

$$k+2: \quad \text{OP } R_1, A_2(X_2); \quad R_1 = (R_1) \text{ OP } ((A_2) + (X_2))$$

- 一次变址相关: $R_1(k) = X_2(k+1)$
- 二次变址相关: $R_1(k) = X_2(k+2)$

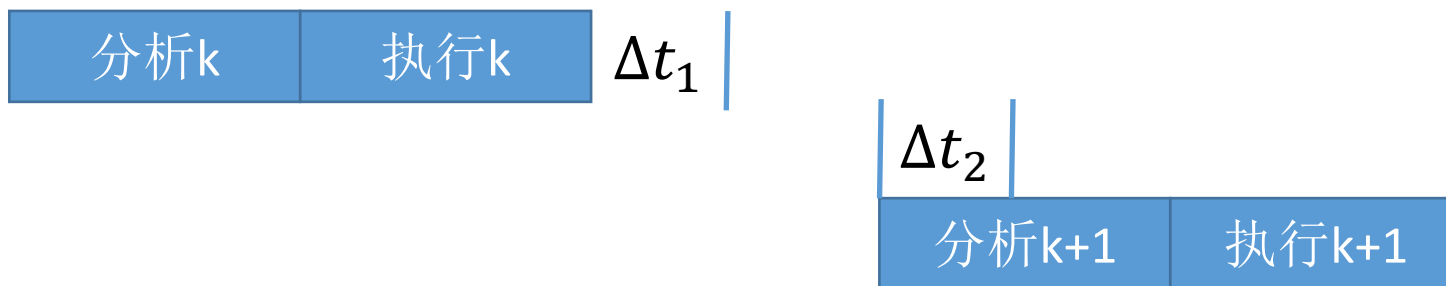
若 $\Delta t_1 > \Delta t_2$ ，则“分析k+2”计算出来的有效地址将是错误的



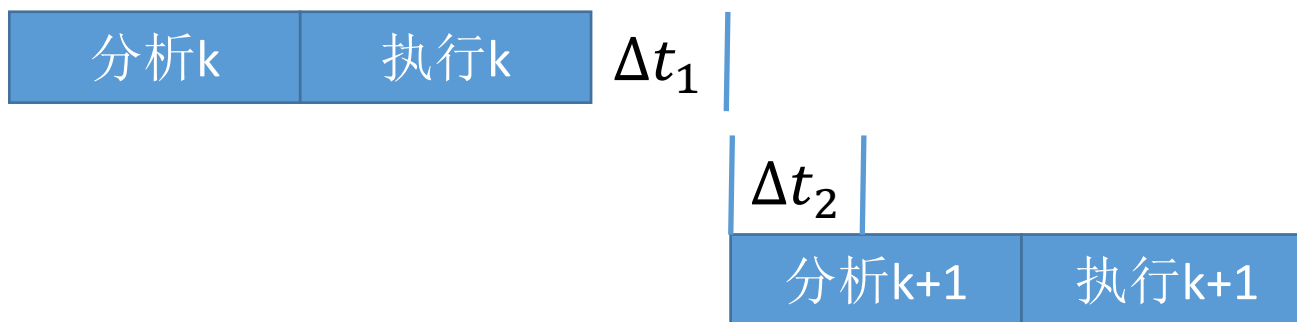
Δt_1 : 数据写入寄存器后的稳定时间

Δt_2 : 指令译码，计算变址地址前

- 方法一：推后分析。对于一次变址相关，把“分析k+1”推后两个周期或者一个周期再加一个节拍

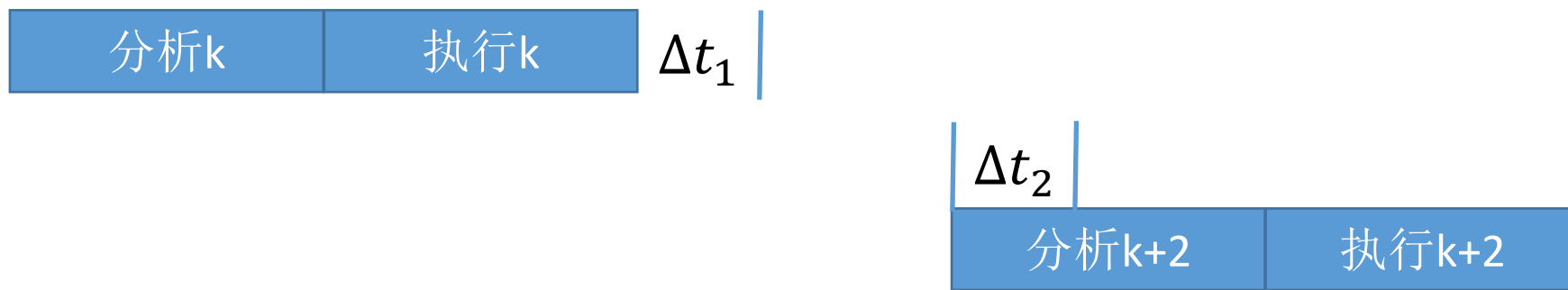


推后两个周期

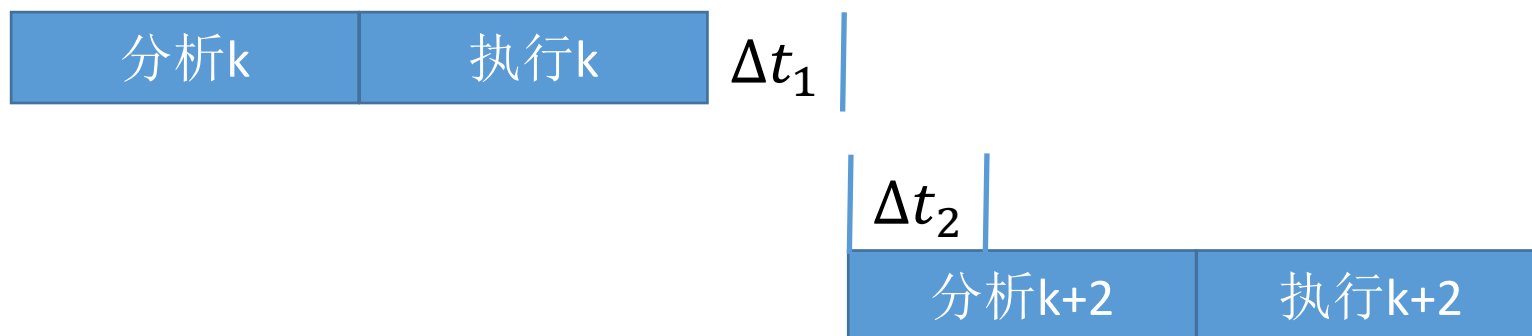


推后一个周期再加一个节拍

- 方法一：推后分析。对于二次变址相关，需要把“分析k+2”推后一个周期或者节拍

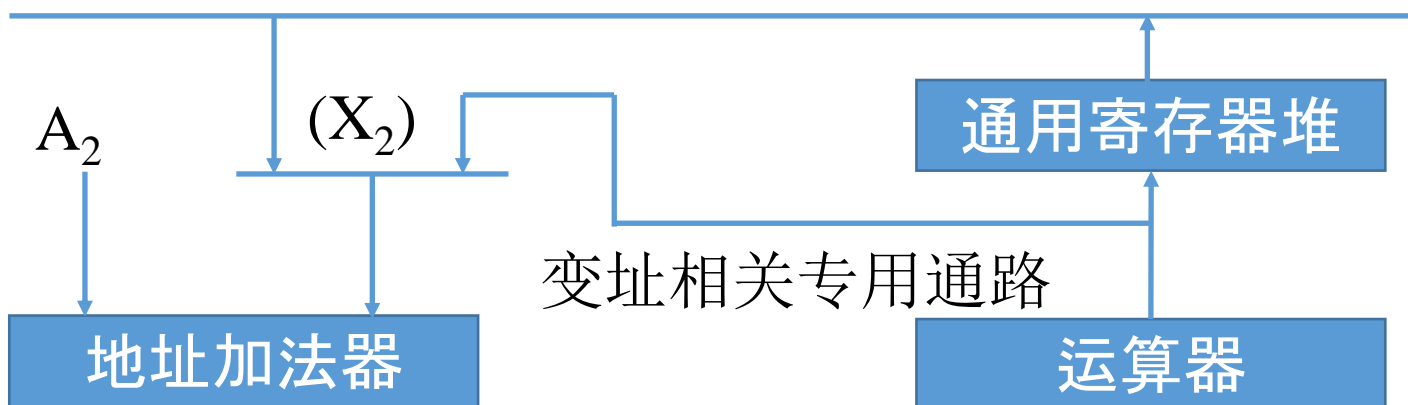


推后一个周期



推后一个节拍

- 方法一：推后分析。对于二次变址相关，需要把“分析k+2”推后一个周期或者节拍；对于一次变址相关，把“分析k+1”推后两个周期或者一个周期再加一个节
- 方法二：设置变址相关专用通路



5.1.4 控制相关

- 因程序的执行方向可能被改变而引起的相关，也称为全局相关。
- 主要包括：无条件转移、一般条件转移、复合条件转移、中断等。

5.1.4 控制相关

1. 无条件转移

k: ...

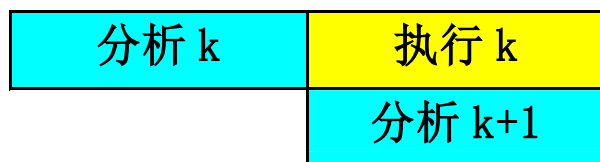
k+1: JMP L

...: ...

L: ...

- 在流水线处理机中，无条件转移指令不进入执行流水段，一般在指令译码阶段就实际执行完成。
- 如果在处理机中设置有指令先行缓冲栈，则要全部或部分作废先行指令缓冲栈中的指令。

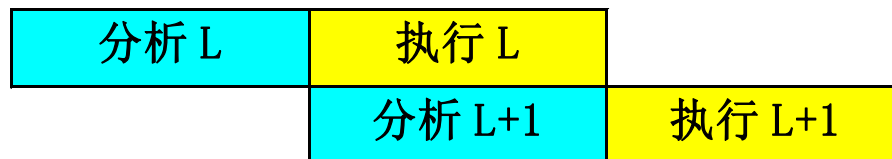
- 如果转移目标指令L不在先行指令缓冲栈中，则要将先行指令缓冲栈中的所有指令全部作废，并等待取出转移目标指令L。
- 如果转移目标指令L在先行指令缓冲栈中，只要作废先行指令缓冲栈中的部分指令。



指令 L 不在先行指令缓冲栈



指令 L 在先行指令缓冲栈中：



无条件转移指令的执行时序

- 无条件转移指令一般对指令执行部件的工作不会造成影响。
- 为进一步减少无条件转移指令造成的影响，在先行指令缓冲栈的入口处增设一个专门处理无条件转移指令的指令分析器

5.1.4 控制相关

2. 一般条件转移

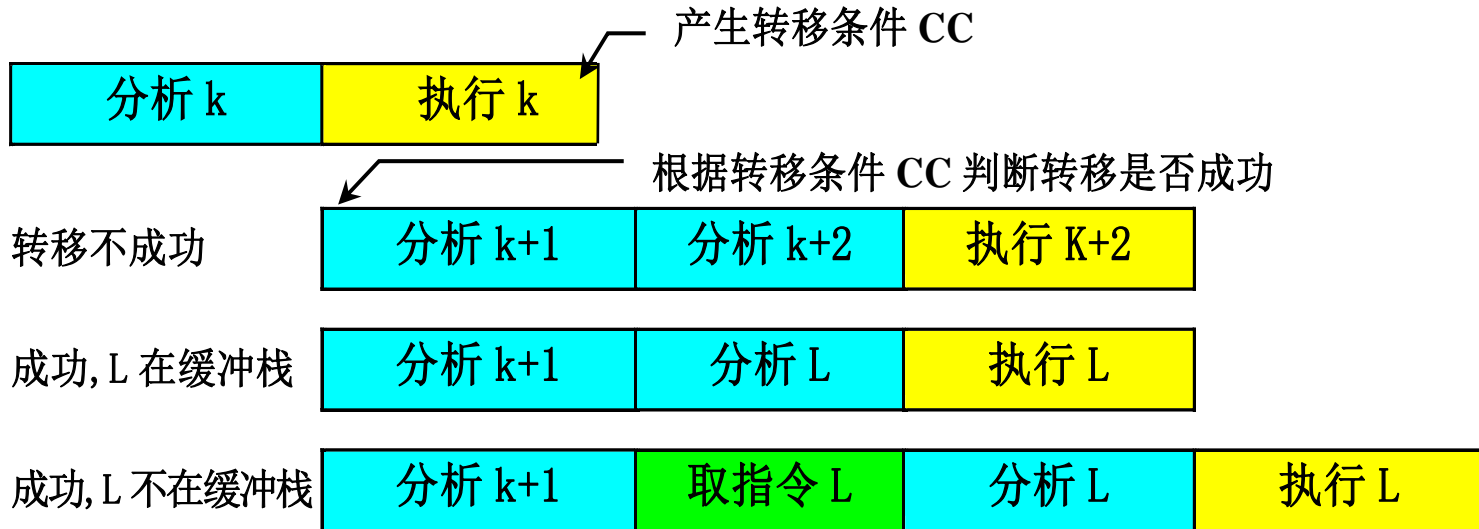
k: ;置条件码CC

k+1: JMP (CC) L ;如果CC为真转向L

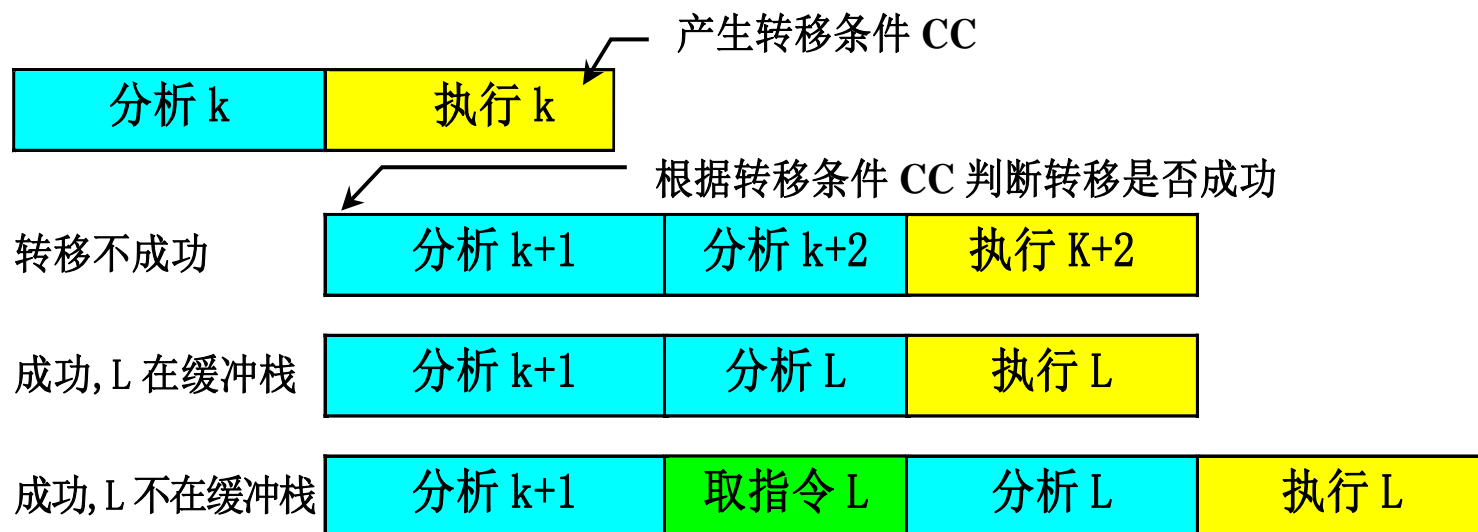
.....

L:

- 当条件码是上一条指令产生时，相关最严重



- 无论转移是否成功，条转移指令都在指令分析阶段就已经执行完成。
- 无论转移不成功或不成功，指令分析器要停顿一段时间，等待条件码产生。



- 如果转移成功：指令L已经在先行指令缓冲栈，指令分析器接着“分析L”，如果指令L不在先行指令缓冲栈，指令分析器要等待一个周期。
- 转移不成功，对程序执行影响不大，
- 当转移成功时，不仅指令执行过程变成完全串行，而且要作废先行指令缓冲栈中的大量指令。
- 在采用流水线方式的处理机中，要通过软件与硬件的多种手段来尽可能地降低转移成功的概率，减少转移成功造成的影响。

5.1.4 控制相关

3. 复合条件转移

k: OP L ;产生条件码，并决定是否转向L

.....

L:

- 如果转移不成功：不造成任何影响，就象普通的运算型指令一样
- 如果转移成功：造成的影响比一般条件转移指令还要大得多。全部或部分作废先行指令缓冲栈、先行操作栈、先行读数栈和指令分析器中的指令。
- 必须采取策略，减小转移成功造成的影响。

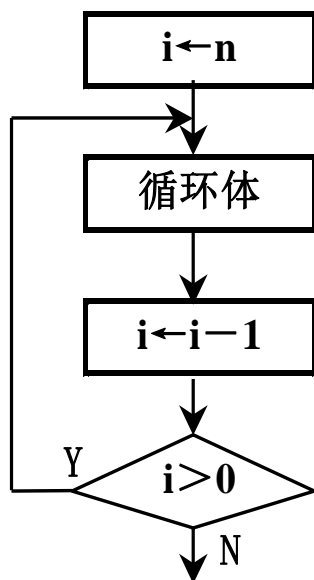
5.1.4 控制相关

4 转移预测技术

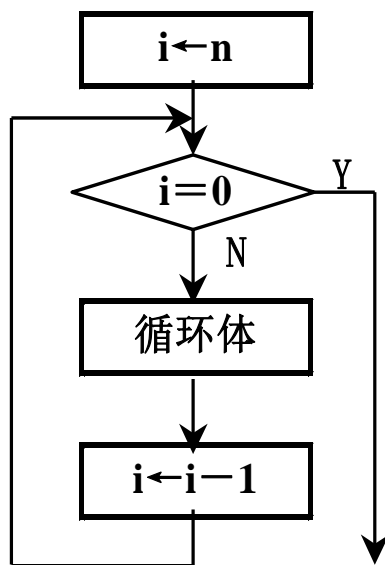
1) 软件“猜测法”

目标：通过编译器尽量降低转移成功的概率。

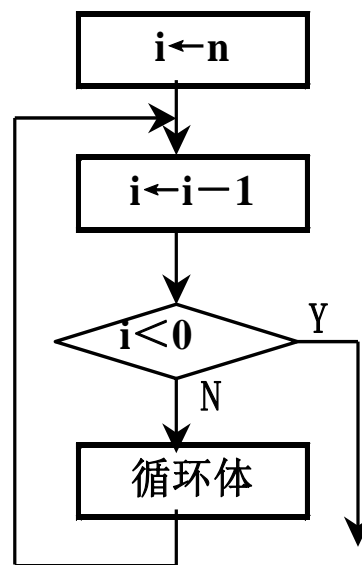
例如：对于循环程序，普通编译器生成的目标代码，转移成功的概率很高，不成功的只有一次。这种编译结果对流水线极为不利。



(a) 原来程序



(b) 一般条件转移



(c) 复合型条件转移

5.1.4 控制相关

4 转移预测技术

2) 硬件“猜测法”

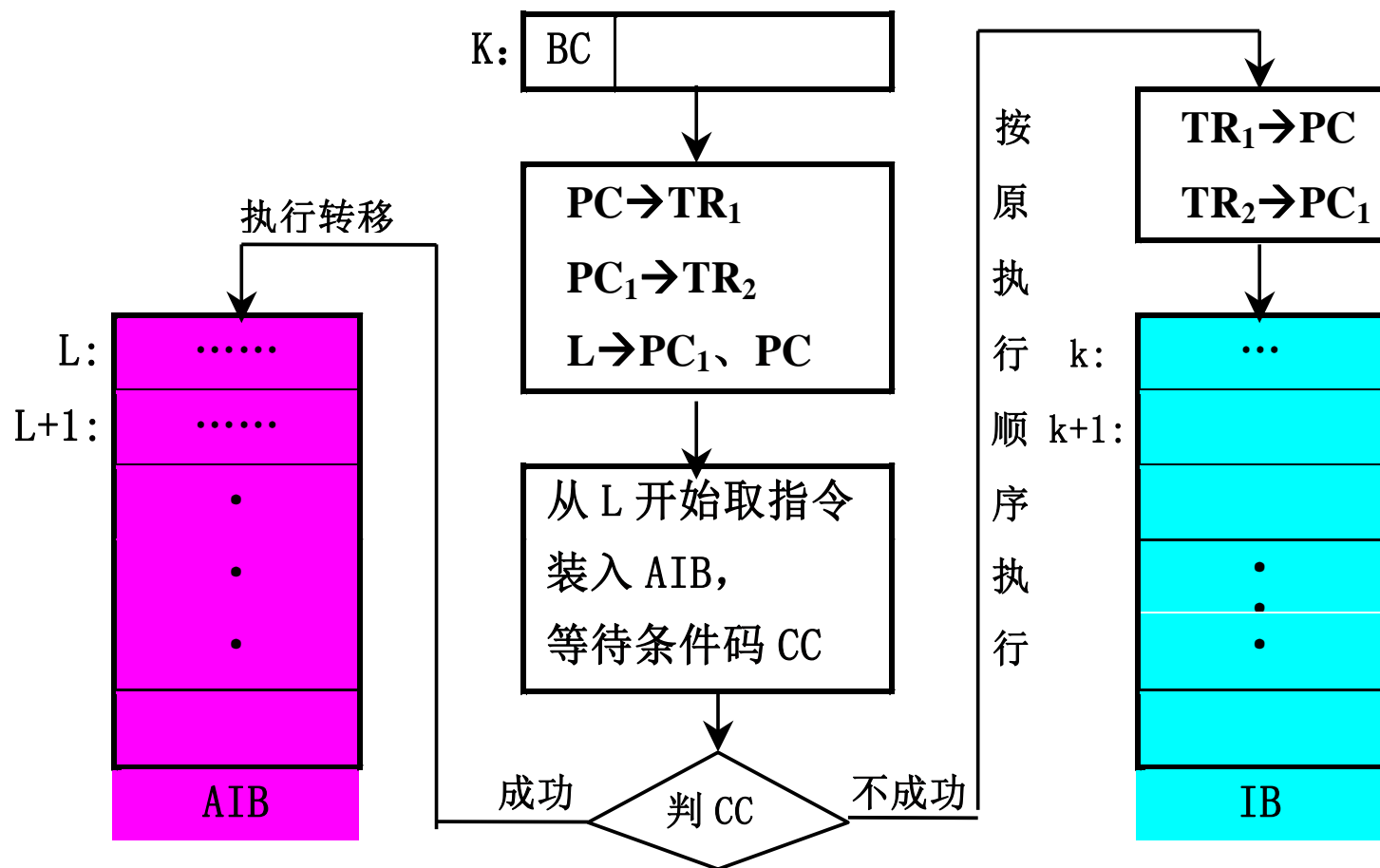
- 方法：通过改变硬件结构来降低转移指令对流水线的影响
- 在先行指令缓冲栈的人口处设置一个简单的指令分析器，当检测到转移指令时，就把转移目标地址L送入先行程序计数器 PC_1 中，同时保留当前 PC_1 中的内容到另一寄存器中。
- 转移成功，猜测正确。对转移指令对流水线不造成影响。
- 转移不成功，用保存下来的地址恢复 PC_1 和PC，清楚先行指令缓冲栈、先行操作栈和先行读数栈，重新开始取指令

5.1.4 控制相关

4 转移预测技术

3) 两个先行指令缓冲栈

- 普通条件转移指令（如“IF”语句），转移成功与不成功各50%
- 在先行指令缓冲栈中增加一个先行目标缓冲栈
- 按照转移成功的方向预取指令到先行目标缓冲栈中。
- 先行指令缓冲栈仍然按照转移不成功的方向继续预取指令。
- 如果转移不成功，则继续分析原来先行指令缓冲栈中指令。
- 如果转移成功，则分析新增设的先行目标缓冲栈中的指令。



两个先行指令缓冲栈的条件转移指令执行流程

5.2 流水线技术

5.2.1 流水线工作原理

5.2.2 流水线的分类

5.2.3 线性流水线的性能分析

5.2.4 非线性流水线的调度

5.2.5 局部相关

5.2.6 全局相关

5.2.1 流水线工作原理

- 流水线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他子过程同时进行
- 处理机各个部分几乎都可以采用流水线方式工作
 - 指令的执行过程可以采用流水线，称为“指令流水线”
 - 运算器中的操作部件（如浮点加法器、浮点乘法器等）可以采用流水线，称为“操作部件流水线”
 - 访问主存储部件也可以采用流水线
 - 甚至处理机之间，也可以采用流水线

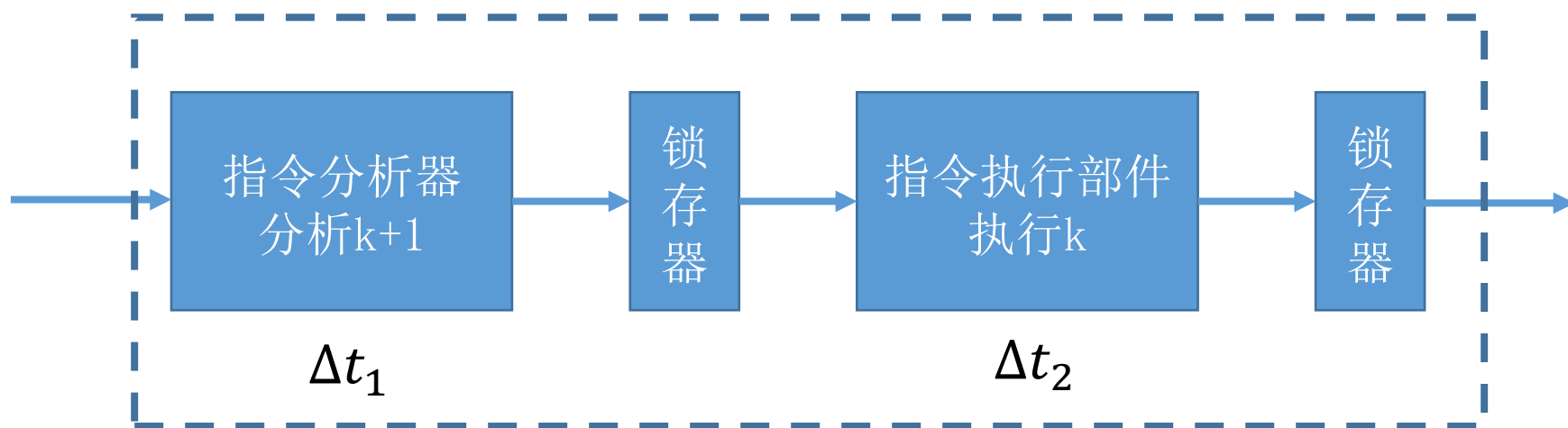
5.2.1 流水线工作原理

➤ 流水寄存器

- 流水线的每一个阶段称为流水步、流水步骤、流水段、流水线阶段、流水功能段、功能段、流水级、流水节拍等。
- 在每一个流水段的末尾或开头必须设置一个寄存器，称为流水寄存器、流水锁存器、流水闸门寄存器等。
- 加入流水寄存器，会增加指令的执行时间。
- 在一般流水线时空图中不画出流水寄存器。

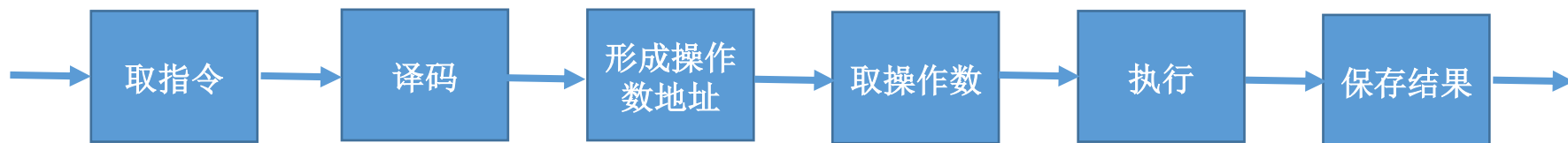
5.2.1 流水线工作原理

➤ 一种简单的流水线



5.2.1 流水线工作原理

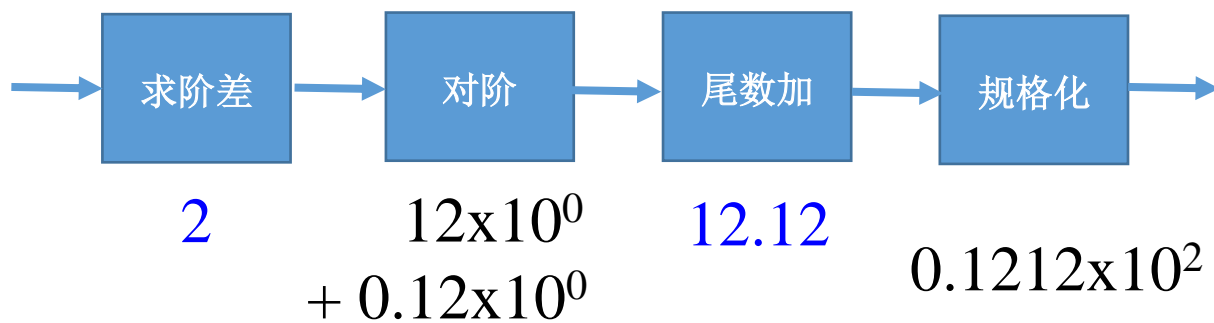
➤ 一种流水线



5.2.1 流水线工作原理

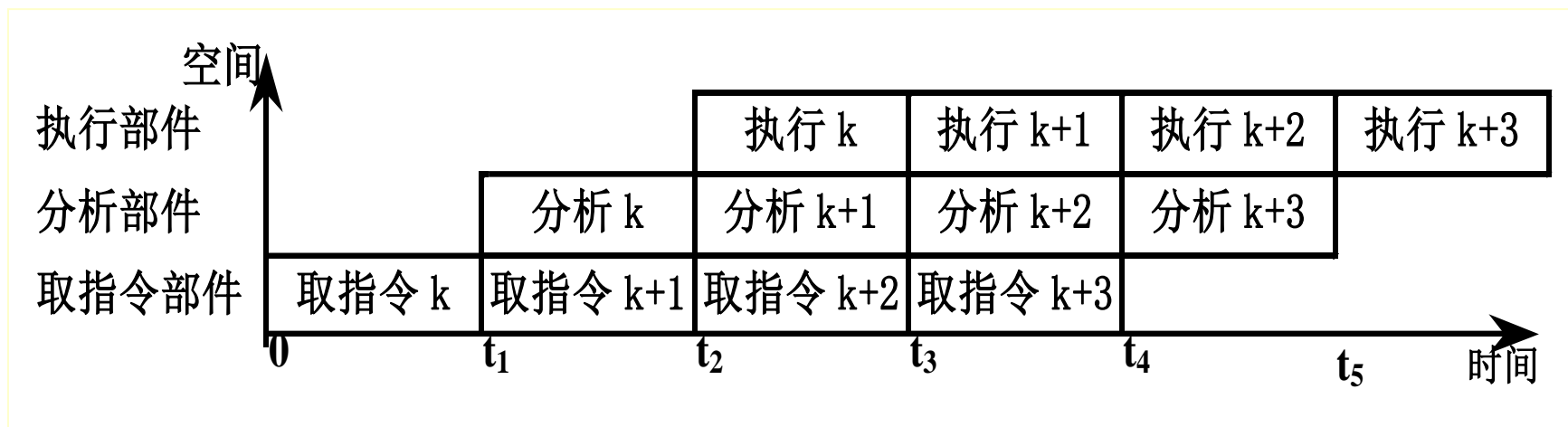
➤ 浮点加法器流水线

$$0.12 \times 10^2 + 0.12 \times 10^0$$



5.2.1 流水线工作原理

➤ 流水线时空图



5.2.1 流水线工作原理

➤ 一个浮点加法器流水线时空图



5.2.1 流水线工作原理

➤ 流水线的主要特点:

□ 只有连续提供同类任务才能发挥流水线效率

- 尽量减少因条件分支造成的“断流”
- 通过编译技术提供连续的相同类型操作

□ 每个流水线段都要设置一个流水寄存器

- 时间开销：流水线的执行时间加长
- 是流水线中需要增加的主要硬件

□ 各流水段的时间应尽量相等

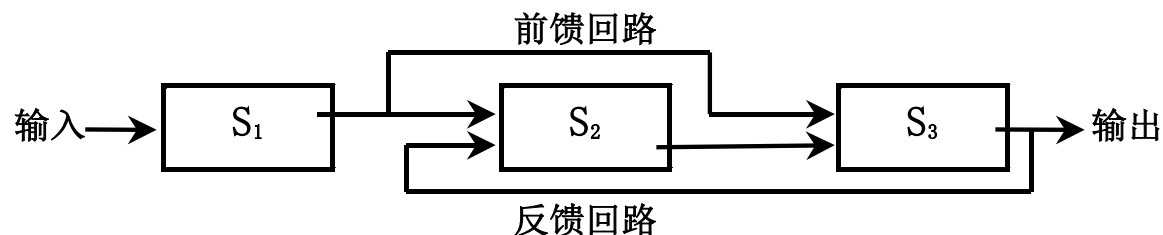
- 流水线处理机的基本时钟周期等于时间最长的流水段的时间长度。

□ 流水线需要有“装入时间”和“排空时间”

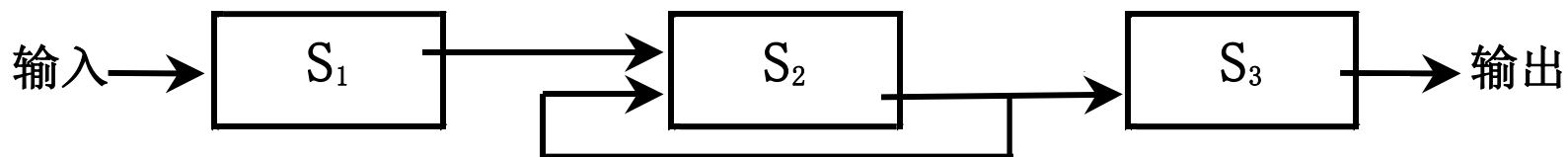
5.2.2 流水线的分类

1. 线性流水线与非线性流水线

- 流水线的各个流水段之间是否有反馈信号
- 线性流水线** (Linear Pipelining)：每一个流水段都流过一次，而且仅流过一次
- 非线性流水线** (Nonlinear Pipelining)：某些流水段之间有反馈回路或前馈回路。
- 线性流水线能够用流水线连接图唯一表示
- 非线性流水线必须用流水线连接图和流水线预约表共同表示



一种简单的非线性流水线



反馈回路

一种简单的非线性流水线

流水线连接图

时间

功能段				
	1	2	3	4
	S_1			
	S_2			
	S_3			
	×			
		×	×	
				×

流水线预约表

5.2.2 流水线的分类

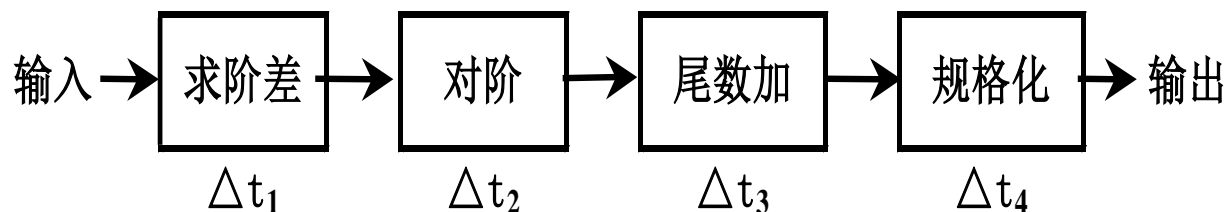
2. 按照流水线的级别来分

- 处理机级流水线，又称为指令流水线。
- 部件级流水线(操作流水线)
- 宏流水线(Macro Pipelining)

5.2.2 流水线的分类

2. 按照流水线的级别来分

- 处理机级流水线，又称为指令流水线。
- 部件级流水线(操作流水线)
 - 如浮点加法器流水线

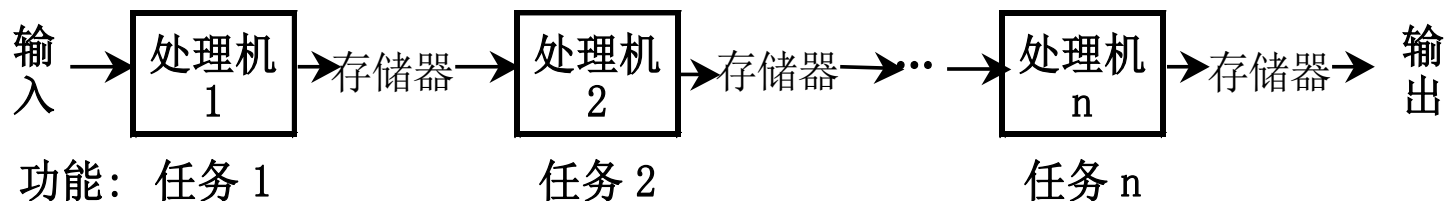


- 宏流水线(Macro Pipelining)

5.2.2 流水线的分类

2. 按照流水线的级别来分

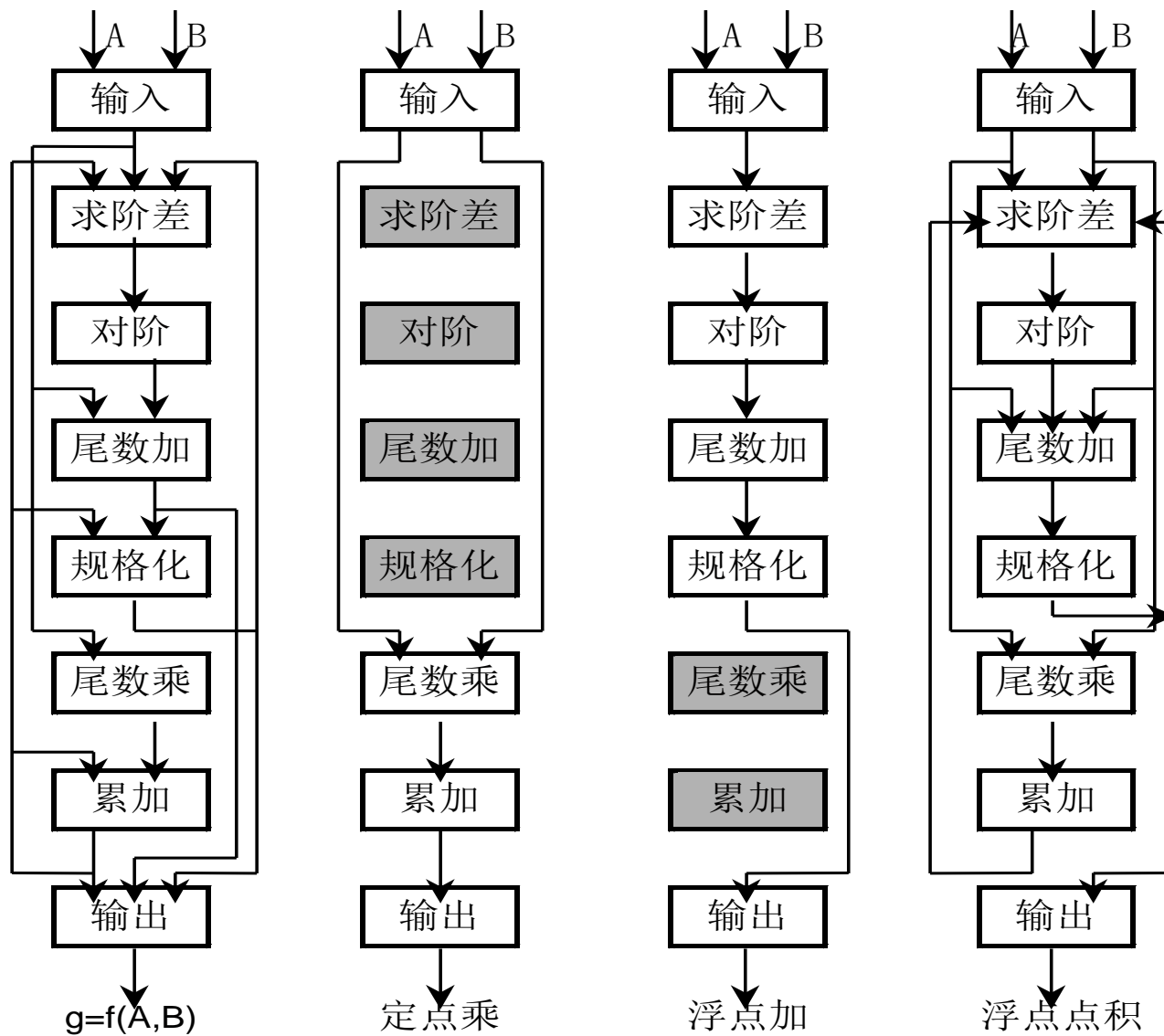
- 处理机级流水线，又称为指令流水线。
- 部件级流水线(操作流水线)
- 宏流水线(Macro Pipelining)
 - 处理机之间的流水线称，每个处理机对同一个数据流的不同部分分别进行处理。



5.2.2 流水线的分类

3. 单功能流水线与多功能流水线

- 单功能流水线（Unifunction Pipelining）：只能完成一种固定功能的流水线。
 - Cray-1计算机种有12条，YH-1计算机有18条
 - Pentium有一条5段定点和一条8段浮点流水线。
 - PentiumIII有两条定点和一条浮点指令流水线。
- 多功能流水线（Multifunction Pipelining）：流水线的各段通过不同连接实现不同功能
 - Texas公司的ASC机，8段流水线，能够实现：定点加减法、定点乘法、浮点加法、浮点乘法、逻辑运算、移位操作、数据转换、向量运算等。



(a) 功能段间的互连

(b) 定点乘法

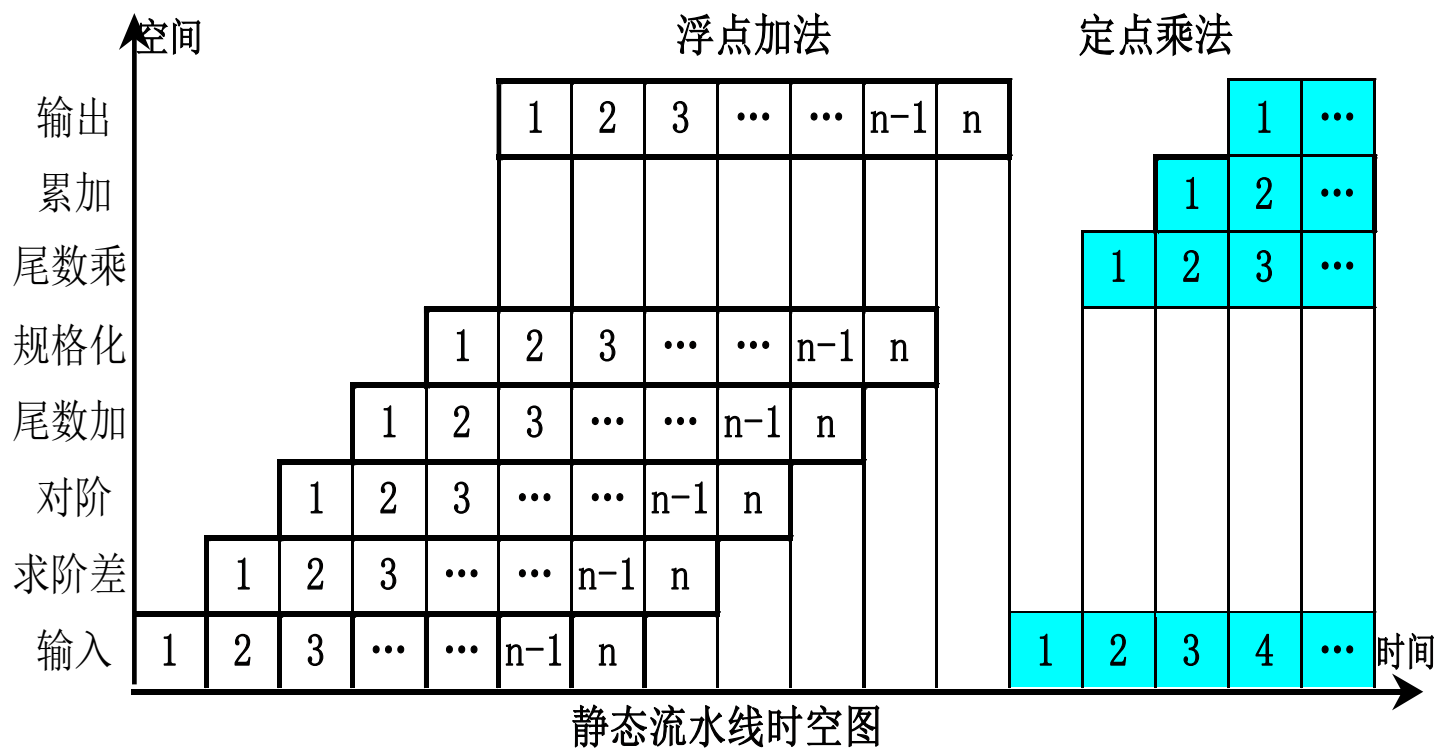
(c) 浮点加法

(d) 浮点点积

5.2.2 流水线的分类

4. 静态流水线与动态流水线

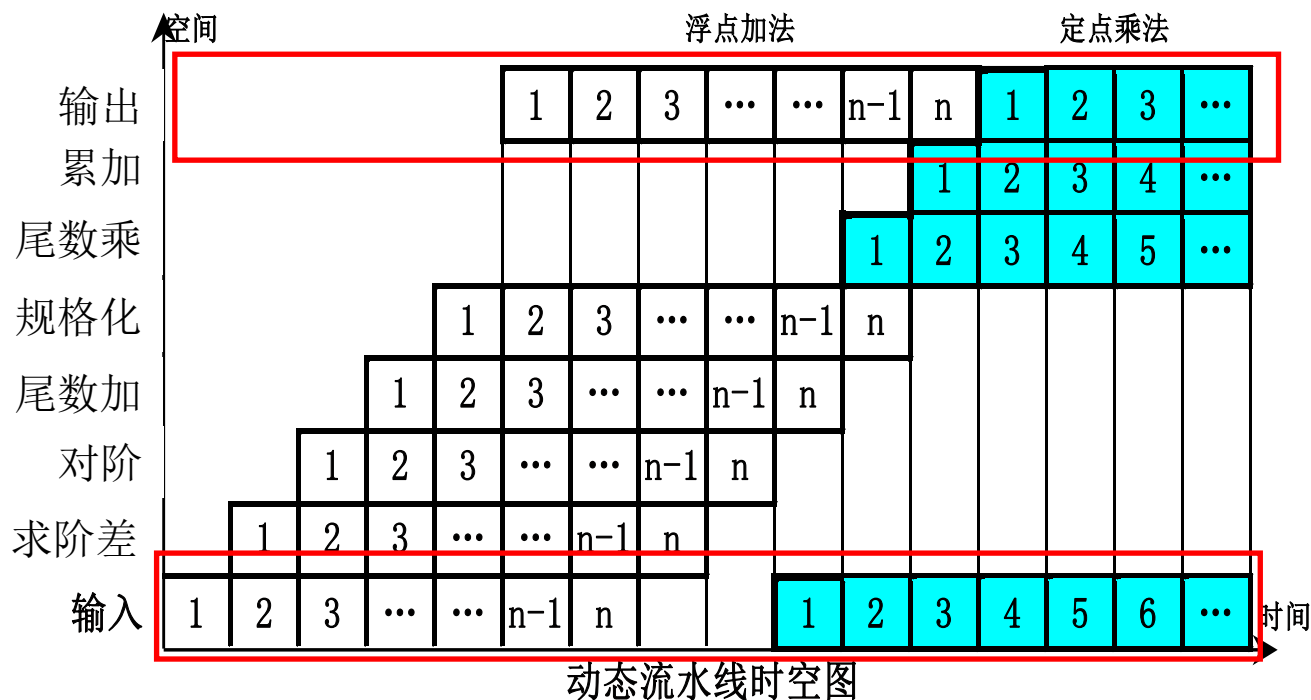
- 静态流水线**：同一段时间内，各个功能段只能按照一种方式连接，实现一种固定的功能。



5.2.2 流水线的分类

4. 静态流水线与动态流水线

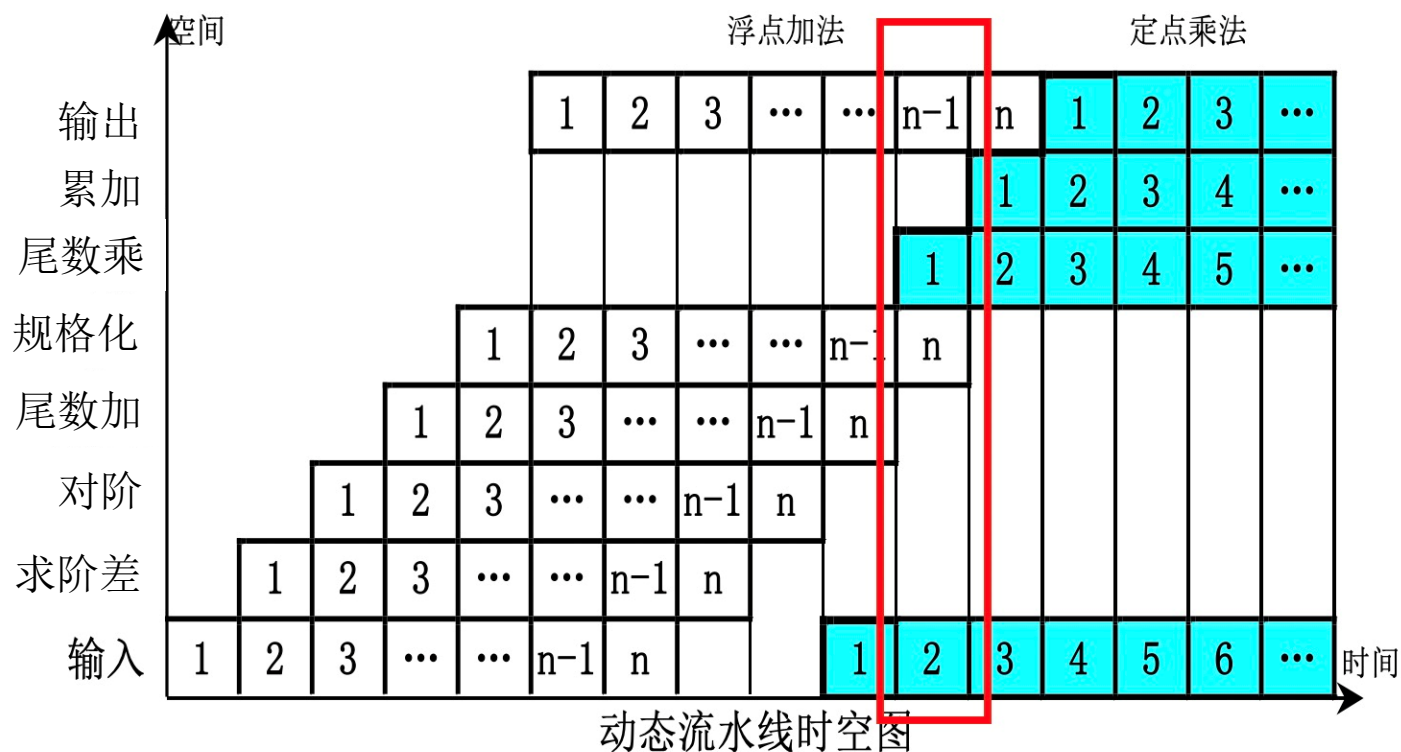
- **动态流水线：**在同一段时间内，各段可以按照不同的方式连接，同时执行多种功能。



5.2.2 流水线的分类

4. 静态流水线与动态流水线

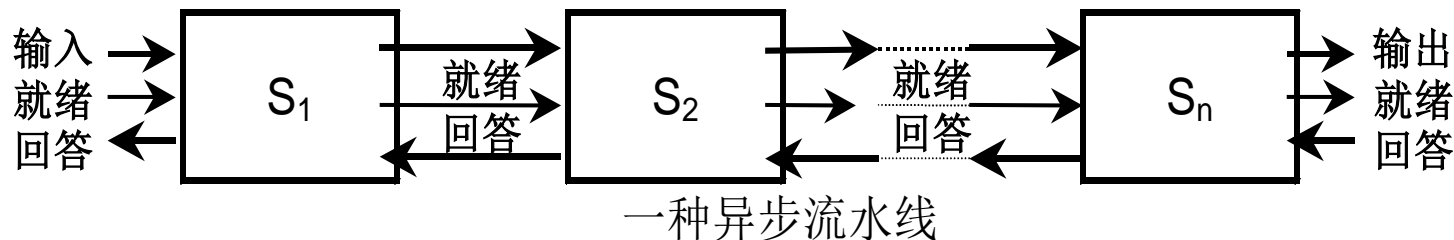
- **动态流水线：**在同一段时间内，各段可以按照不同的方式连接，同时执行多种功能。



5.2.2 流水线的分类

5. 流水线的其他分类方法

- 按照数据表示方式：标量流水线和向量流水线
- 按照控制方式：同步流水线和异步流水线



- 顺序流水线与乱序流水线，乱序流水线又称为无序流水线、错序流水线或异步流水线等。

5.2.3 线性流水线的性能分析

主要指标：吞吐率、加速比和效率

1. 吞吐率 (Throughput)

- 流水线吞吐率的最基本公式：

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

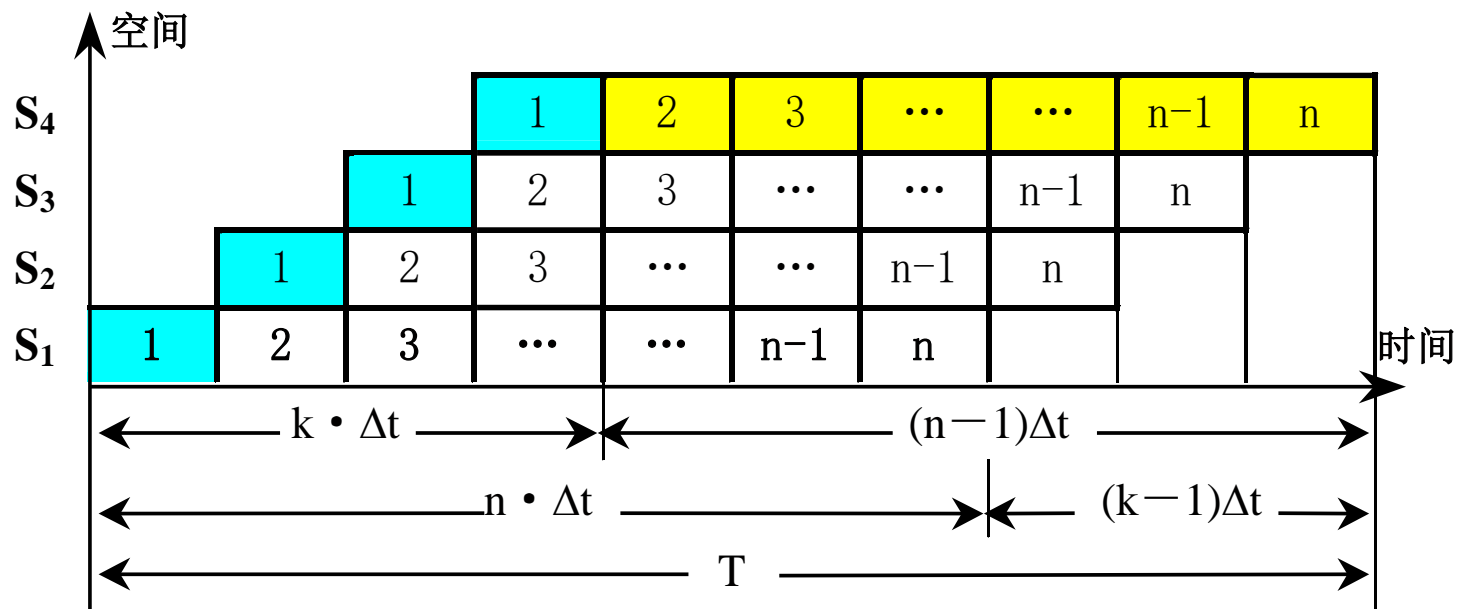
其中： n 为任务数， T_k 为完成 n 个任务所用的时间。

- 各段执行时间相等，输入连续任务情况下，完成 n 个任务需要的总时间为：

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

其中： k 为流水线的段数， Δt 为时钟周期。

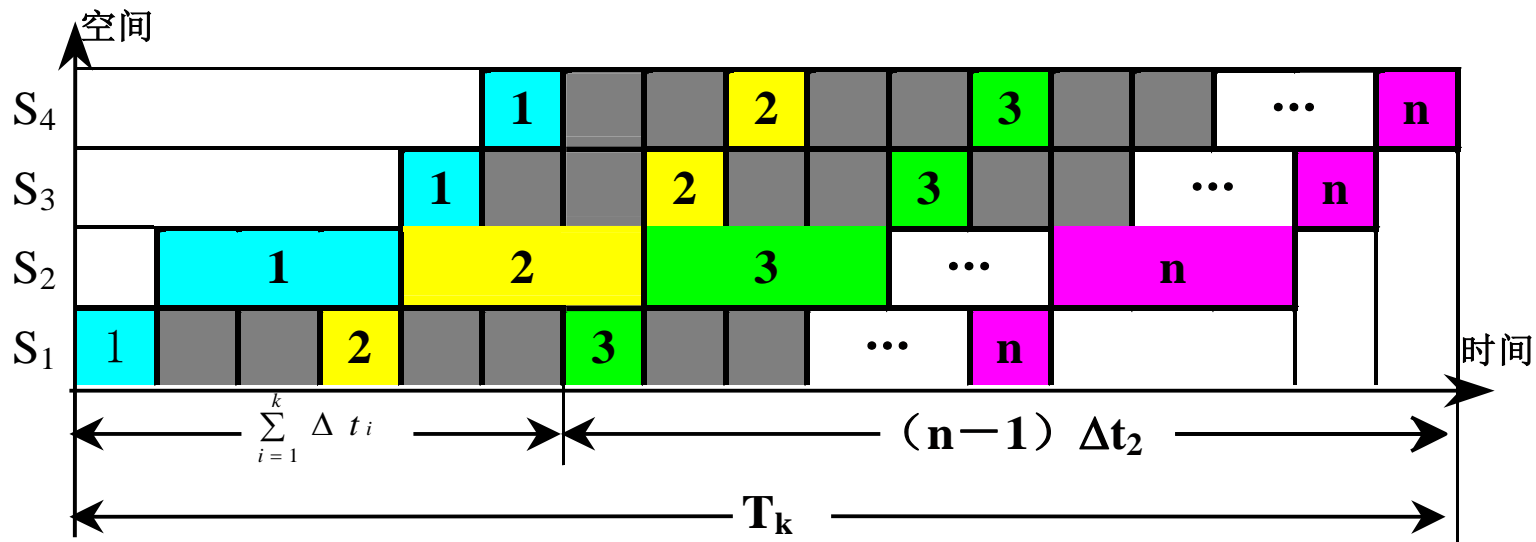
$$T_k = k \cdot \Delta t + (n - 1)\Delta t = (k + n - 1)\Delta t$$



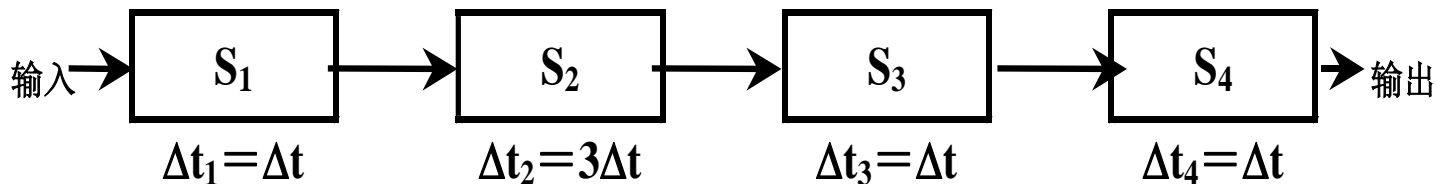
• 吞吐率为: $TP = \frac{n}{(k+n-1)\Delta t}$

• 最大吞吐率为: $TP_{max} = \lim_{n \rightarrow \infty} \frac{n}{(k+n-1)\Delta t} = \frac{1}{\Delta t}$

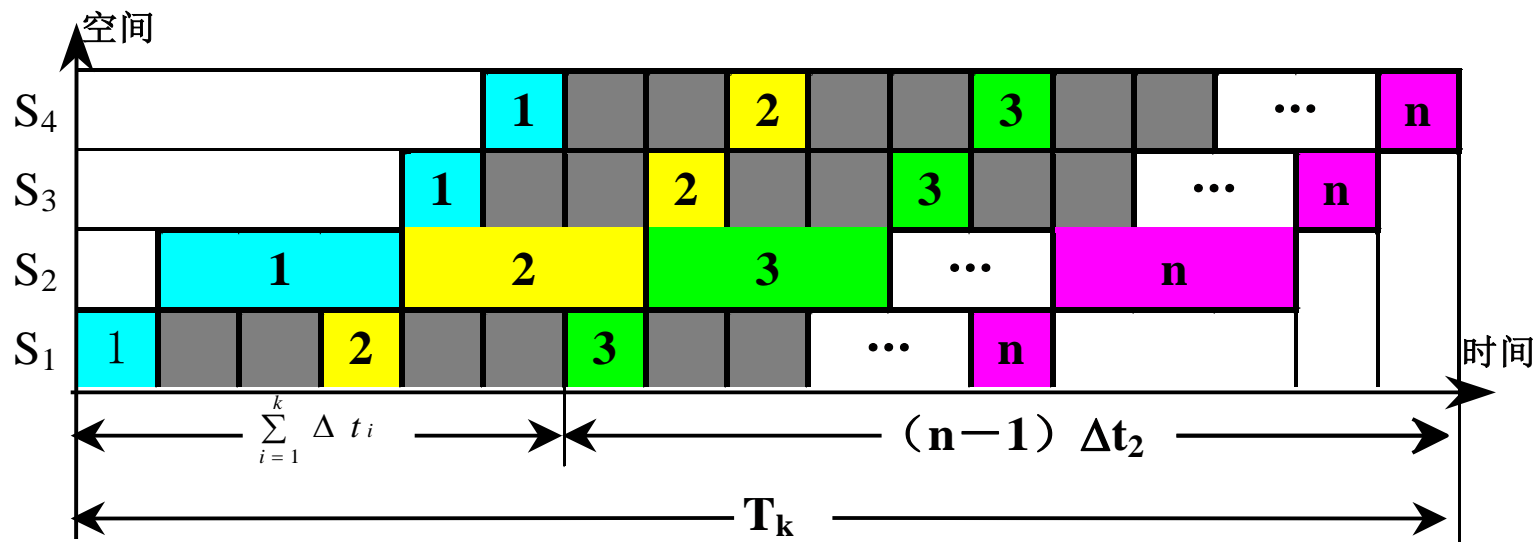
各段时间不等，完成 n 个连续任务：



各段执行时间不相等的流水线及其时空图



各段时间不等，完成 n 个连续任务：

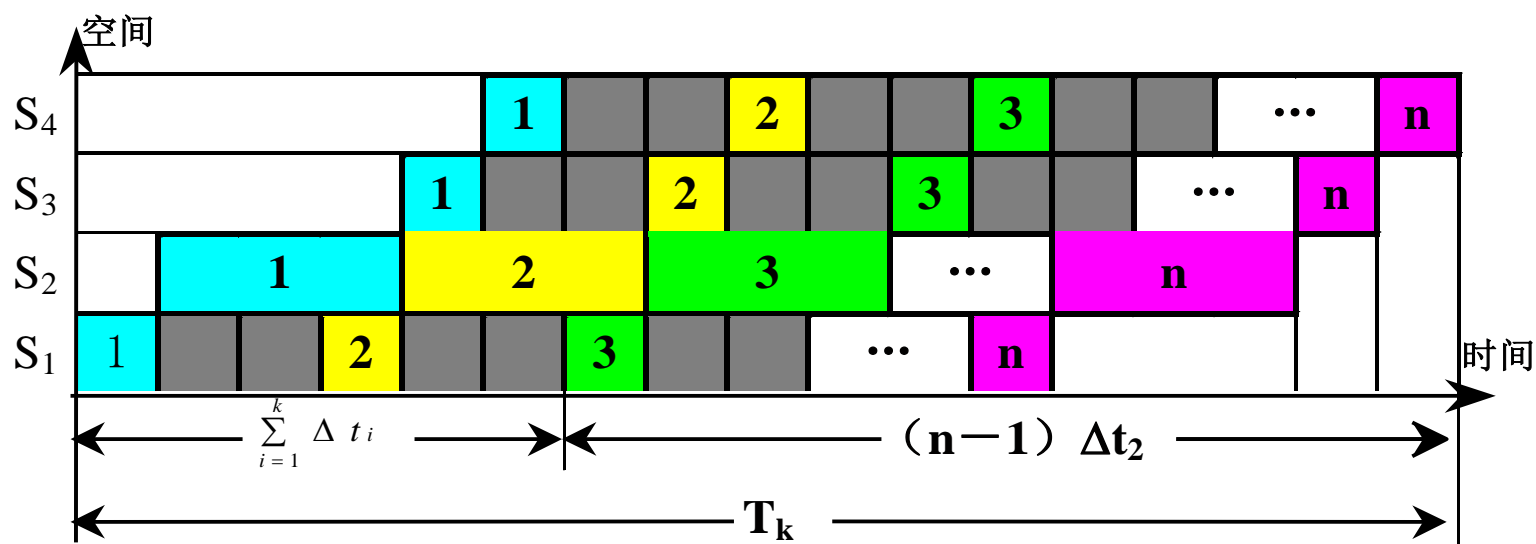
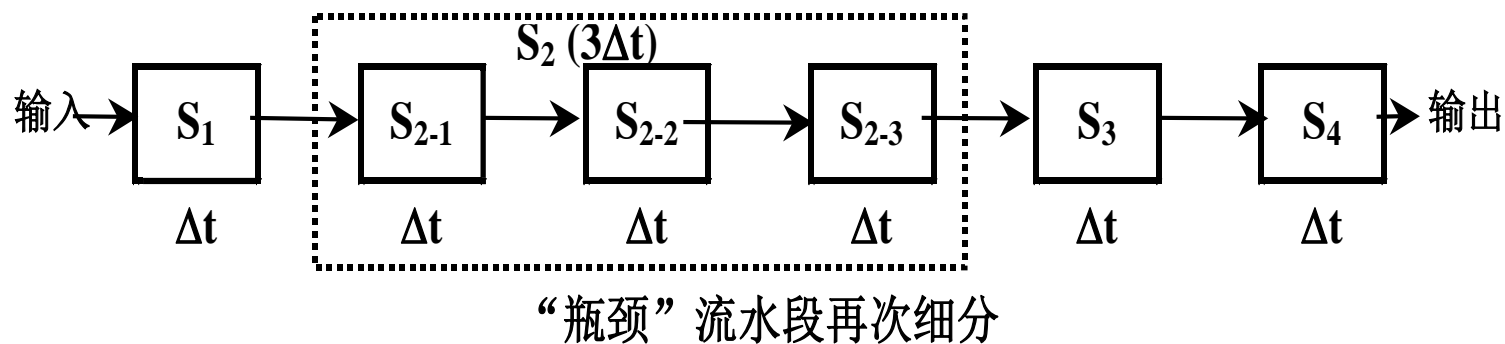


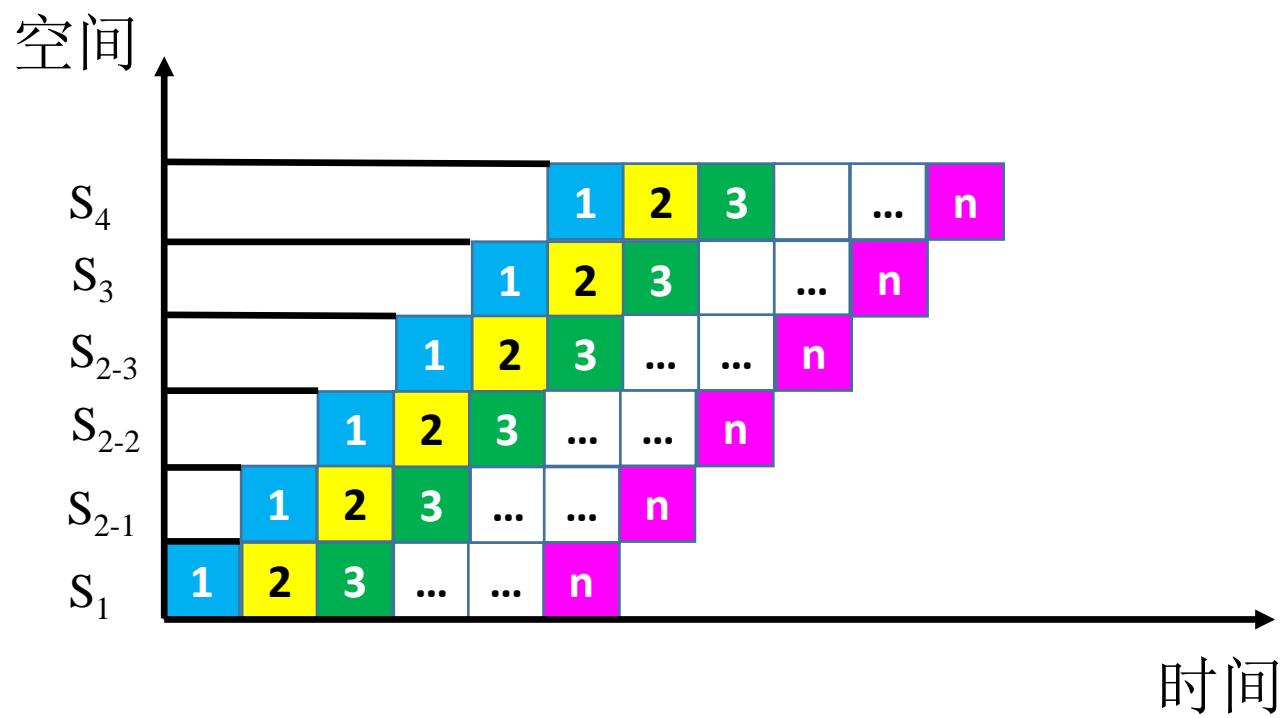
各段执行时间不相等的流水线及其时空图

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

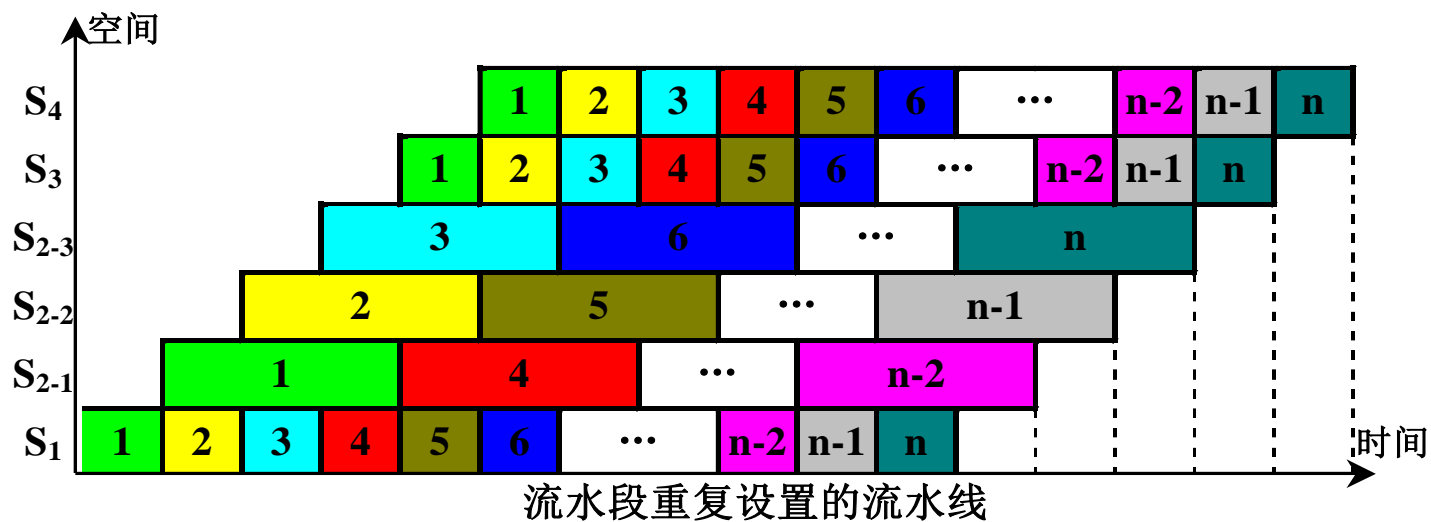
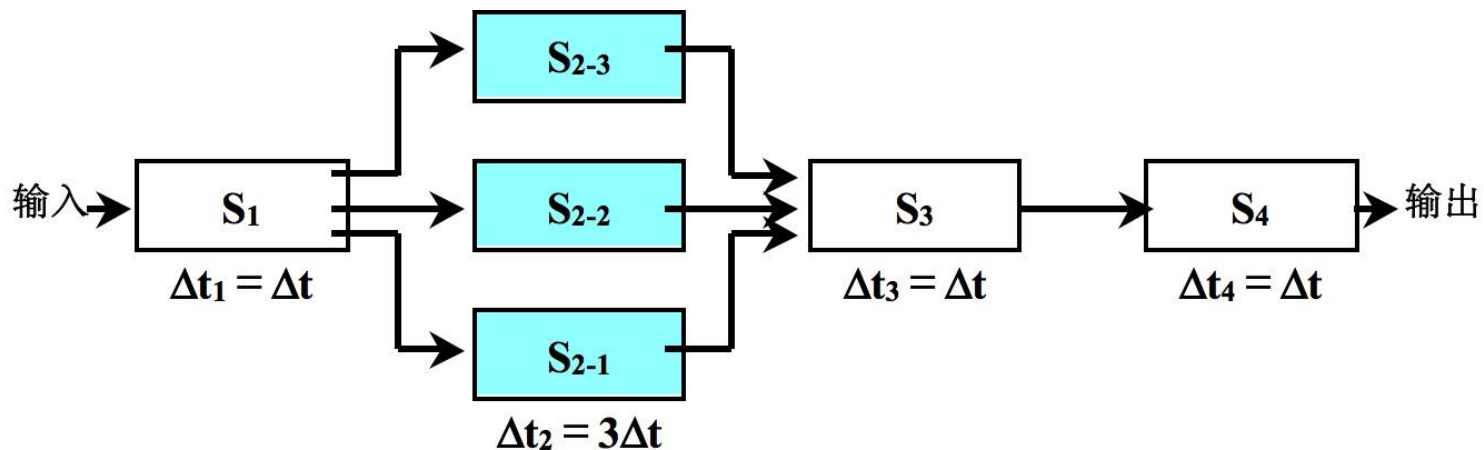
最大吞吐率： $TP = \frac{1}{\max(\Delta t_1, \dots, \Delta t_k)}$

(1) 将“瓶颈”部分再细分（如果可分的话）





(2) “瓶颈”流水段重复设置：增加分配器和收集器



5.2.3 线性流水线的性能分析

2. 加速比 (Speedup)

➤ 计算加速比的基本公式： $S = \frac{\text{顺序执行时间 } T_0}{\text{流水线执行时间 } T_k}$

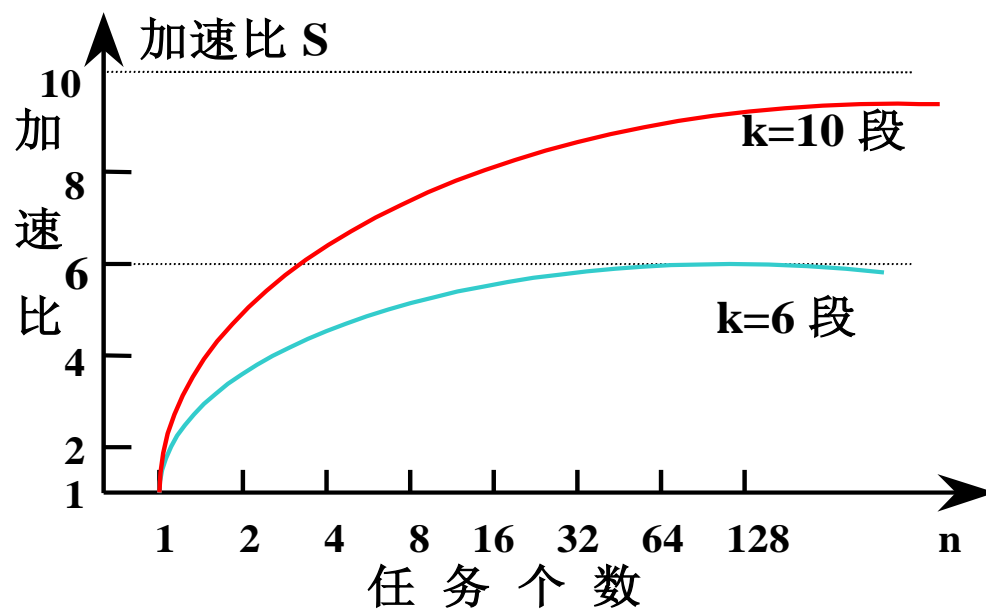
➤ 各段执行时间相等，输入连续任务情况下，

- 加速比： $S = \frac{k \cdot n \cdot \Delta t}{(k+n-1)\Delta t} = \frac{k \cdot n}{k+n-1}$
- 最大加速比： $S_{max} = \lim_{n \rightarrow \infty} \frac{k \cdot n}{k+n-1} = k$

➤ 各段时间不等，输入连续任务情况下，实际加速比为：

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

- 当流水线段数增加时，需要连续输入的任务数也必须增加

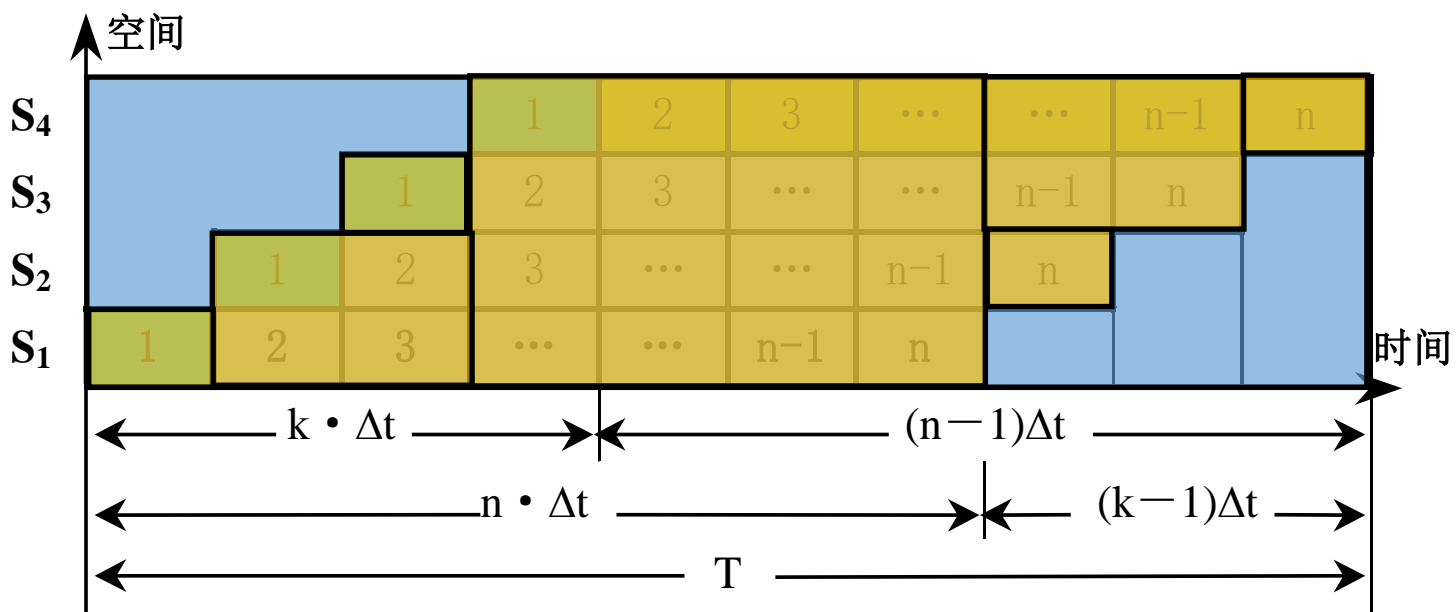
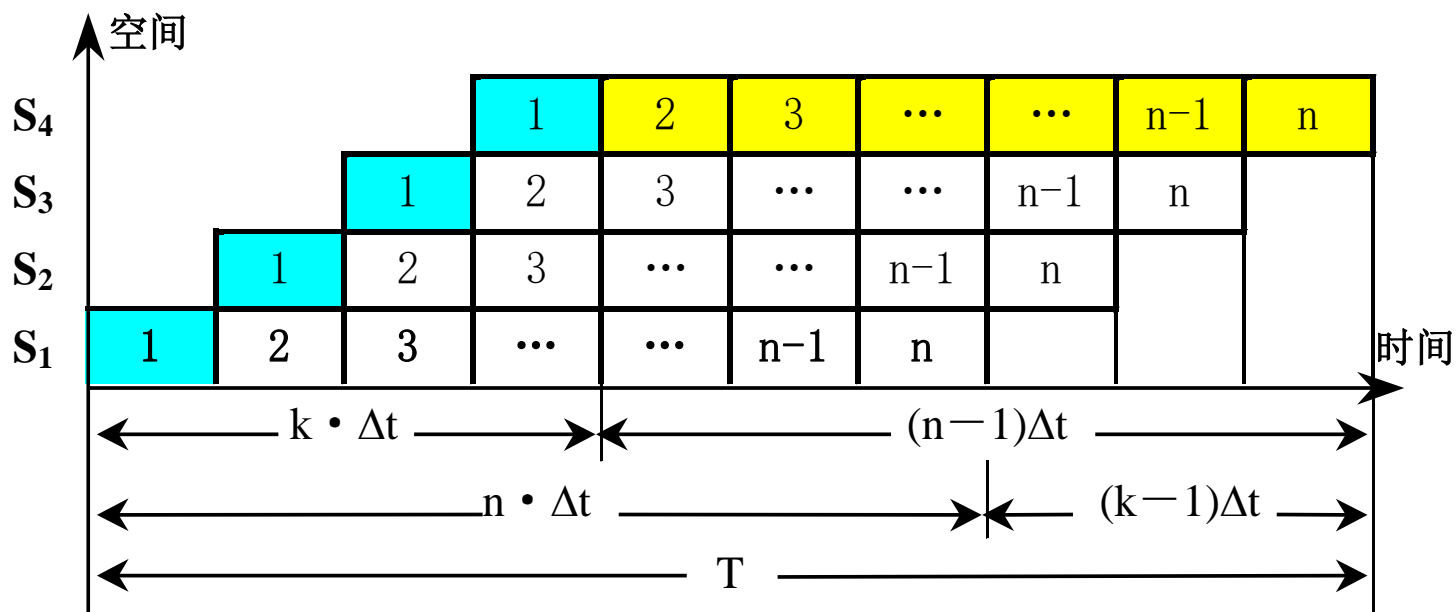


5.2.3 线性流水线的性能分析

3. 效率 (Efficiency)

- 计算流水线效率的一般公式: $E = \frac{n \text{个任务占用的时空区}}{k \text{个流水段的总的时空区}} = \frac{T_0}{k \cdot T_k}$
- 各流水段时间相等, 输入 n 个连续任务, 流水线的效率为:

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



5.2.3 线性流水线的性能分析

3. 效率 (Efficiency)

- 计算流水线效率的一般公式: $E = \frac{n \text{个任务占用的时空区}}{k \text{个流水段的总的时空区}} = \frac{T_0}{k \cdot T_k}$

- 各流水段时间相等, 输入 n 个连续任务, 流水线的效率为:

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

- 最高效率为: $E_{\max} = \lim_{n \rightarrow \infty} \frac{n}{k + n - 1} = 1$

- 各流水段时间不等, 输入 n 个连续任务, 流水线效率为:

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

5.2.3 线性流水线的性能分析

3. 效率 (Efficiency)

- 各段设备量或价格不等时，流水线的效率为：

$$E = \frac{n \text{个任务占用的加权时空区}}{k \text{个流水段的总的加权时空区}}$$

即：

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

其中， $a_i < k$ ，且 $\sum_{i=1}^k a_i = k$ 。

5.2.3 线性流水线的性能分析

3. 效率 (Efficiency)

- 流水线的吞吐率、加速比与效率的关系：

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

因此： $E = TP \cdot \Delta t$, $S = k \cdot E$

5.2.3 线性流水线的性能分析

4. 流水线最佳段数的选择

- 采用顺序执行方式完成一个任务的时间为 t
- 在同等速度的 k 段流水线上执行一个任务的时间为: $\frac{t}{k} + d$
 - d 为流水锁存器的延迟时间
- 流水线的最大吞吐率为: $P = 1/(t/k + d)$
- 流水线的总价格估计为: $C = a + b k$,
 - a 为功能段身的总价格, b 为每个锁存器的价格
- A. G. Larson把流水线的性能价格比PCR定义为:

$$PCR = \frac{P}{C} = \frac{1}{t/k + d} \cdot \frac{1}{a + bk}$$

求PCR的最大值为:

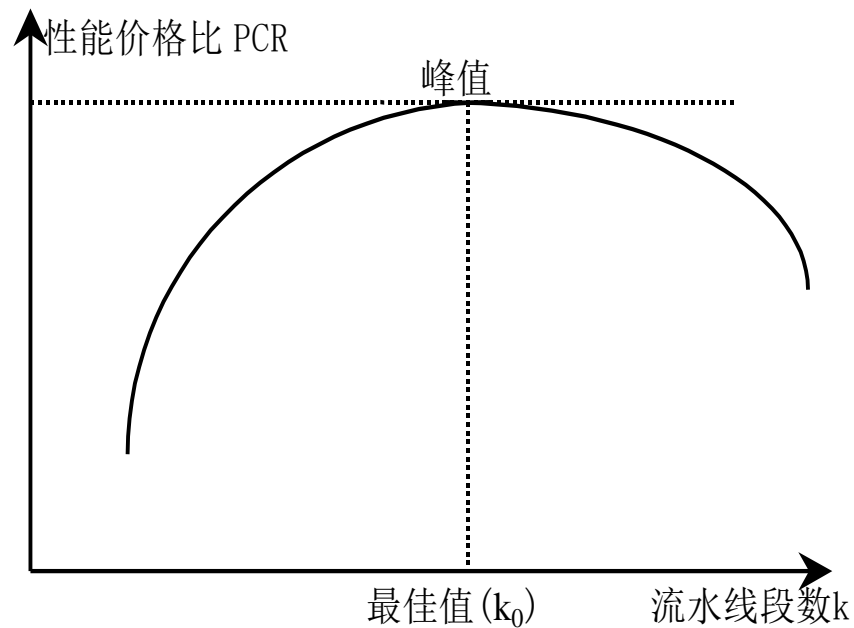
$$\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$$

$$\left(\frac{1}{\frac{t}{k}+d} \cdot \frac{1}{a+bk}\right)' = \left(\frac{k}{t+dk} \cdot \frac{1}{a+bk}\right)' = 0 \quad (uv)' = u'v + uv'$$

$$(t + dk)(a + bk) - dk(a + bk) - bk(t + dk) = 0$$

$$at - bdk^2 = 0$$

$$k = \sqrt{\frac{t \cdot a}{d \cdot b}}$$



5.2.3 线性流水线的性能分析

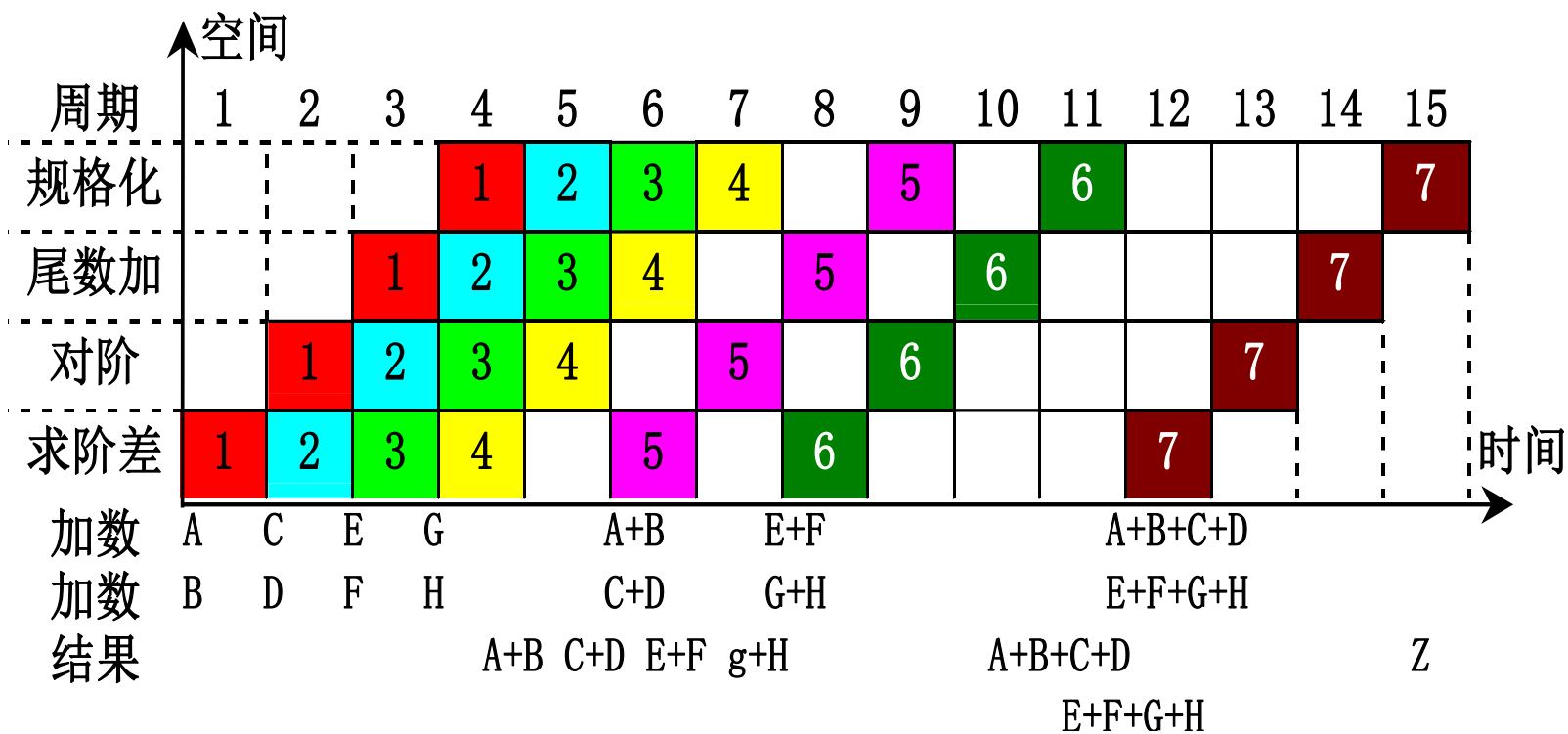
5. 流水线性能分析举例

- 对于单功能线性流水线，输入连续任务的情况，通过上面给出的公式很容易计算出流水线的吞吐率、加速比和效率。
- 对于输入不连续任务，或多功能流水线，通常采用基本公式计算。

用一条4段浮点加法器流水线求8个浮点数的和：

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

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



用一条 4 段浮点加法器流水线求 8 个数之和的流水线时空图

7 个浮点加法共用了 15 个时钟周期。

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

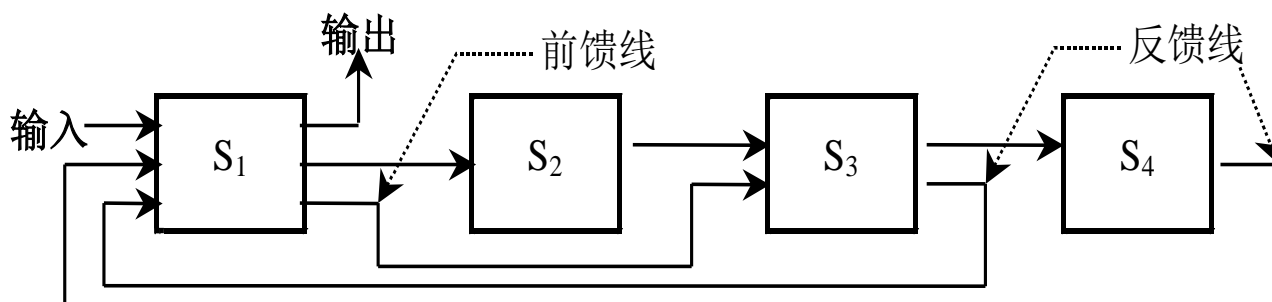
流水线的效率为: $E = \frac{T_0}{k \cdot T_k} = \frac{4 \times 7 \cdot \Delta t}{4 \times 15 \cdot \Delta t} = 0.47$

5.2.4 非线性流水线的调度

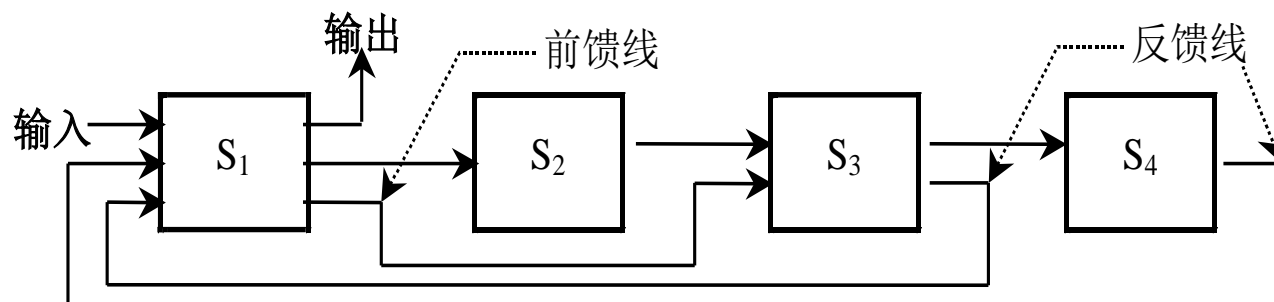
非线性流水线调度的任务是要找出一个最小的循环周期，按照这周期向流水线输入新任务，流水线的各个功能段都不会发生冲突，而且流水线的吞吐率和效率最高。

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
- 对于非线性流水线，连接图不能唯一表示工作流程，因此，引入流水线预约表



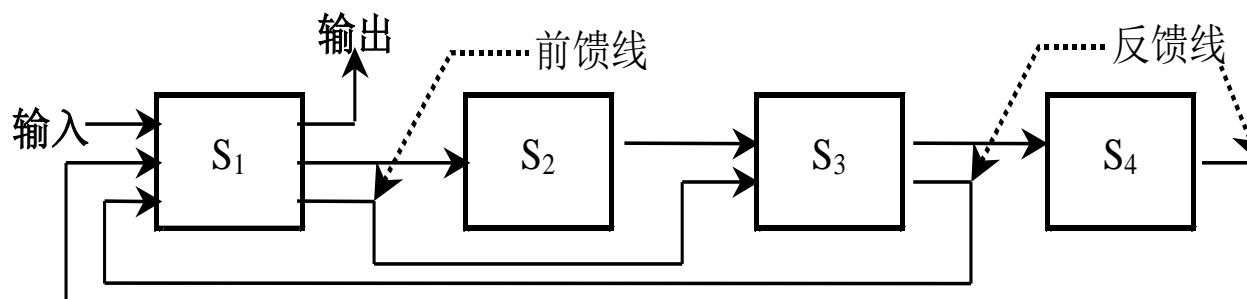
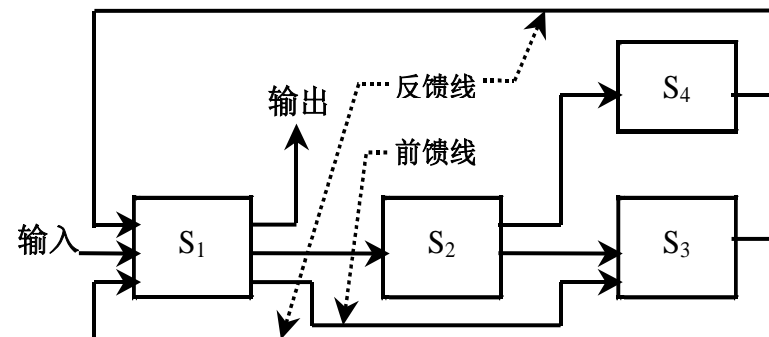
非线性流水线的连接图和预约表



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

一张预约表可能与多个流水线连接图相对应

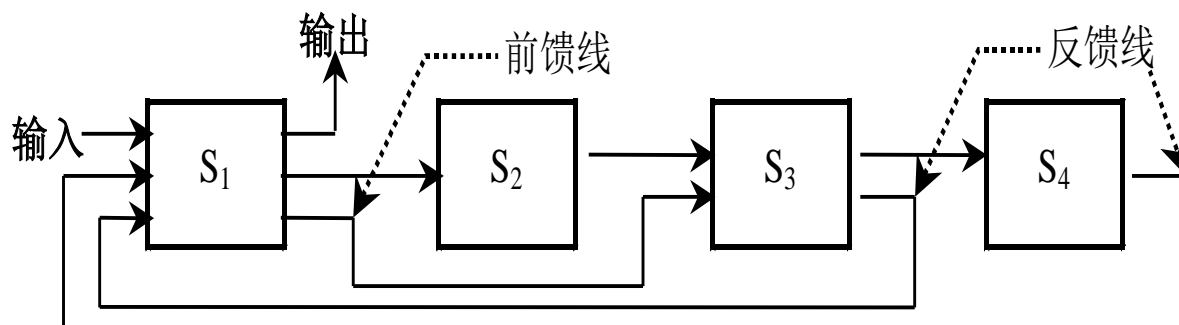
时间 功能段	1	2	3	4	5	6	7
S_1	×			×			×
S_2		×			×		
S_3		×				×	
S_4			×				



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	×			×			×
S_2		×			×		
S_3		×				×	
S_4			×				

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



2. 非线性流水线的冲突

- 启动距离：连续输入两个任务之间的时间间隔
- 流水线冲突：几个任务争用同一个流水段

启动距离为 3 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 ₁ X ₂			X ₁ X ₂ X ₃			X ₂ X ₃ X ₄		...
S ₂		X ₁			X ₁ X ₂			X ₂ X ₃			X ₃ X 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 ₁ X ₄	X ₃	X ₂ X ₅	X ₄	X ₃ X 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 ₁ X ₃		X ₂ X ₄		X ₃ X ₅		...
S ₄			X ₁		X ₂		X ₃		X ₄		X ₅	...

启动距离为 5 时的流水线不冲突

时间 功能段	1	2	3	4	5	6	7	8	9	10	11	...
S ₁	X ₁			X ₁		X ₂	X ₁		X ₂		X ₃	...
S ₂		X ₁			X ₁		X ₂			X ₂		...
S ₃		X ₁				X ₁	X ₂				X ₂	...
S ₄			X ₁					X ₂				...

启动距离为 (1, 7) 循环时的流水线预约表

时间 功能段	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
S ₁	X ₁	X ₂		X ₁	X ₂		X ₁	X ₂	X ₃	X ₄		X ₃	X ₄		X ₃	X ₄	...
S ₂		X ₁	X ₂		X ₁	X ₂				X ₃	X ₄		X ₃	X ₄			...
S ₃		X ₁	X ₂			X ₁	X ₂			X ₃	X ₄			X ₃	X ₄		...
S ₄			X ₁	X ₂							X ₃	X ₄					...

3. 无冲突调度方法

- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。上例中为
(3, 4, 6)

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


- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。

上例中为 (3, 4, 6)

功能段 \ 时间	1	2	3	4	5	6	7
S ₁	×			×			×
S ₂		×			×		
S ₃		×				×	
S ₄			×				

- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。
上例中为 (3, 4, 6)

功能段 \ 时间	1	2	3	4	5	6	7
S ₁	×			×			×
S ₂		×		4	×		
S ₃		×				×	
S ₄			×				



- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。

上例中为 (3, 4, 6)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S1	X1			X1 X2			X1 X2			X2						
S2		X1			X1 X2			X2								
S3		X1			X2	X1			X2							
S4			X1			X2										

- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。

上例中为 (3, 4, 6)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S1	X1			X1	X2		X1	X2			X2					
S2		X1			X1	X2			X2							
S3		X1				X1 X2				X2						
S4			X1				X2									

- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。

上例中为 (3, 4, 6)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S1	X1			X1			X1 X2			X2			X2			
S2		X1			X1			X2			X2					
S3		X1				X1		X2				X2				
S4			X1						X2							

- **禁止向量**：预约表中每一行任意两个“×”之间距离的集合。

上例中为 (3, 4, 6)

- **冲突向量**： $C = (C_m C_{m-1} \dots C_2 C_1)$

其中：m是禁止向量中的最大值。

如果i在禁止向量中，则 $C_i = 1$ ，否则 $C_i = 0$ 。

上例中 $C = \overset{6\ 5\ 4\ 3\ 2\ 1}{(101100)}$

时间 功能段	1	2	3	4	5	6	7
S ₁	×			×			×
S ₂		×			×		
S ₃		×				×	
S ₄			×				

- **禁止向量:** (3, 4, 6)
- **初始冲突向量:** C=(101100)
- **状态图:** 将冲突向量逻辑右移, 若移出去的位是1, 则表示用相应启动距离向流水线输入新任务时会产生功能段冲突; 若移出去的位是0, 则表示不会产生功能段冲突

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S1	X1			X1			X1 X2			X2			X2			
S2		X1			X1			X2			X2					
S3		X1				X1		X2				X2				
S4			X1						X2							

逻辑右移6位时, 移动出去的位是1

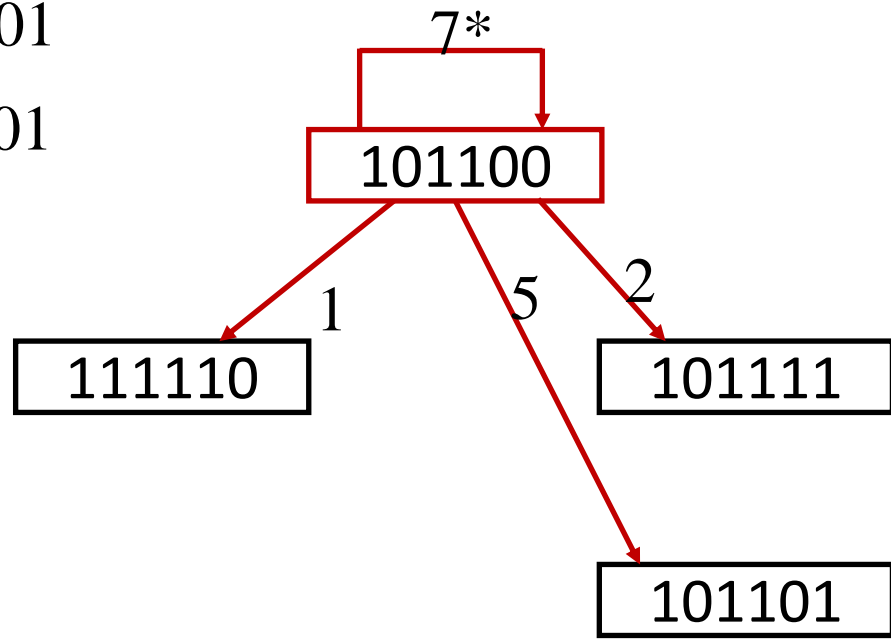
- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$
- 状态图:

$010110 \vee 101100 = 111110$

$001011 \vee 101100 = 101111$

$000001 \vee 101100 = 101101$

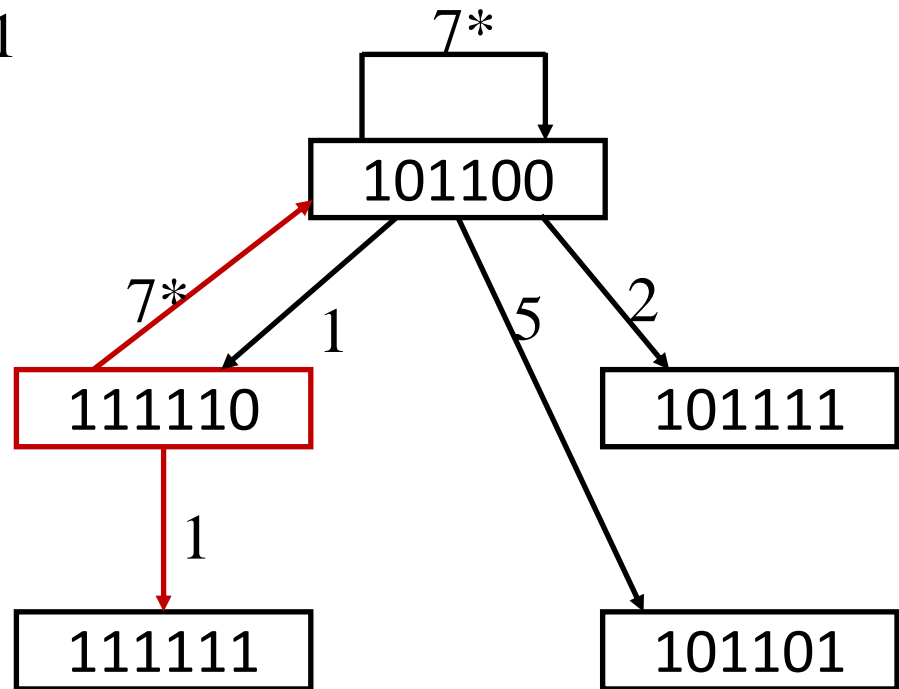
$000000 \vee 101100 = 101101$



- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$
- 状态图:

$011111 \vee 101100 = 111111$

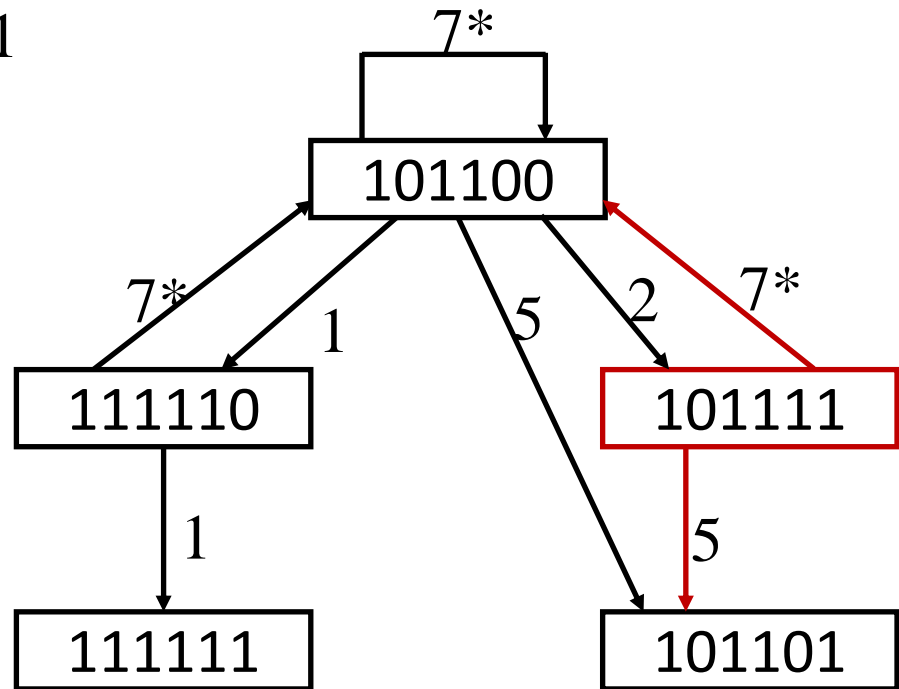
$000000 \vee 101100 = 101101$



- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$
- 状态图:

$000001 \vee 101100 = 101101$

$000000 \vee 101100 = 101101$



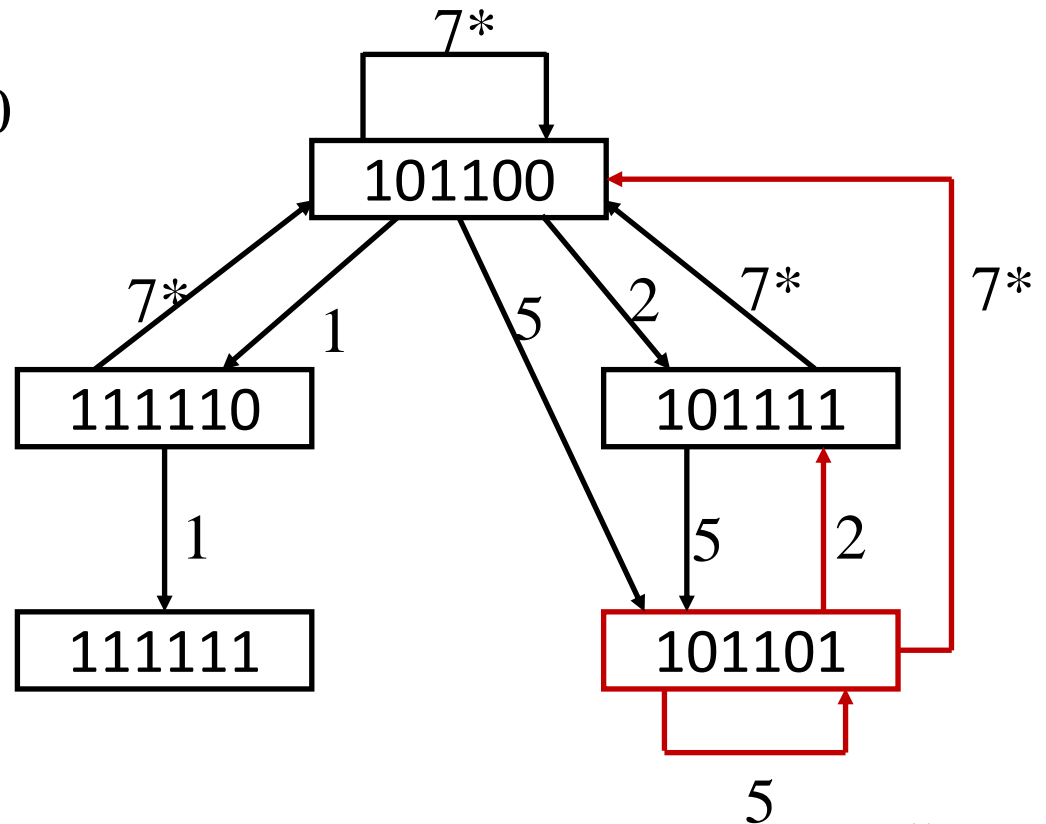
- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$

• 状态图:

$001011 \vee 101100 = 101111$

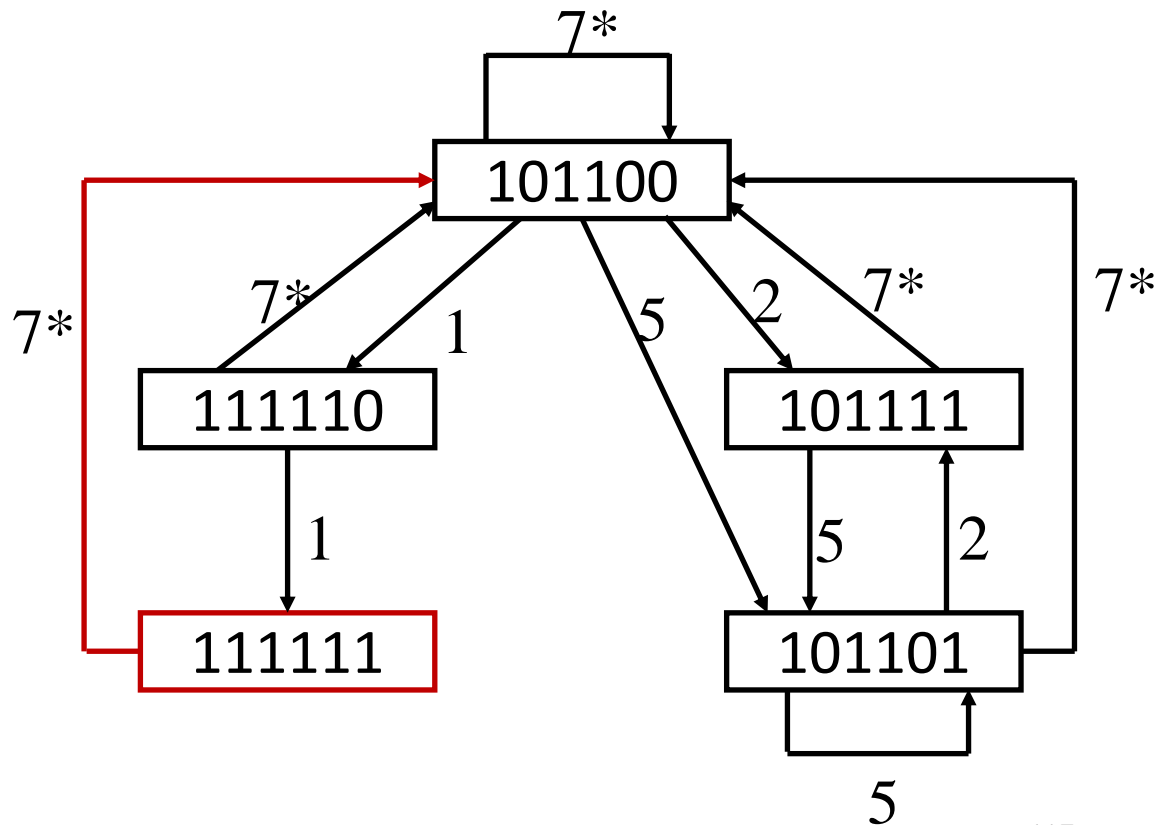
$000001 \vee 101100 = 101101$

$000000 \vee 101100 = 101100$

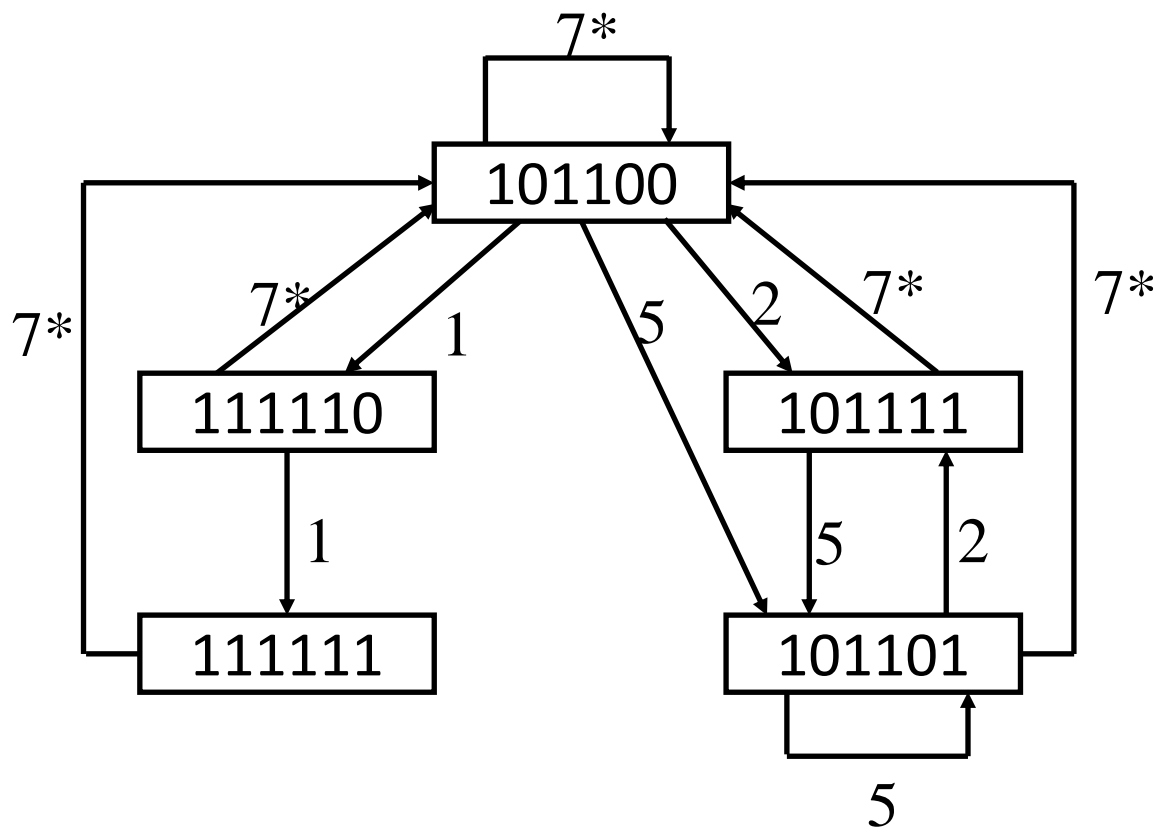


- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$
- 状态图:

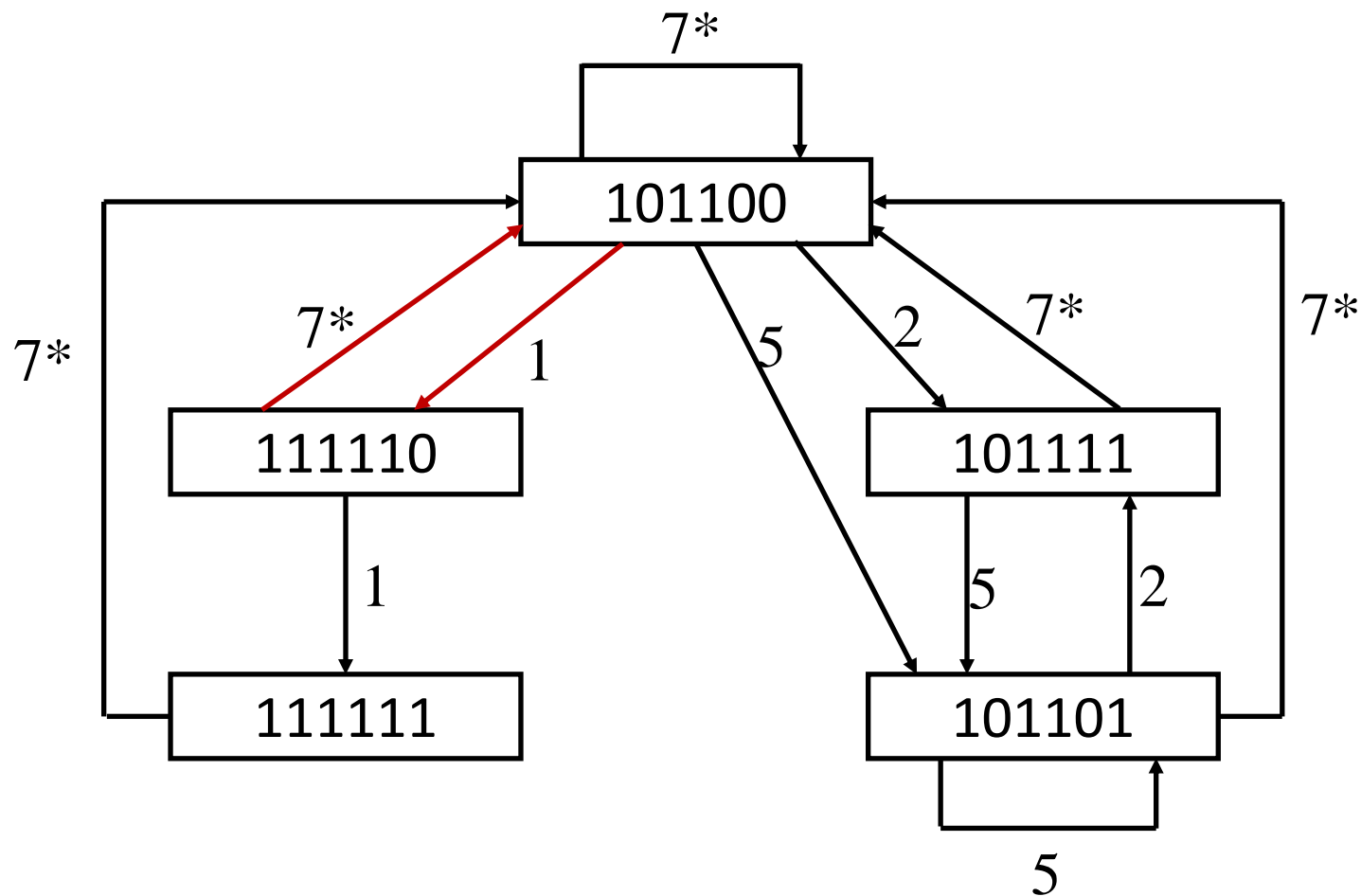
$$000000 \vee 101100 = 101100$$



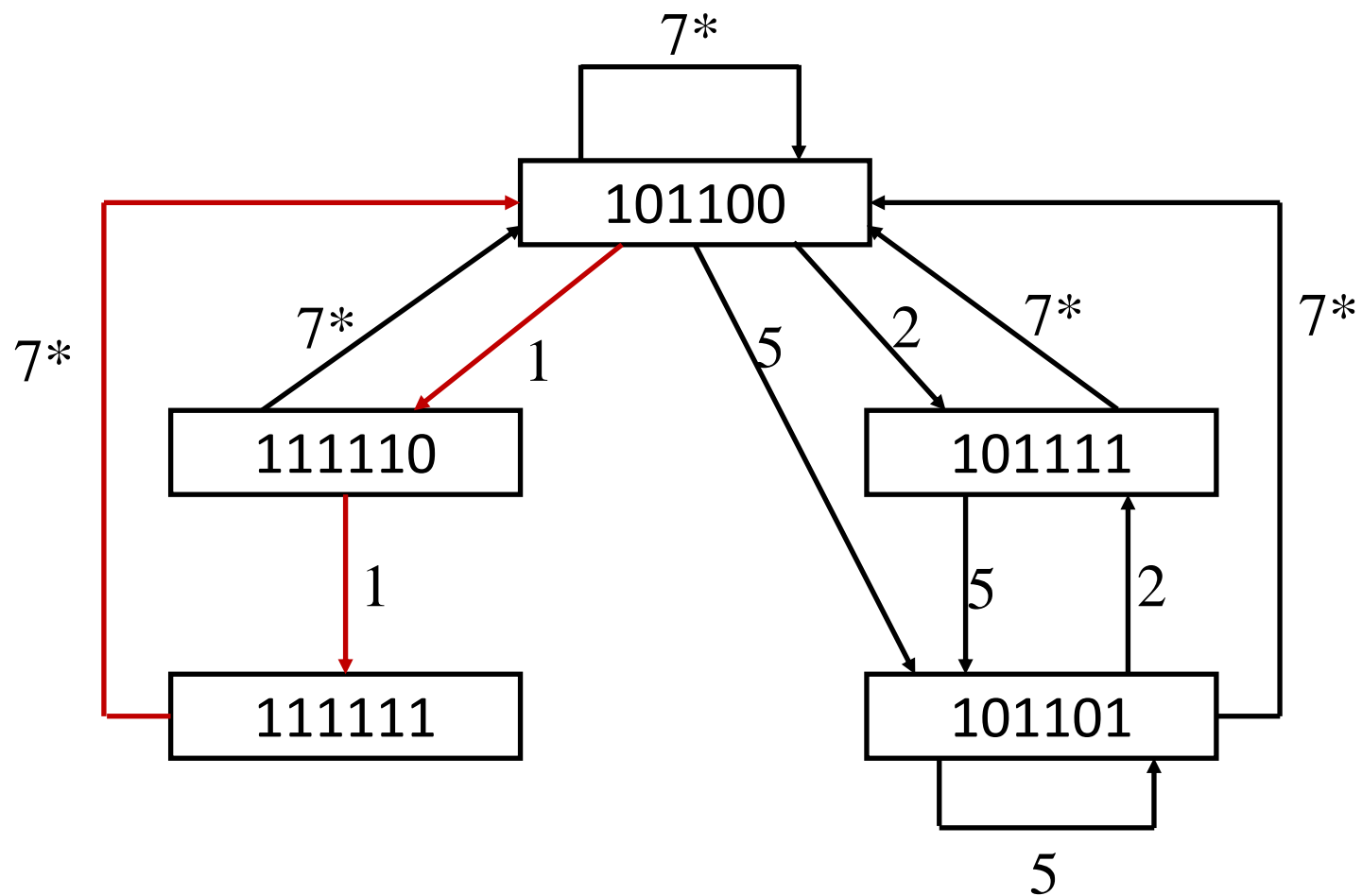
- 禁止向量: (3, 4, 6)
- 初始冲突向量: $C = (101100)$
- 状态图:



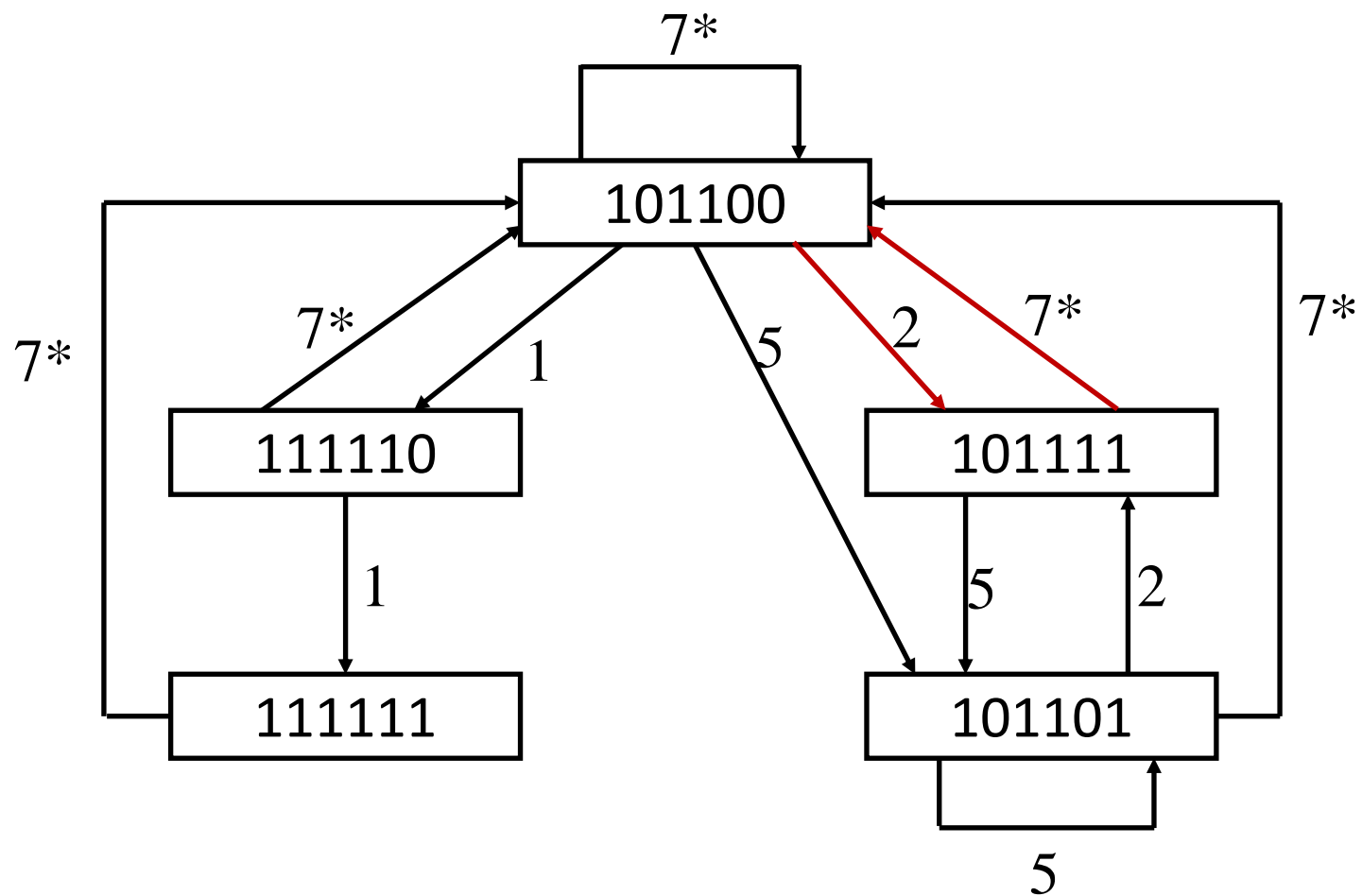
简单循环：在状态图中各种冲突向量只经过一次的启动循环



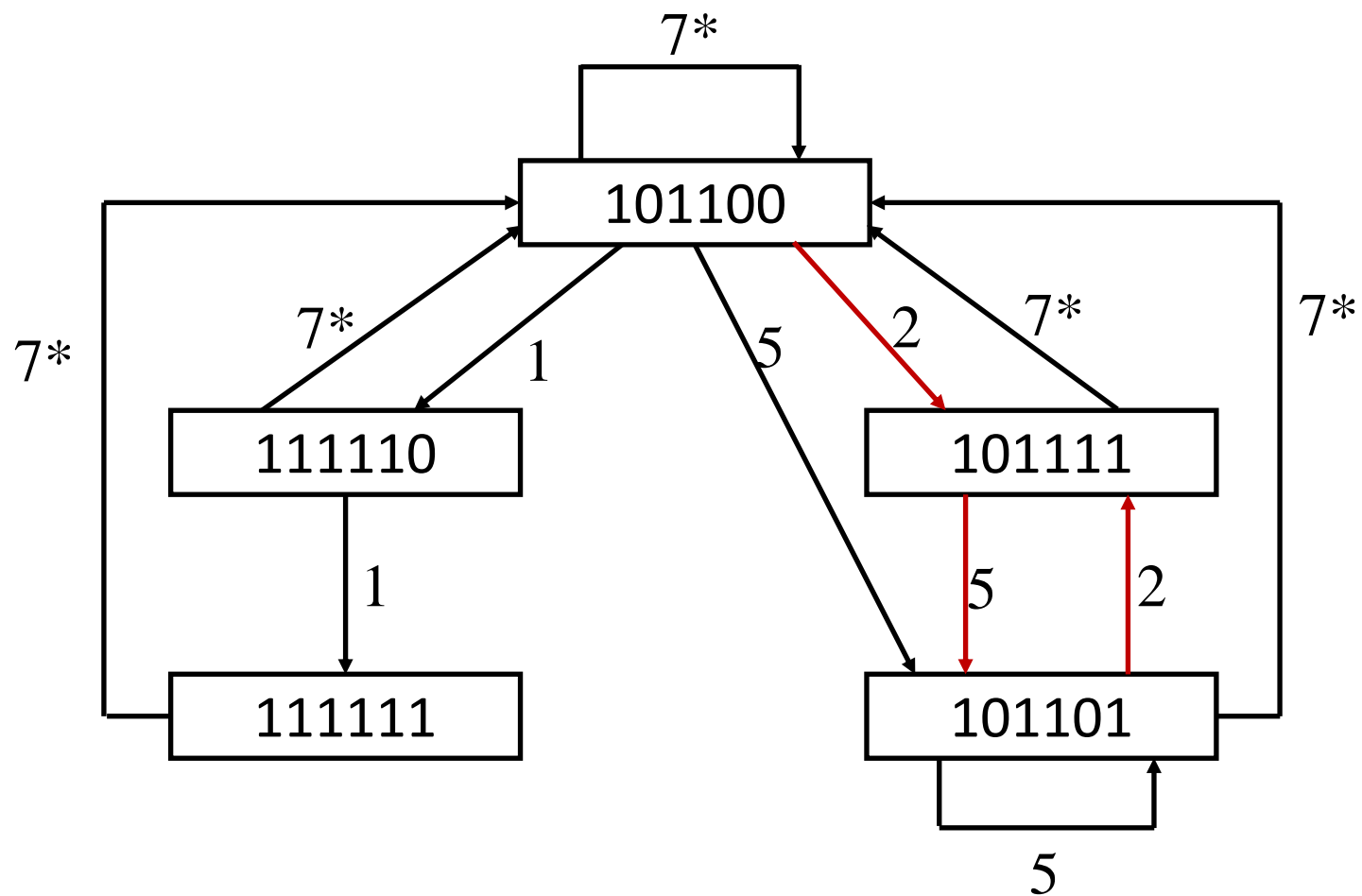
(1,7)



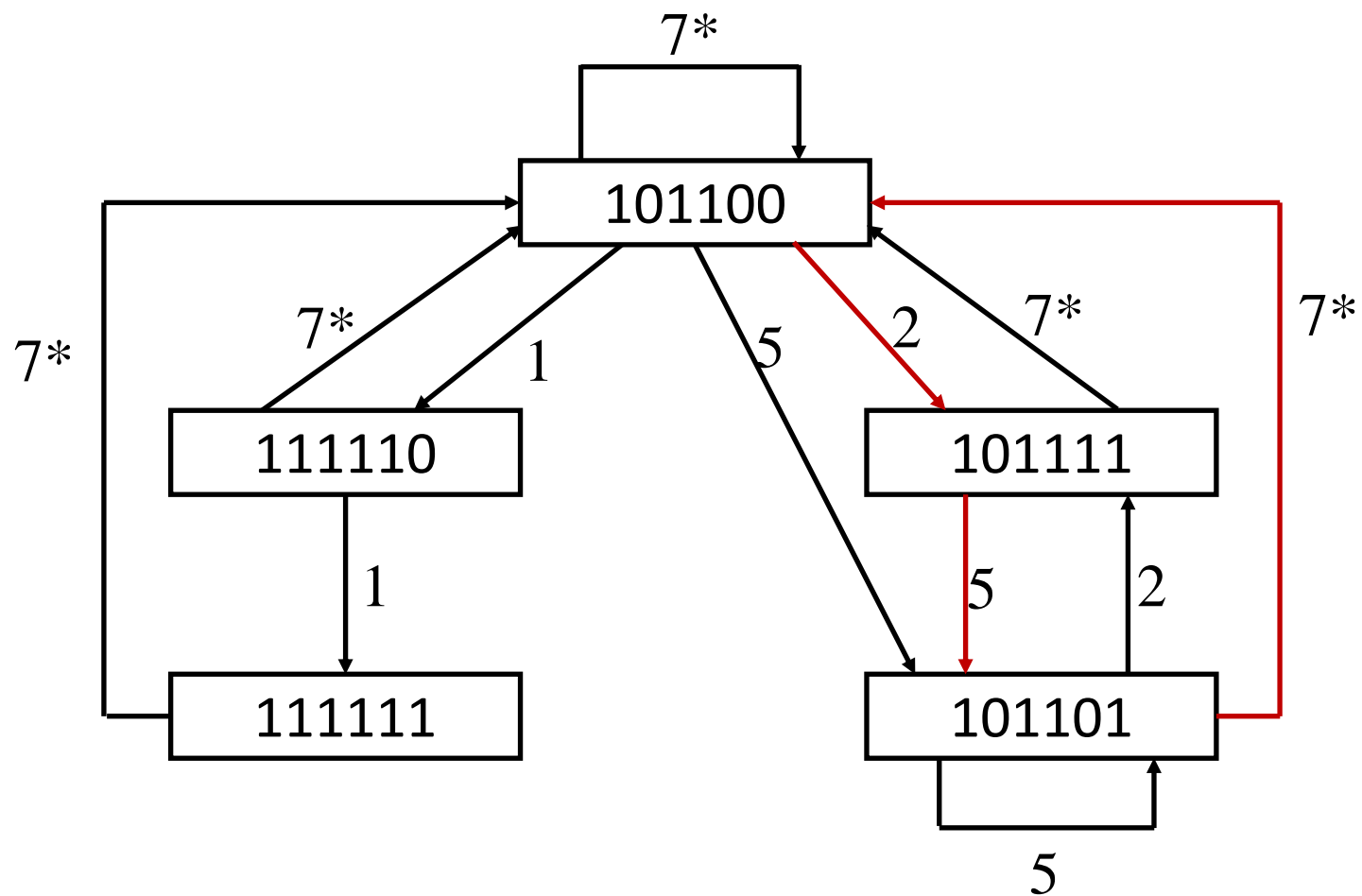
(1,1,7)



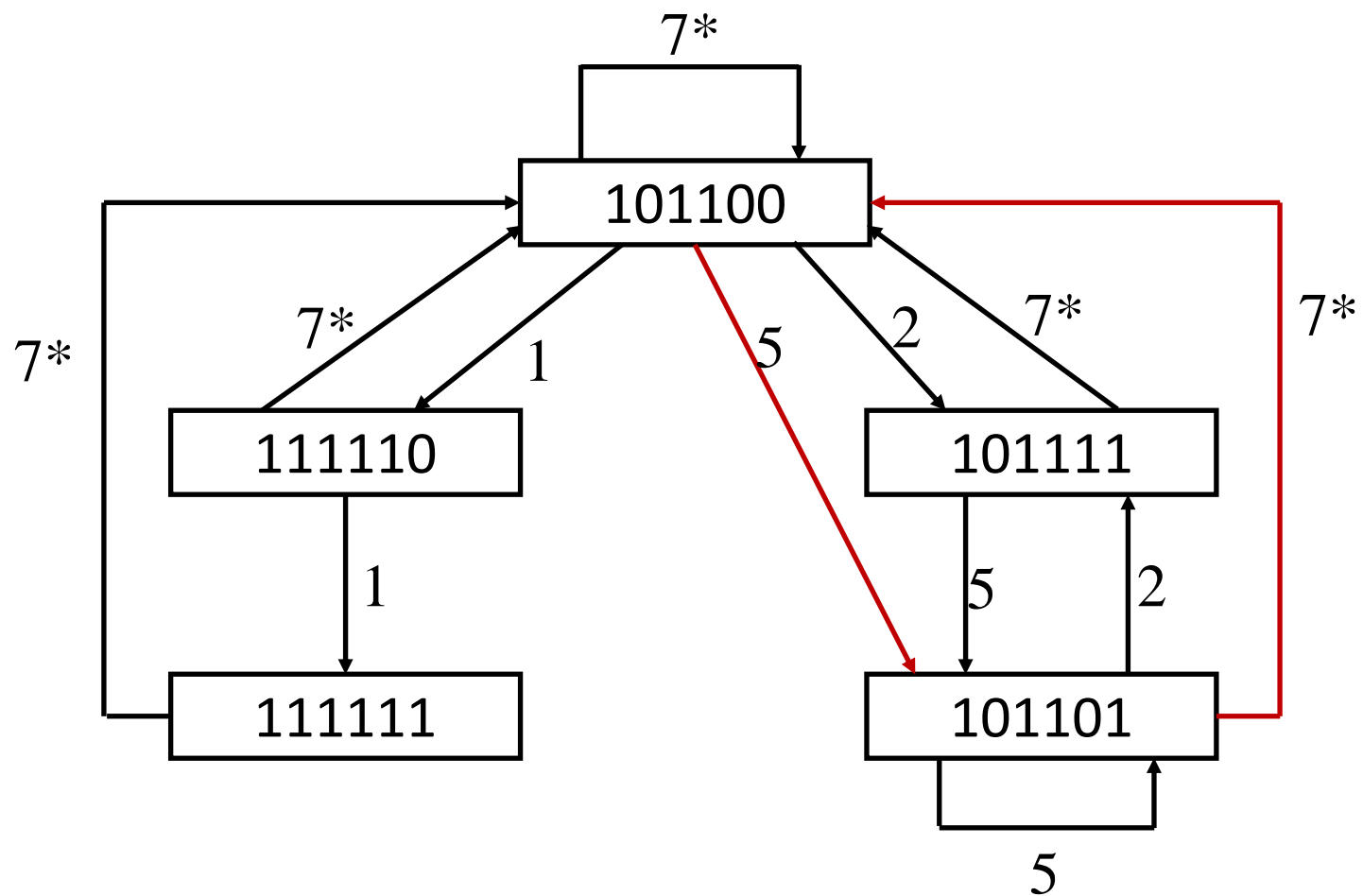
(2,7)



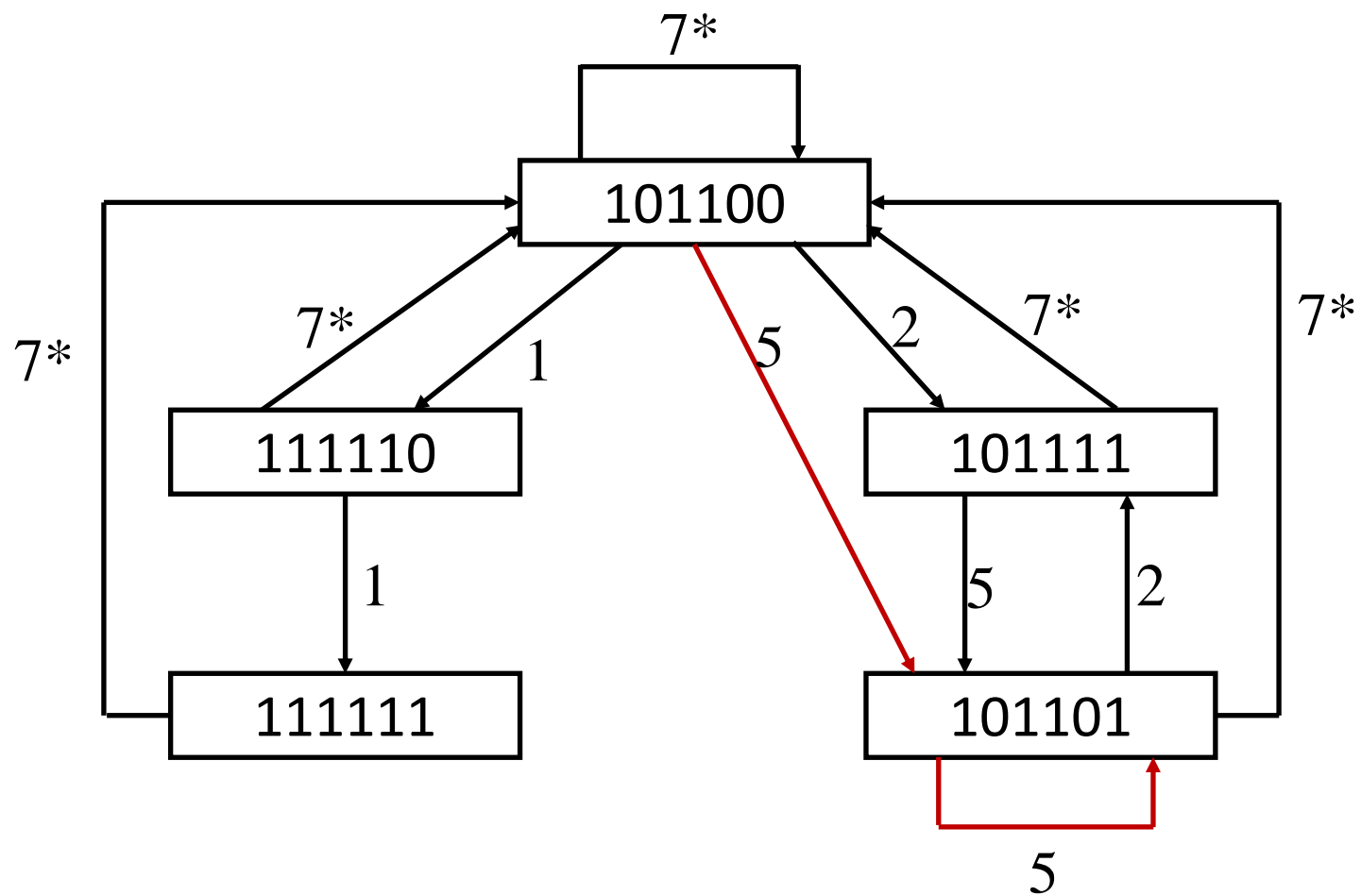
(2,5)



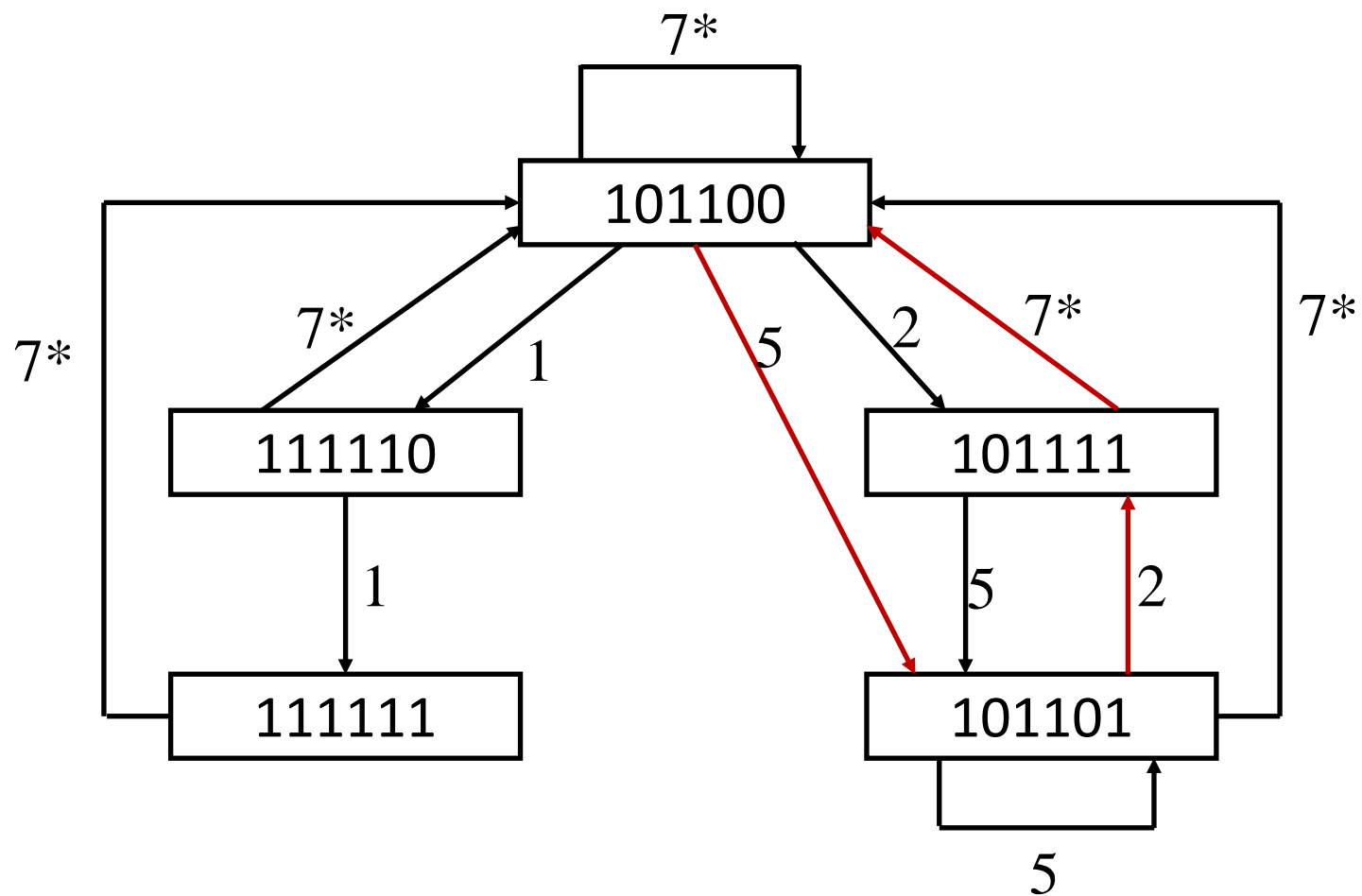
(2,5,7)



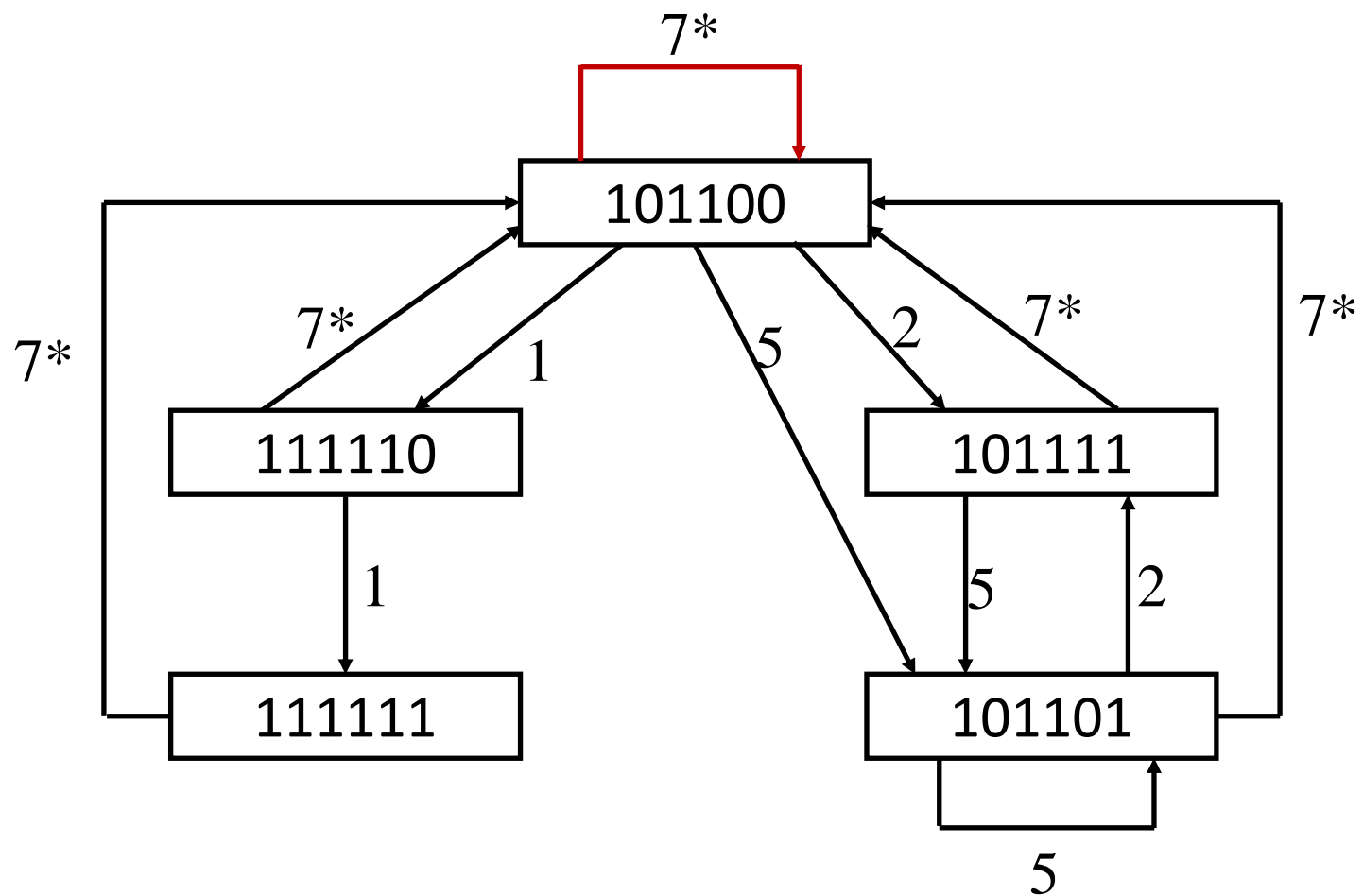
(5,7)



(5)



(5,2,7)



(7)

简单循环	平均启动距离
(1,7)	4
(1,1,7)	3
(2,7)	4.5
(2,5)	3.5
(2,5,7)	4.7
(5,7)	6
(5)	5
(5,2,7)	4.7
(7)	7

最小启动循环：(1,1,7)
 最小恒定启动循环：(5)

最小启动循环(1,1,7)的预约表

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S1	X1	X2	X3	X1	X2	X3	X1	X2	X3	X4	X5	X6	X4	X5	X6	X4
S2		X1	X2	X3	X1	X2	X3				X4	X5	X6	X4	X5	X6
S3		X1	X2	X3		X1	X2	X3			X4	X5	X6		X4	X5
S4			X1	X2	X3							X4	X5	X6		

例5.3：一条4功能段的非线性流水线，每个功能段的延迟时间都相等，它的预约表如下：

- (1) 写出流水线的禁止向量和初始冲突向量。
- (2) 画出调度流水线的状态图。
- (3) 求最小启动循环和最小平均启动距离。
- (4) 求平均启动距离最小的恒定循环。

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

解:

(1) 禁止向量为: $(2, 4, 6)$

初始冲突向量: $S = 101010$

(2) 构造状态图

S逻辑右移2、4、6位时, 不作任何处理,

逻辑右移1、3、5和大于等于7时:

S右移1位之后: $010101 \vee 101010 = 111111$,

S右移3位之后: $000101 \vee 101010 = 101111$,

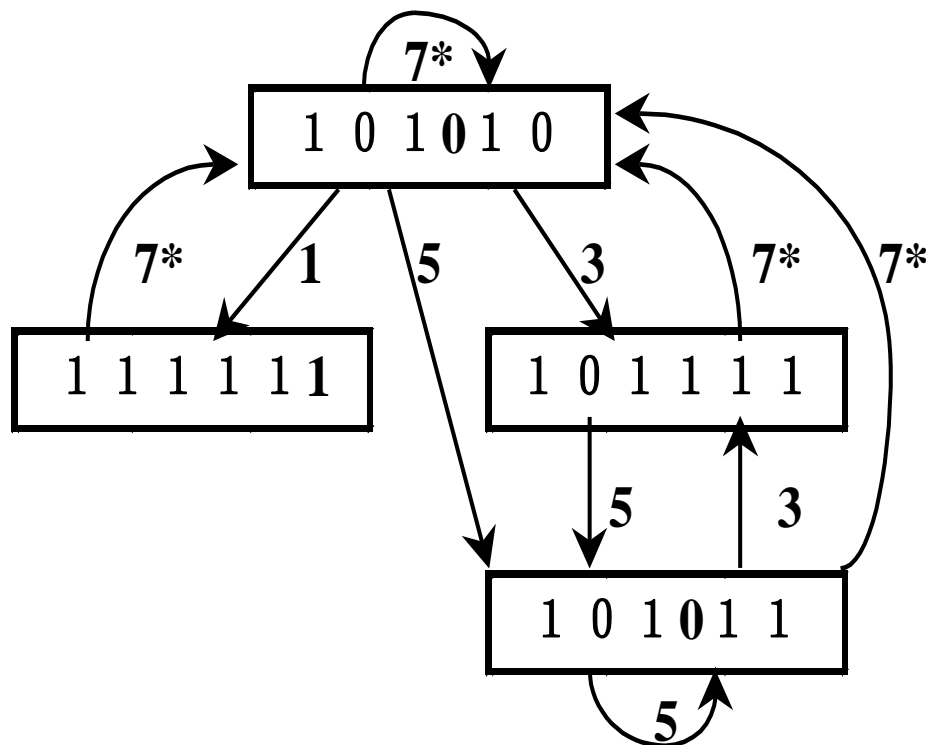
S右移5位之后: $000001 \vee 101010 = 101011$,

S右移7位或大于7位后: 还原到它本身。

101111右移5位之后: $000001 \vee 101010 = 101011$,

101011右移3位之后: $000101 \vee 101010 = 101111$,

101011右移5位之后: $000001 \vee 101010 = 101011$ 。



非线性流水线的状态图

简单循环：状态图中各种冲突向量只经过一次的启动循环。

(3) 最小的启动循环为 (1, 7)、(5, 3) 和 (3, 5), 平均启动距离为 4。

(4) 启动距离最小的恒定循环为 (5)

调度方案	平均启动距离
(1, 7)	4
(3, 5)	4
(5, 3)	4
(5, 7)	6
(3, 7)	5
(5, 3, 7)	5
(3, 5, 7)	5
(5)	5
(7)	7

最小启动循环(3, 5)的流水线工作状态

时间 功能段	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
S ₁	X ₁			X ₂			X ₁		X ₃	X ₂		X ₄			X ₃	...
S ₂		X ₁			X ₂	X ₁			X ₂	X ₃			X ₄	X ₃		...
S ₃			X ₁		X ₁	X ₂		X ₂			X ₃		X ₃	X ₄		...
S ₄				X ₁			X ₂					X ₃			X ₄	...

←
→
 启动周期

←
→
 重复启动周期

最小启动循环(1, 7)的流水线工作状态

时间 功能段	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
S ₁	X ₁	X ₂					X ₁	X ₂	X ₃	X ₄					X ₃	...
S ₂		X ₁	X ₂			X ₁	X ₂			X ₃	X ₄			X ₃	X ₄	...
S ₃			X ₁	X ₂	X ₁	X ₂					X ₃	X ₄	X ₃	X ₄		...
S ₄				X ₁	X ₂							X ₃	X ₄			...

←
→
 启动周期

←
→
 重复启动周期

恒定启动循环(5)的流水线工作状态

时间 功能段	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
S ₁	X ₁					X ₂	X ₁				X ₃	X ₂				...
S ₂		X ₁				X ₁	X ₂				X ₂	X ₃				...
S ₃			X ₁		X ₁			X ₂		X ₂			X ₃		X ₃	...
S ₄				X ₁					X ₂					X ₃		...

← 启动周期 →

← 重复启动周期 →

5.2.5 局部相关

1. **顺序流动方式：**任务按顺序流入流水线, 也按顺序流出流水线

把如下一段程序输入到这条流水线中：

k: **R0** ← (R1)

k+1:

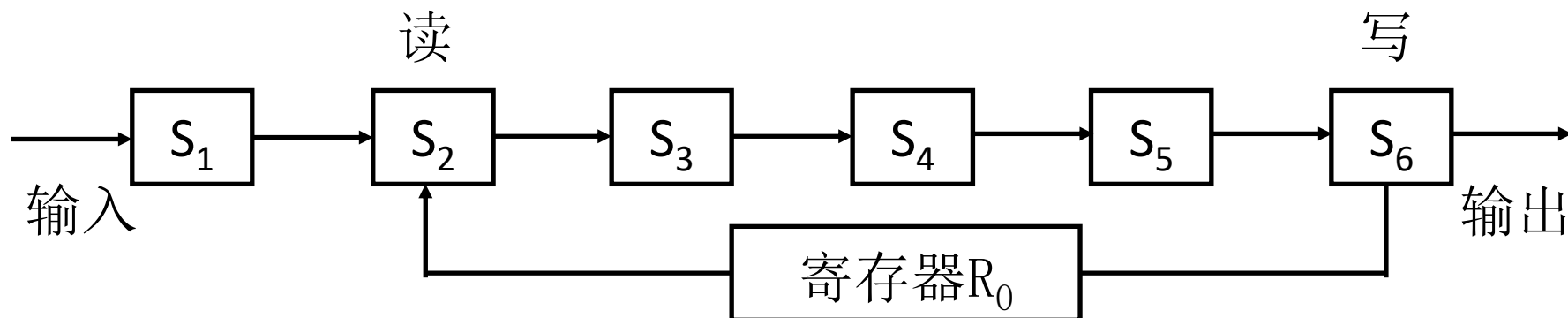
k+2: **R2** ← (R0) + (R3)

k+3:

k+4:

k+5:





- 指令 $k+2$ 无法继续执行，要在功能段 S_2 中等待。
- 后续的指令 $k+3$ 、 $k+4$ 、……等也不能进入流水线。
- 功能段 S_3 、 S_4 、 S_5 将逐渐空闲。
- 缺点：吞吐率和效率降低
- 优点：流水线的控制逻辑比较简单

	↑ 时钟周期						
t_{i+4}	k+4	k+3	k+2	空闲	空闲	空闲	
t_{i+3}	k+3	k+2	空闲	空闲	空闲	k+1	
t_{i+2}	k+3	k+2	空闲	空闲	k+1	k	
t_{i+1}	k+3	k+2	空闲	k+1	k	k-1	
t_i	k+3	k+2	k+1	k	k-1	k-2	
正常流动	k+5	k+4	k+3	k+2	k+1	k	
功能段	S_1	S_2	S_3	S_4	S_5	S_6	→ 功能段

顺序流动方式

➤ 流水线“断流”，有些功能段“空闲”

2. 乱序流动方式:

- 指令流出流水线的顺序与流入流水线的顺序不同。又称为错序流动方式、无序流动方式、异步流动方式等。

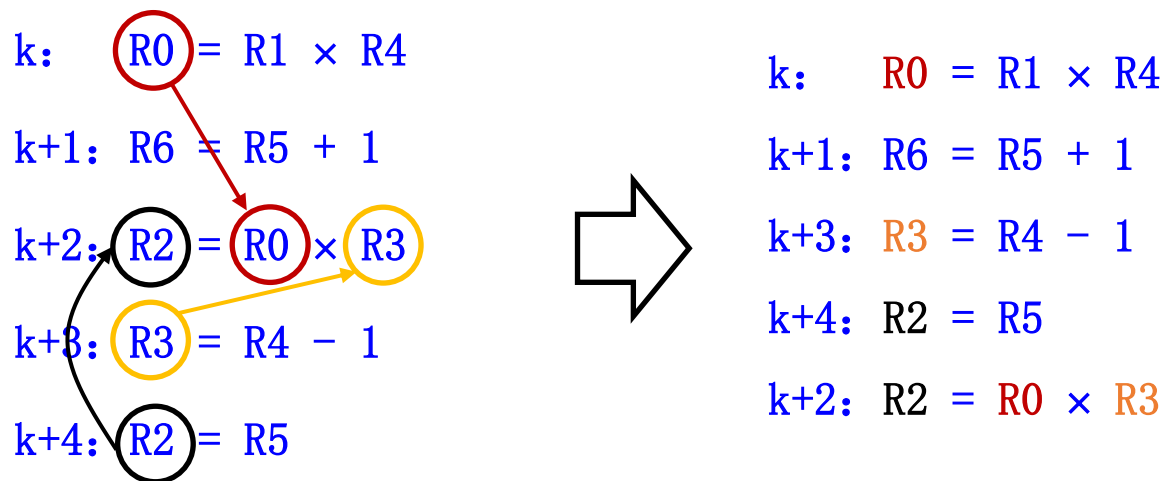
	↑时钟周期					
t_{i+5}	$k+8$	$\begin{matrix} (k+7) \\ k+6 \end{matrix}$	$k+2$	$K+5$	$k+4$	$k+3$
t_{i+4}	$k+7$	$\begin{matrix} (k+6) \\ k+2 \end{matrix}$	$k+5$	$K+4$	$k+3$	$k+1$
t_{i+3}	$k+6$	$\begin{matrix} k+5 \\ (k+2) \end{matrix}$	$k+4$	$k+3$	$k+1$	k
t_{i+2}	$k+5$	$\begin{matrix} k+4 \\ (k+2) \end{matrix}$	$k+3$	$k+1$	k	$k-1$
t_{i+1}	$k+4$	$\begin{matrix} k+3 \\ (k+2) \end{matrix}$	$k+1$	k	$K-1$	$k-2$
t_i	$k+3$	$\begin{matrix} (k+2) \\ k+1 \end{matrix}$	k	$k-1$	$k-1$	$k-3$
正常流动	$k+5$	$k+4$	$k+3$	$k+2$	$k+1$	k
功能段	S_1	S_2	S_3	S_4	S_5	S_6

→ 功能段

乱序流动方式

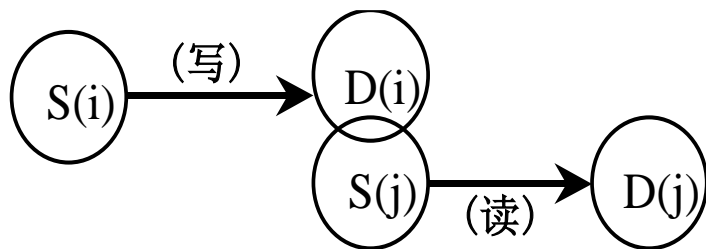
3. 乱序流动中的数据相关

在乱序流动方式中，可能发生三种数据相关

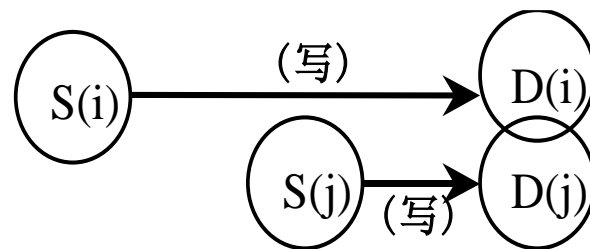


- 1) **写读相关**: 指令k与指令k+2之间关于R0的相关，又称为数据相关、先写后读相关、流相关、WR相关、RAW相关等。
- 2) **读写相关**: 指令k+2与指令k+3之间关于R3的相关，变量名相关、先读后写相关、反相关、RW相关、WAR相关等。
- 3) **写写相关**: 指令k+2与指令k+4左边的R2之间的相关关系称为：输出相关、写写相关、WW相关、WAW相关或写后再写相关等

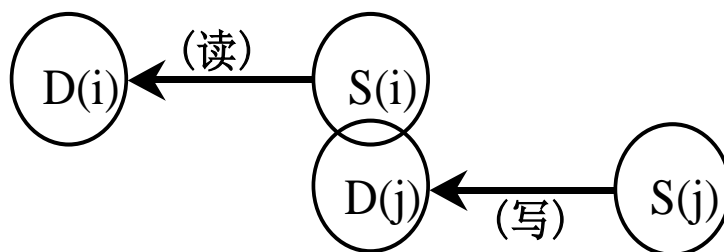
- 测试“先写后读”数据相关：在流水线读操作数功能段设置一个相联比较器，在指令读操作数之前，把源操作数地址与已经在流水线中的从读操作数功能段到写结果功能段之间的所有指令的目标地址进行比较，如果有一个地址是相等的，则表明发生了“先写后读”数据相关
- 测试“先读后写”和“写-写”数据相关：在流水线写结果功能段设置相联比较器，把自己的目标操作数地址分别与已经进入流水线的指令序号比自己小的源操作数地址和目标操作数地址进行比较，如果发现与某一条指令的源操作数地址相等，则说明发生了“先读后写”相关。如果发现与某一条指令的目标操作数地址等等，则发生了“写-写”相关。



(a) 写读相关



(b) 写写相关



(c) 读写相关

三种数据相关可以用下列关系式来表示：

对于写读相关 $D(i) \cap S(j) \neq \emptyset$

对于读写相关 $S(i) \cap D(j) \neq \emptyset$

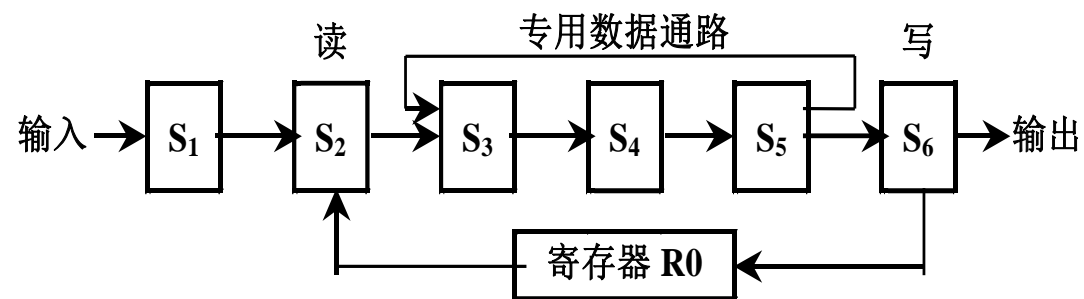
对于写写相关 $D(i) \cap D(j) \neq \emptyset$

4. 数据相关的避免方法

- 延迟执行

	↑时钟周期						
t_{i+4}	k+4	k+3	k+2	空闲	空闲	空闲	
t_{i+3}	k+3	k+2	空闲	空闲	空闲	k+1	
t_{i+2}	k+3	k+2	空闲	空闲	k+1	k	
t_{i+1}	k+3	k+2	空闲	k+1	k	k-1	
t_i	k+3	k+2	k+1	k	k-1	k-2	
正常流动	k+5	k+4	k+3	k+2	k+1	k	功能段 →
功能段	S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	

- 建立专用路径

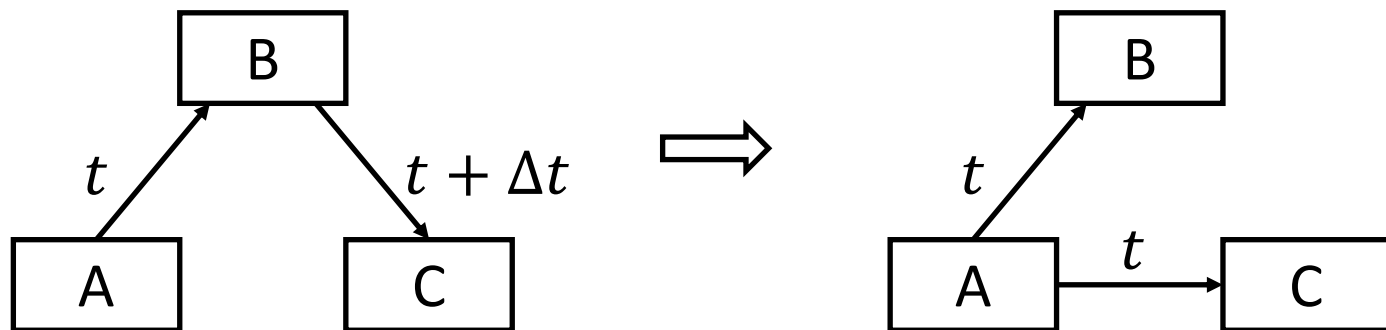


	时钟周期						
t_{i+3}	k+5	k+4	k+3	k+2	空闲	k+1	
t_{i+2}	k+4	k+3	k+2	空闲	k+1	k	
t_{i+1}	k+3	k+2	空闲	k+1	k	K-1	
t_i	k+3	k+2	k+1	k	K-1	K-2	
功能段	S_1	S_2	S_3	S_4	S_5	S_6	功能段

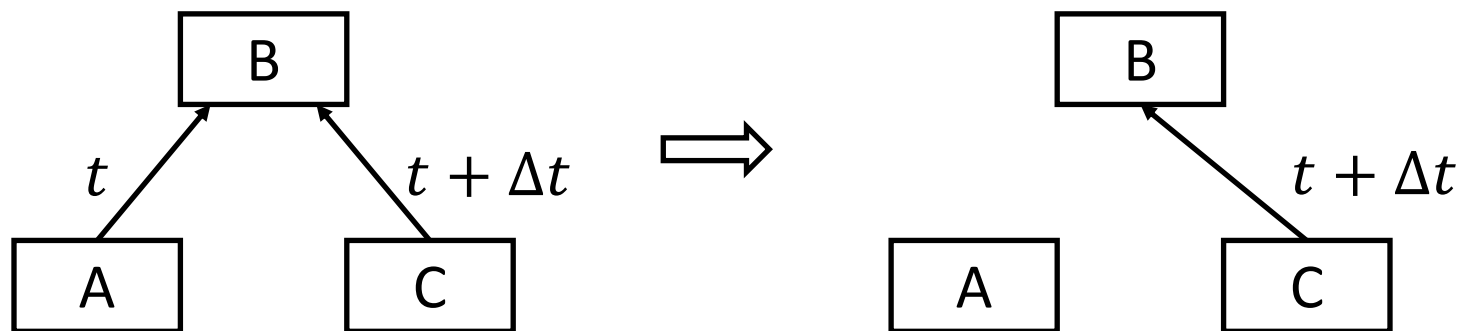
5. 数据重定向方法

1) 三种数据相关的重定向

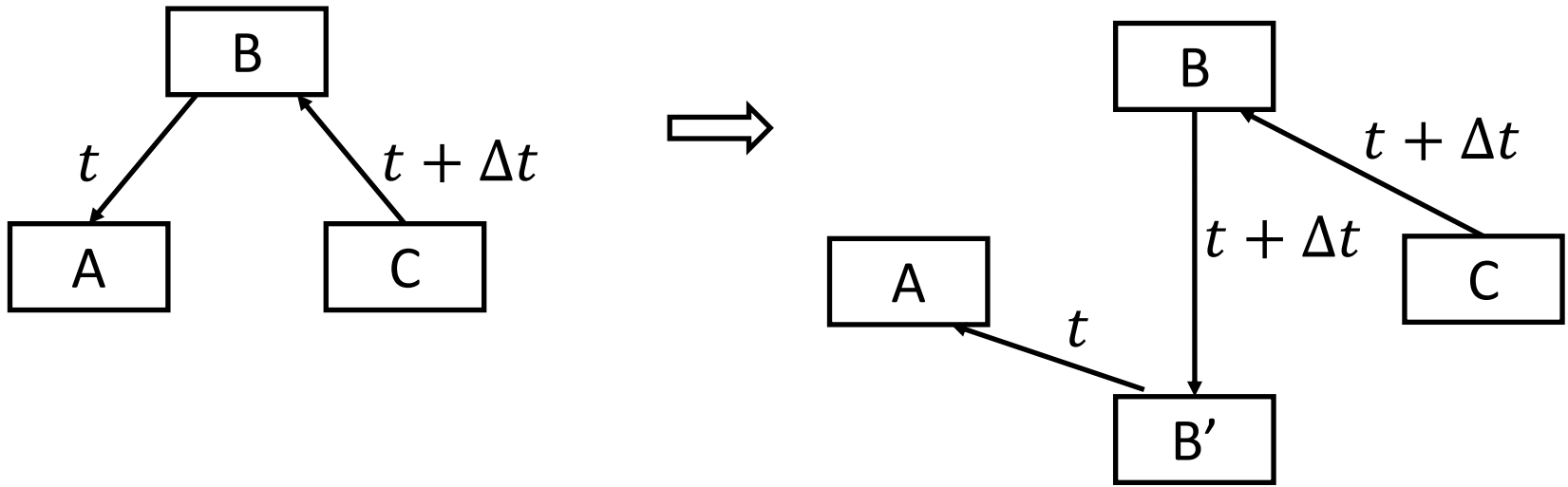
- 读写相关，可以同时执行
- 写写相关，先后顺序无关
- 读写相关，先后顺序无关



读写相关的数据重定向



写写相关的数据重定向

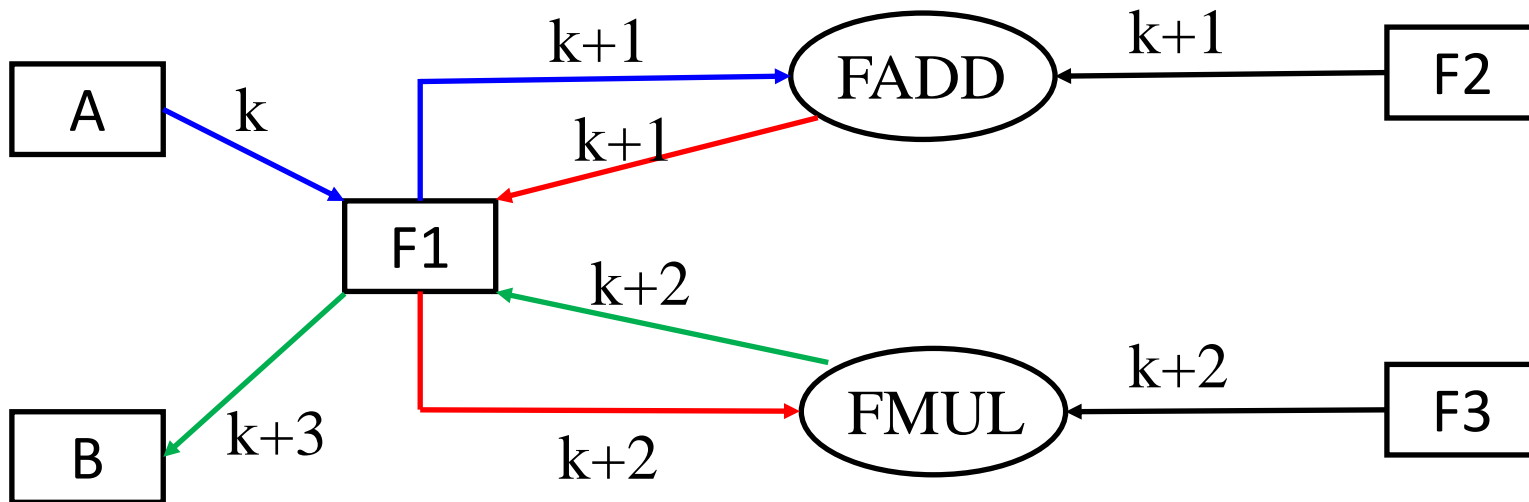


读写相关的数据重定向

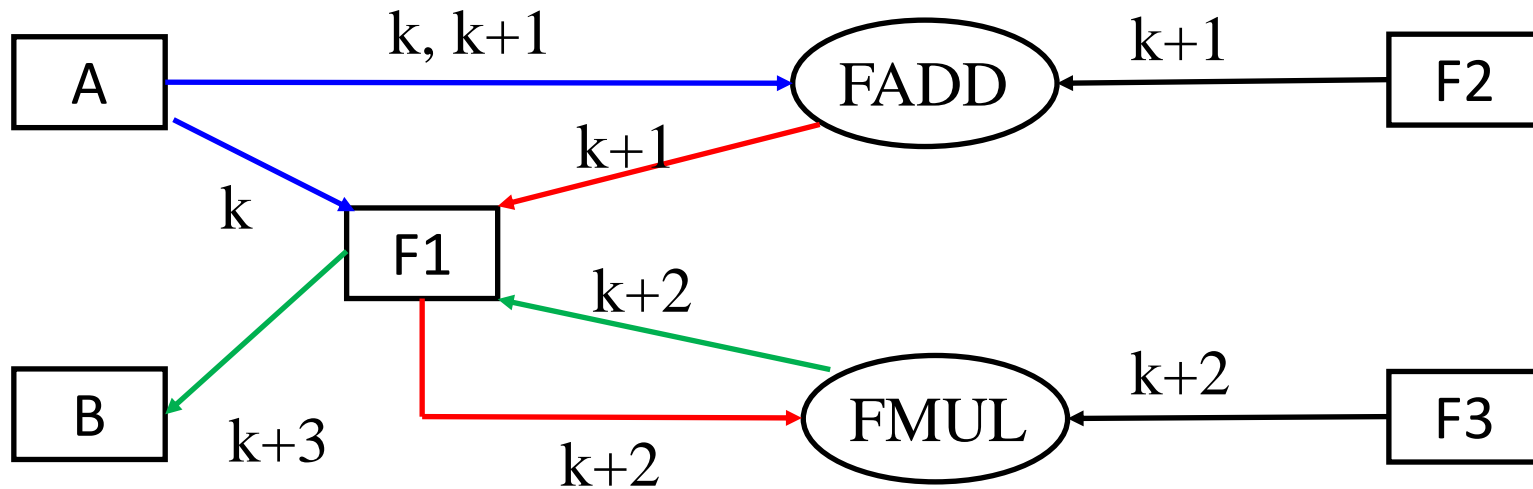
一个简单的程序：

k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B

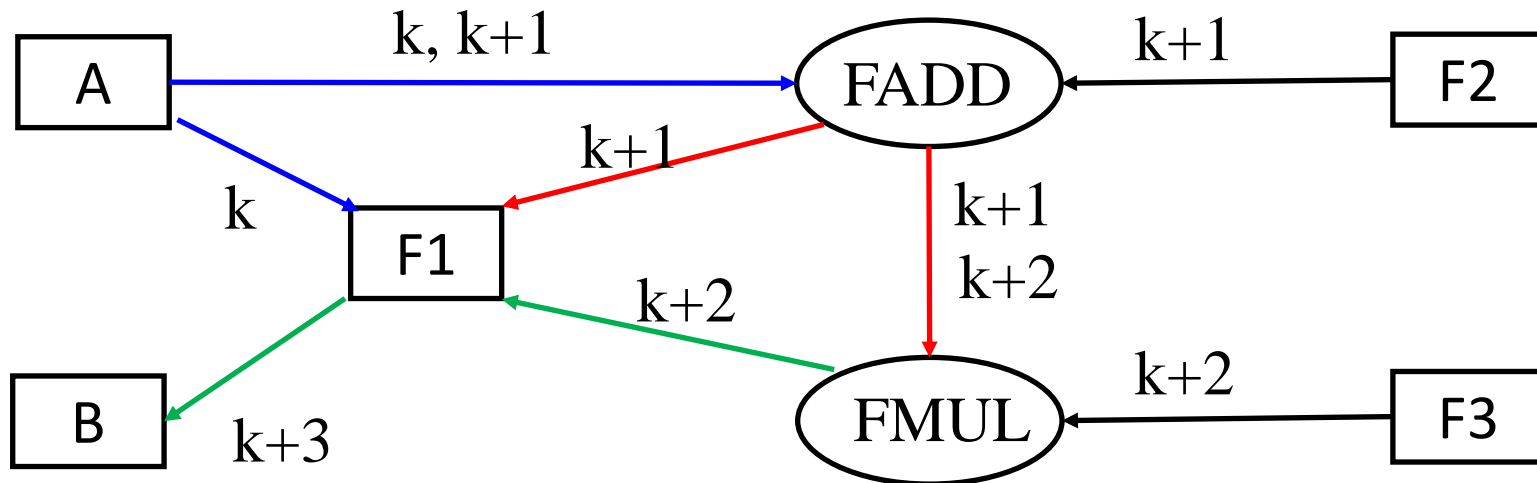
F1、F2、F3是浮点通用寄存器，
A和B是主存储器的单元



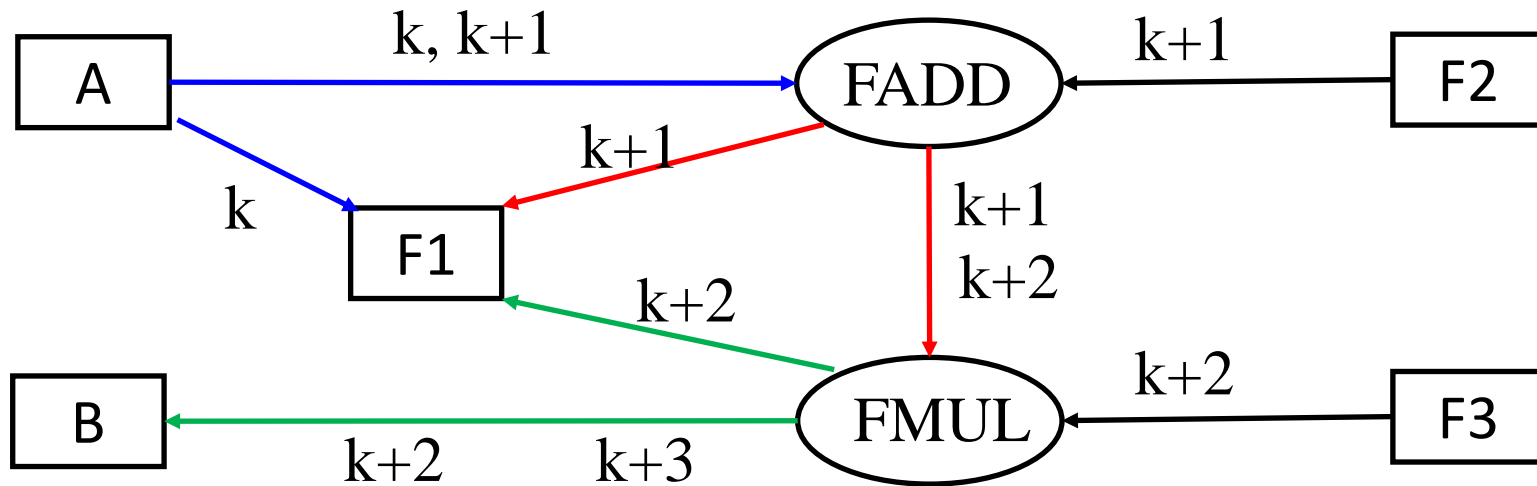
k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B



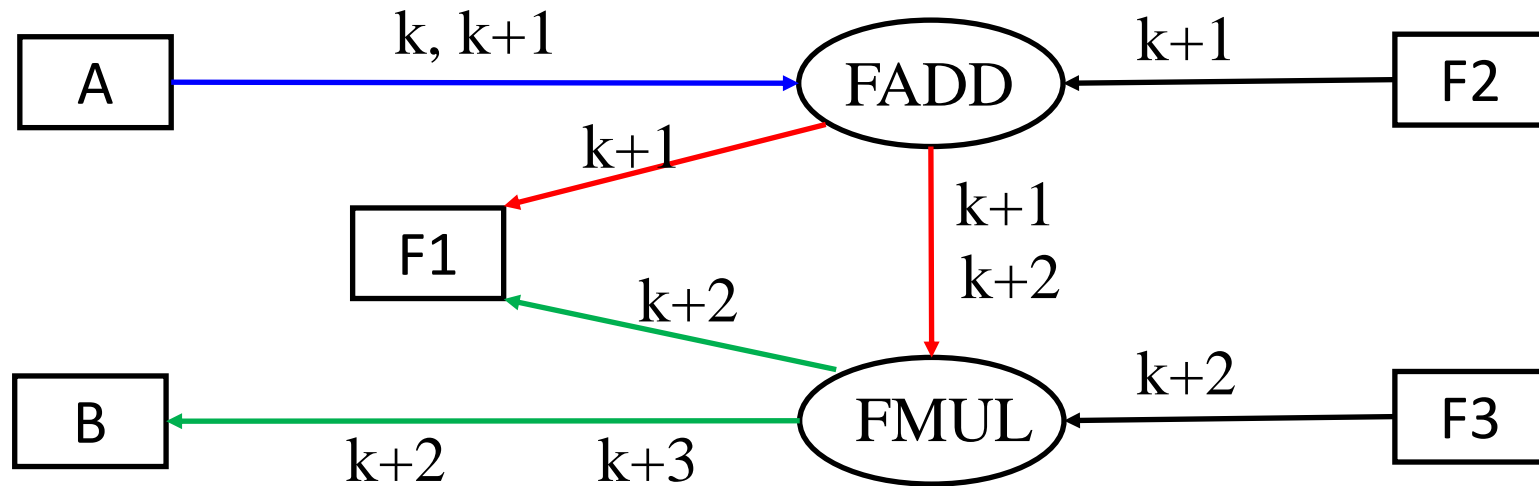
k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B



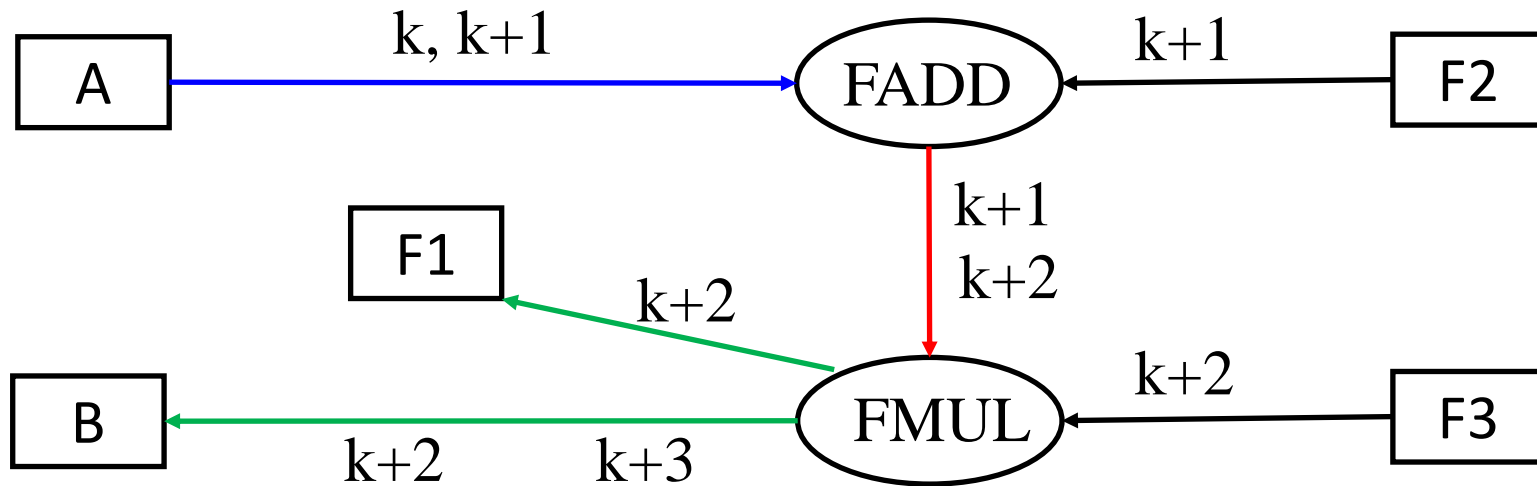
k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B



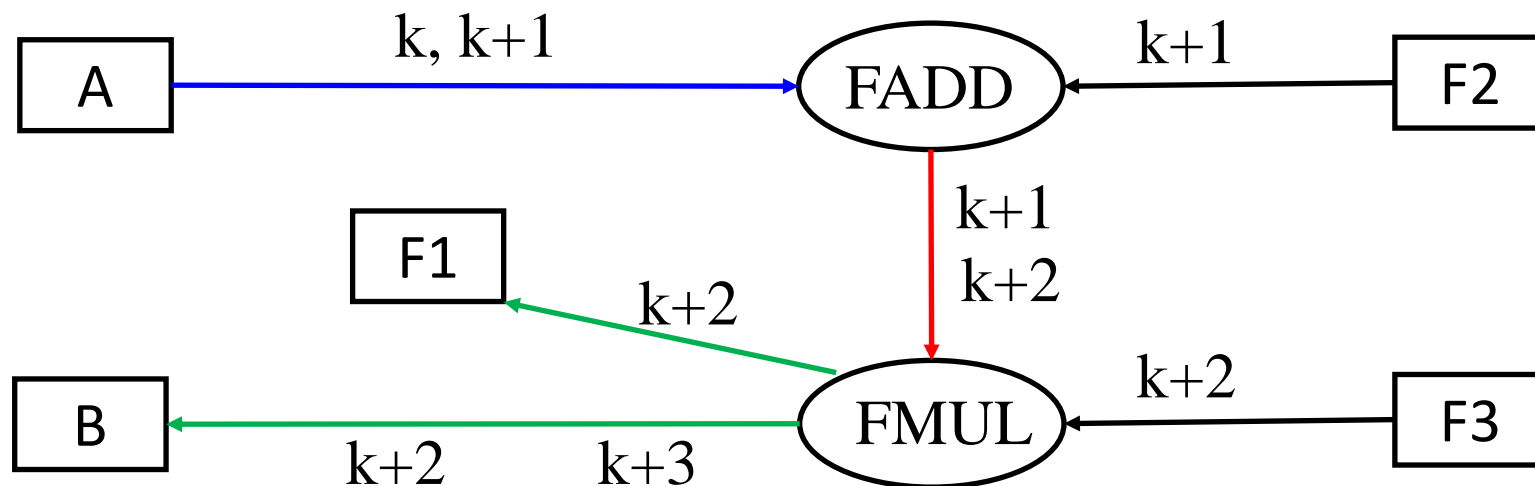
k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B



k: LOAD F1, A
k+1: FADD F1, F2
k+2: FMUL F1, F3
k+3: STORE F1, B



- 处理“先写后读”相关，专门设置： $A \rightarrow \text{FADD}$ 、 $\text{FMUL} \rightarrow B$ 、 $\text{FADD} \rightarrow \text{FMUL}$ 三条专用路径，撤销 $\text{F1} \rightarrow \text{FADD}$ 、 $\text{F1} \rightarrow \text{FMUL}$ 的路径
- 处理“写-写”相关，撤销 $\text{FADD} \rightarrow \text{F1}$ 、 $A \rightarrow \text{F1}$ 的路径。



- k : LOAD F1, A
- $k+1$: FADD F1, F2
- $k+2$: FMUL F1, F3
- $k+3$: STORE F1, B

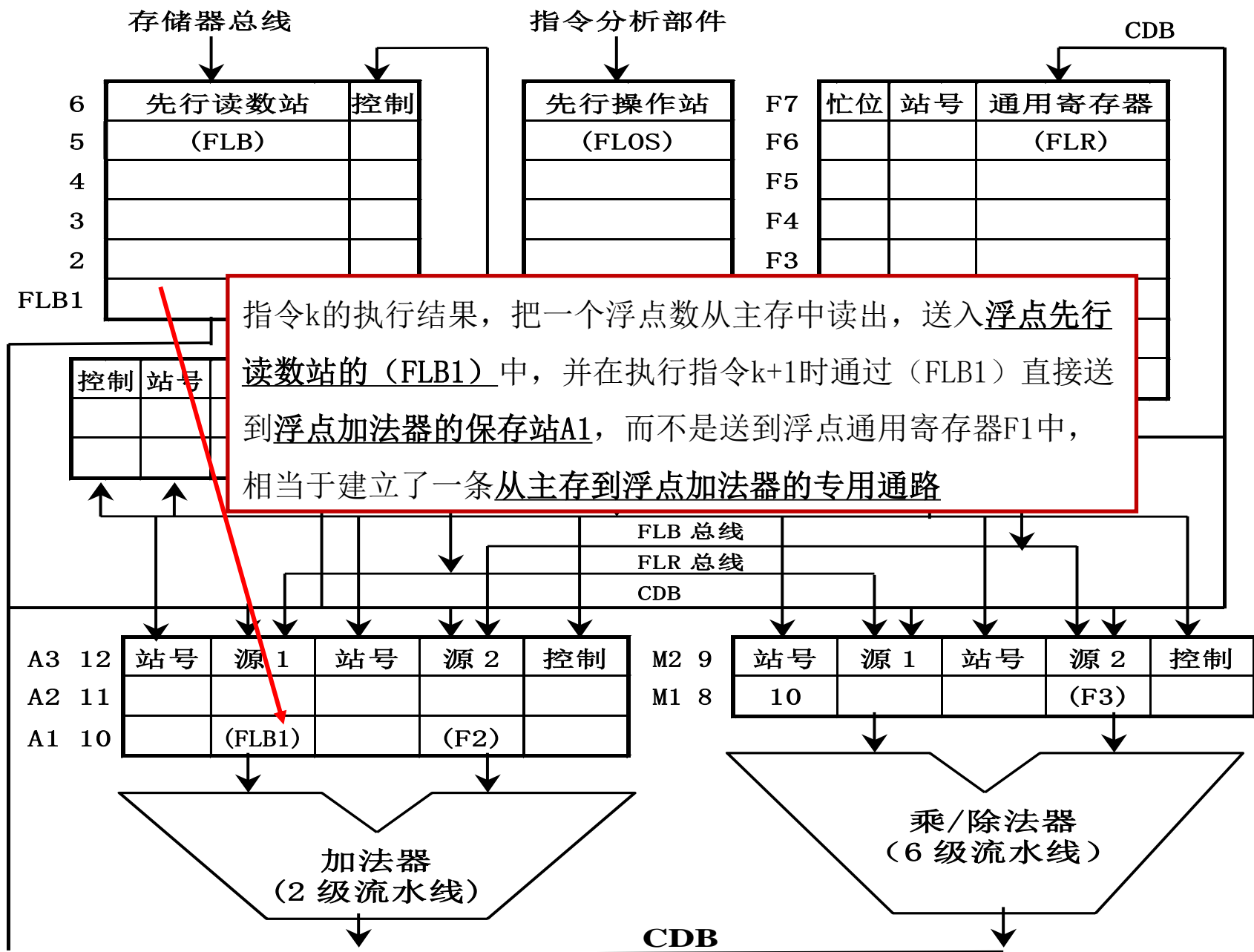
6. Tomasulo动态调度算法

实用的动态调度算法主要有两种：

- 1) 集中控制：CDC计分牌 (scorebord) 算法，最先在CDC 6600大型机中采用。
- 2) 分散控制：Tomasulo算法，公共数据总线法，令牌法等。最早在大型机IBM 360/91的浮点处理部件中被采用。

以上面的一段程序为例说明Tomasulo算法

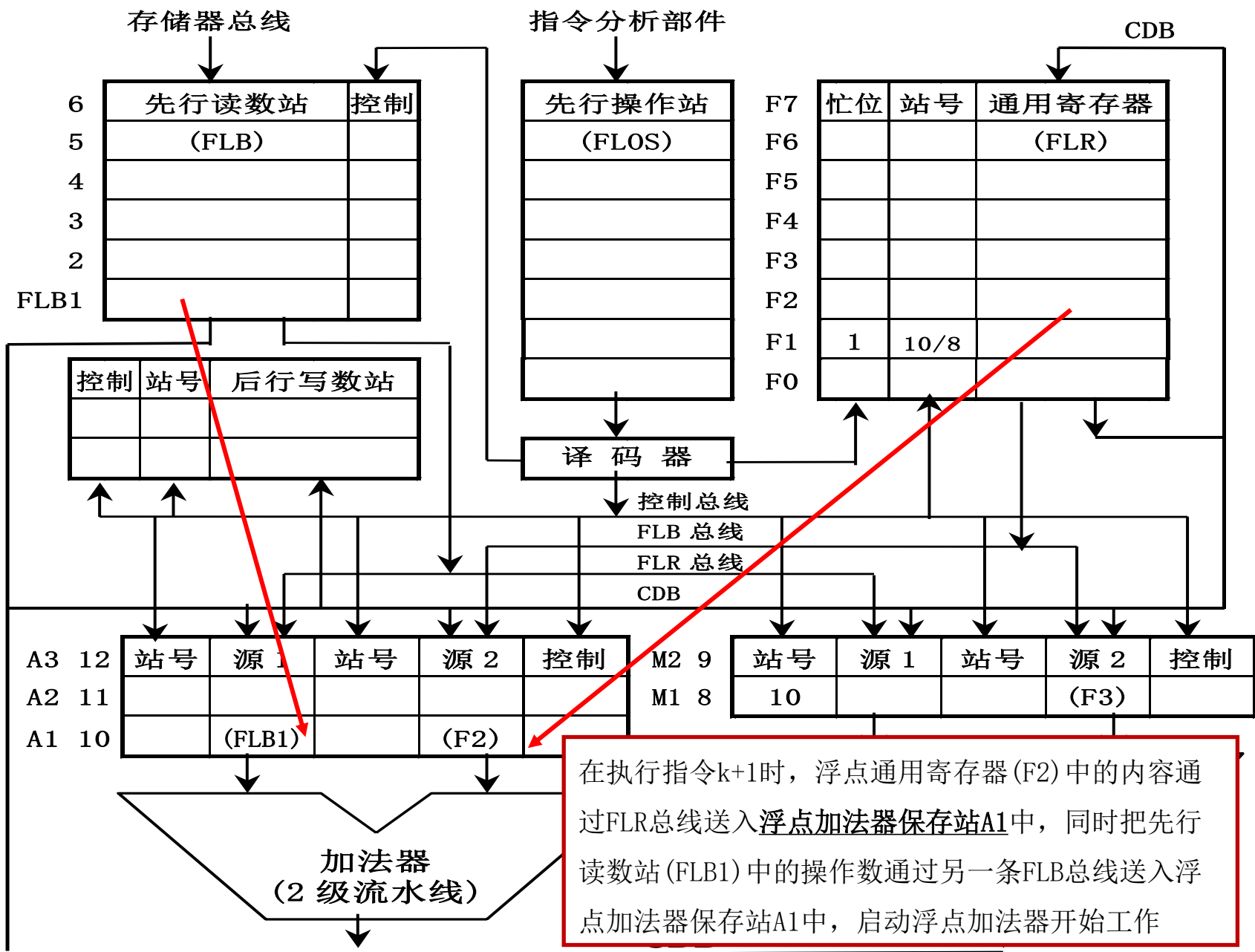
```
k:    LOAD    F1,  A
k+1:  FADD    F1,  F2
k+2:  FMUL    F1,  F3
k+3:  STORE   F1,  B
```

指令k的执行结果，把一个浮点数从主存中读出，送入浮点先行读数站的 (FLB1) 中，并在执行指令k+1时通过 (FLB1) 直接送到浮点加法器的保存站A1，而不是送到浮点通用寄存器F1中，相当于建立了一条从主存到浮点加法器的专用通路

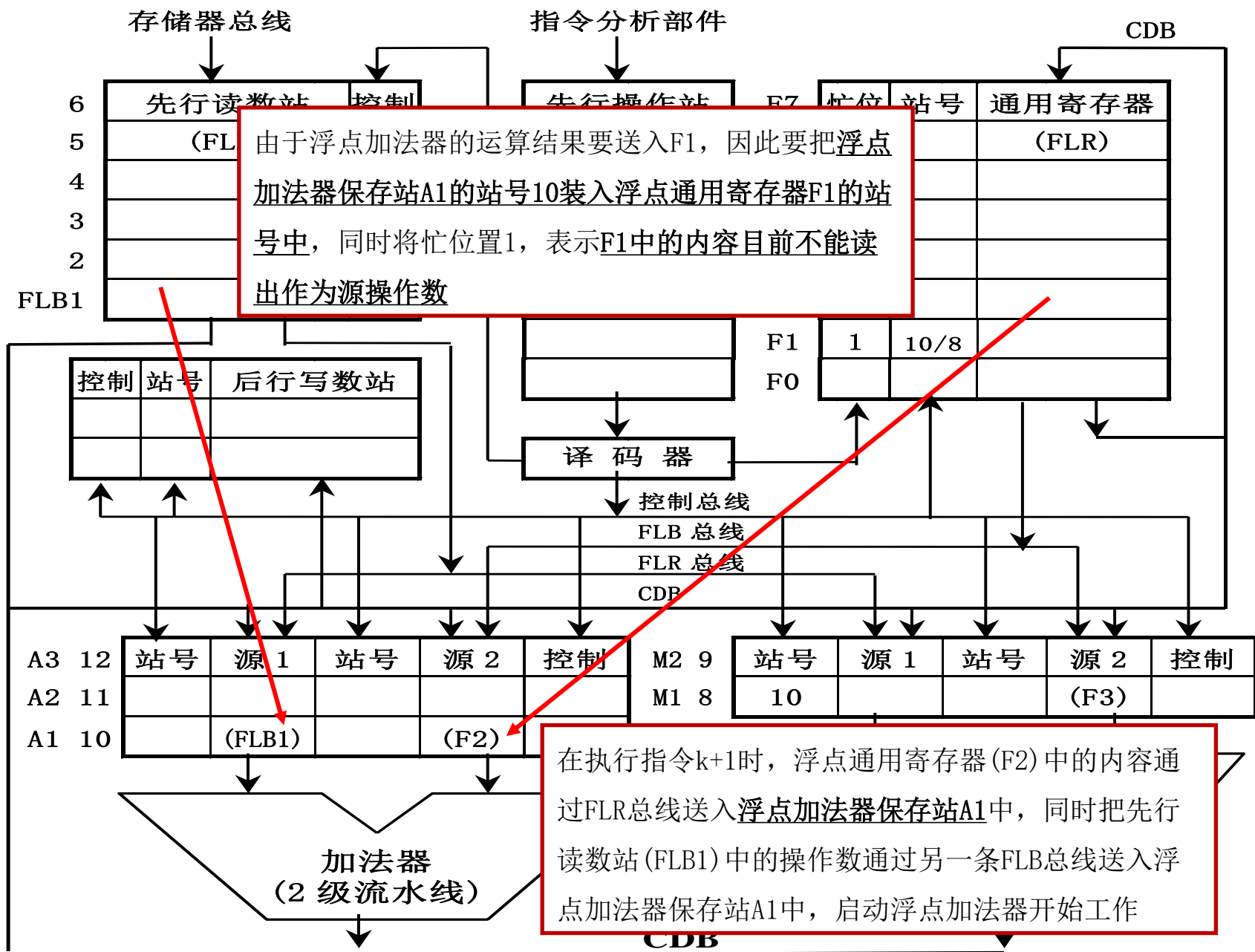
IBM360/91 处理机的浮点执行部件

k: **LOAD F1, A**



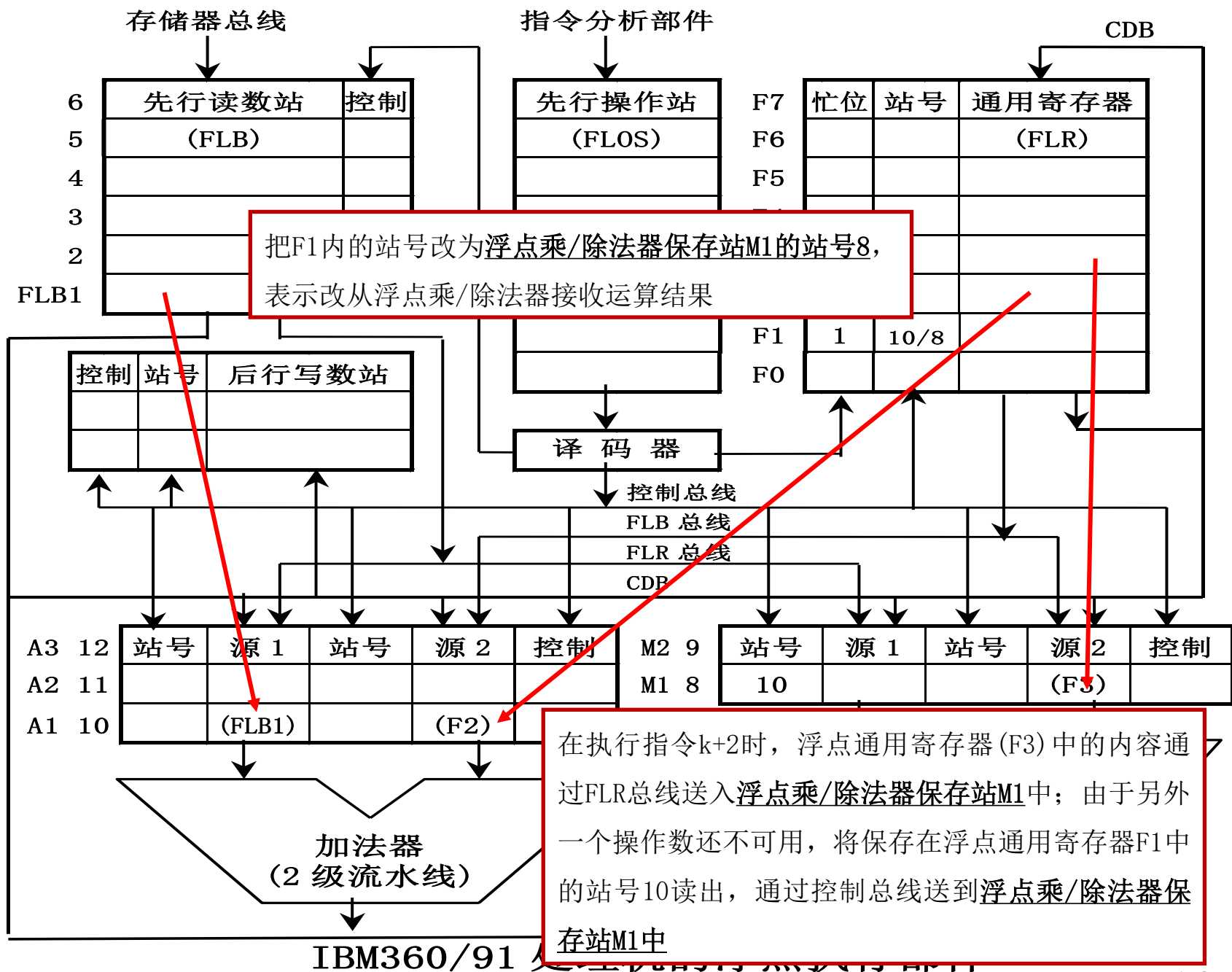
IBM360/91 处理机的浮点执行部件

k+1: FADD F1, F2



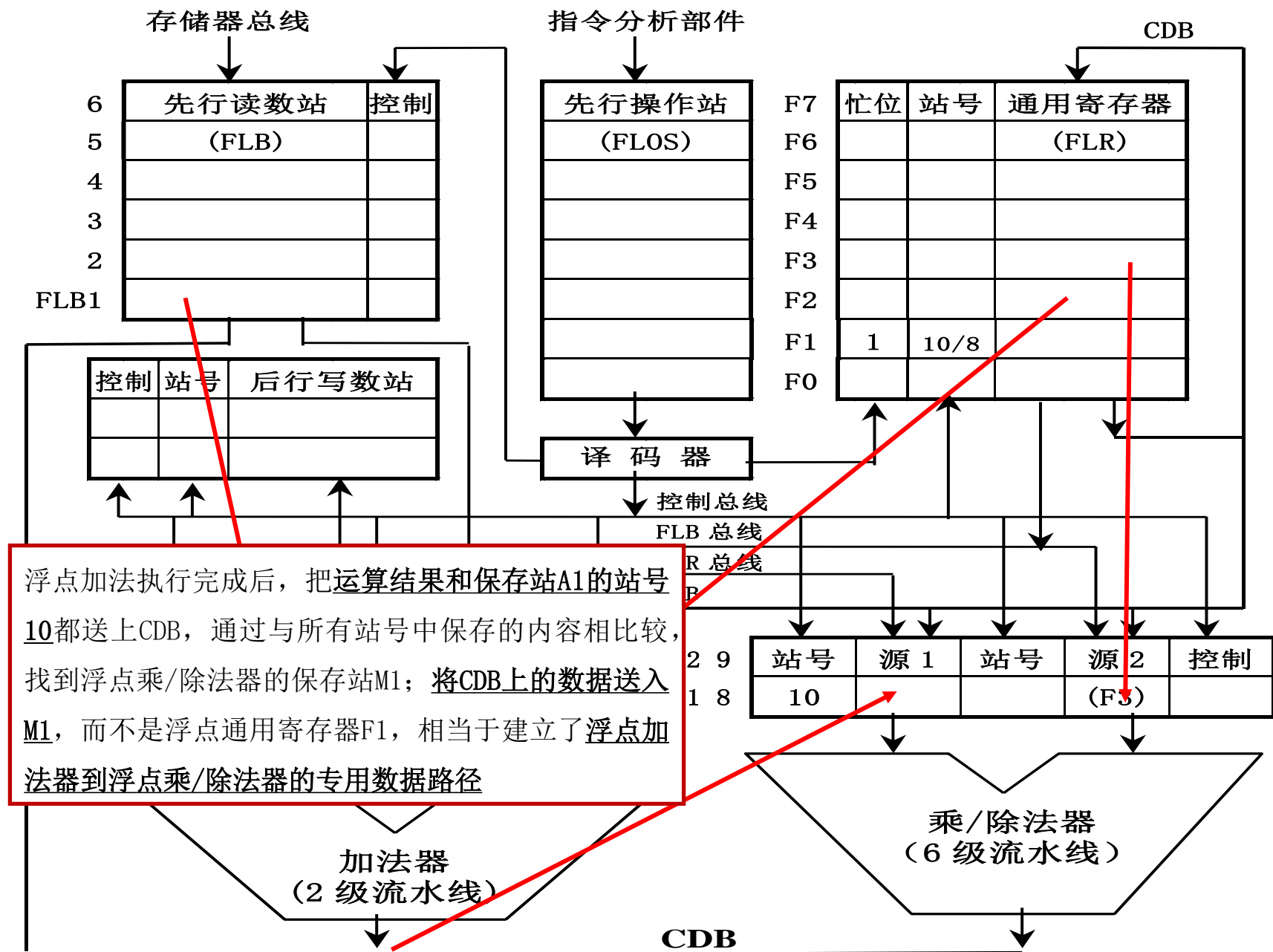
IBM360/91 处理机的浮点执行部件

$k+1$: FADD F1, F2

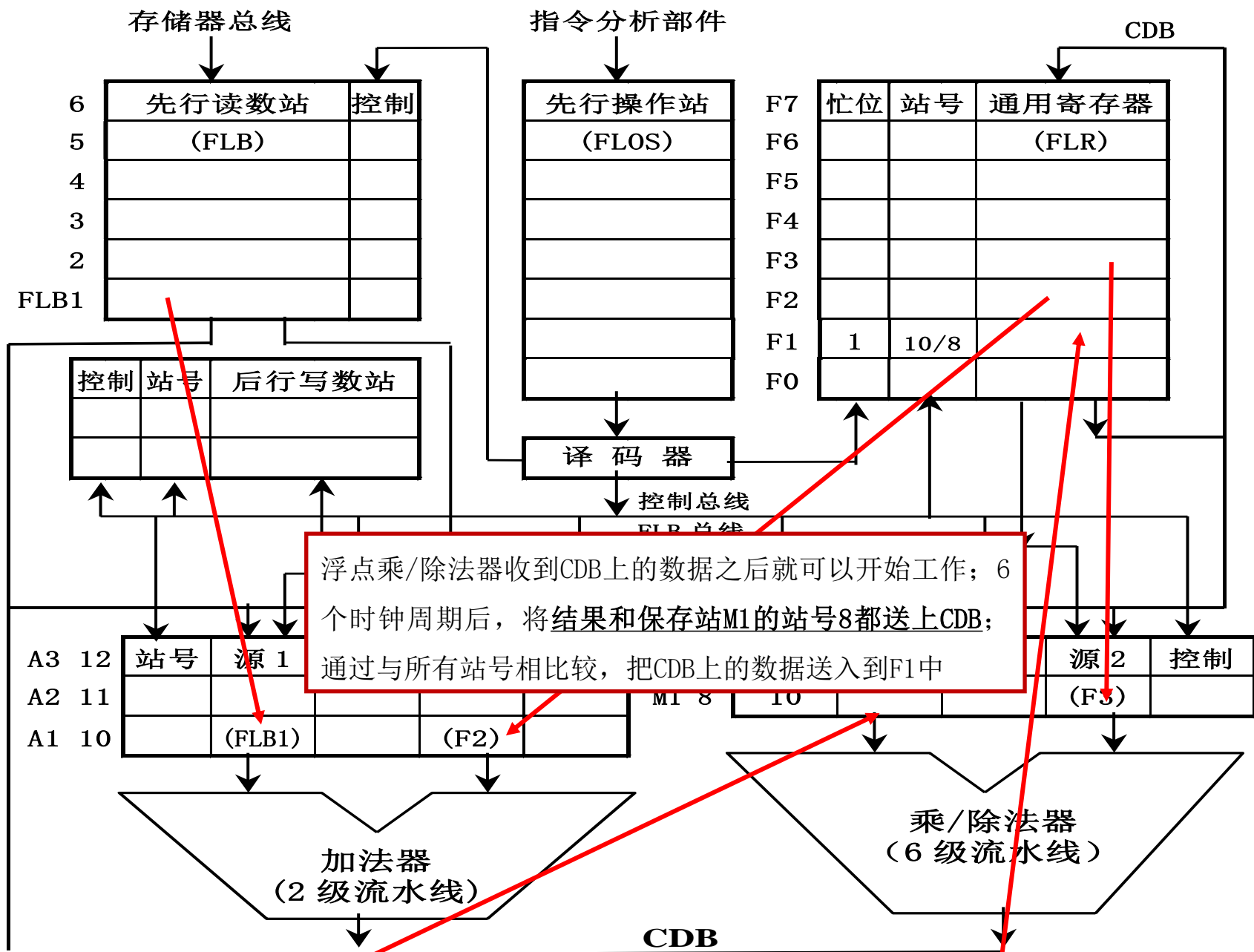


IBM360/91

k+2: FMUL F1, F3



浮点加法执行完成后，把运算结果和保存站A1的站号10都送上CDB，通过与所有站号中保存的内容相比较，找到浮点乘/除法器的保存站M1；将CDB上的数据送入M1，而不是浮点通用寄存器F1，相当于建立了浮点加法器到浮点乘/除法器的专用数据路径



IBM360/91 处理机的浮点执行部件

k+2: FMUL F1, F3

- 标量流水机对局部性相关的处理一般采用总线式分布方式控制管理：
- 相关的判断主要依靠分布于各个寄存器的“忙位”标志来管理
 - 在分散于个流水线的入、出端处设置若干保存站来缓存信息
 - 用站号控制公共数据总线的连接作相关专用通路，使之可为多个子过程的相关所共用
 - 一旦发生相关，用更换站号来推后和控制相关专用通路的连接
 - 采用多条流水线，每条流水线入端有多组保存站，以便发生相关后，可以采用异步的流动方式

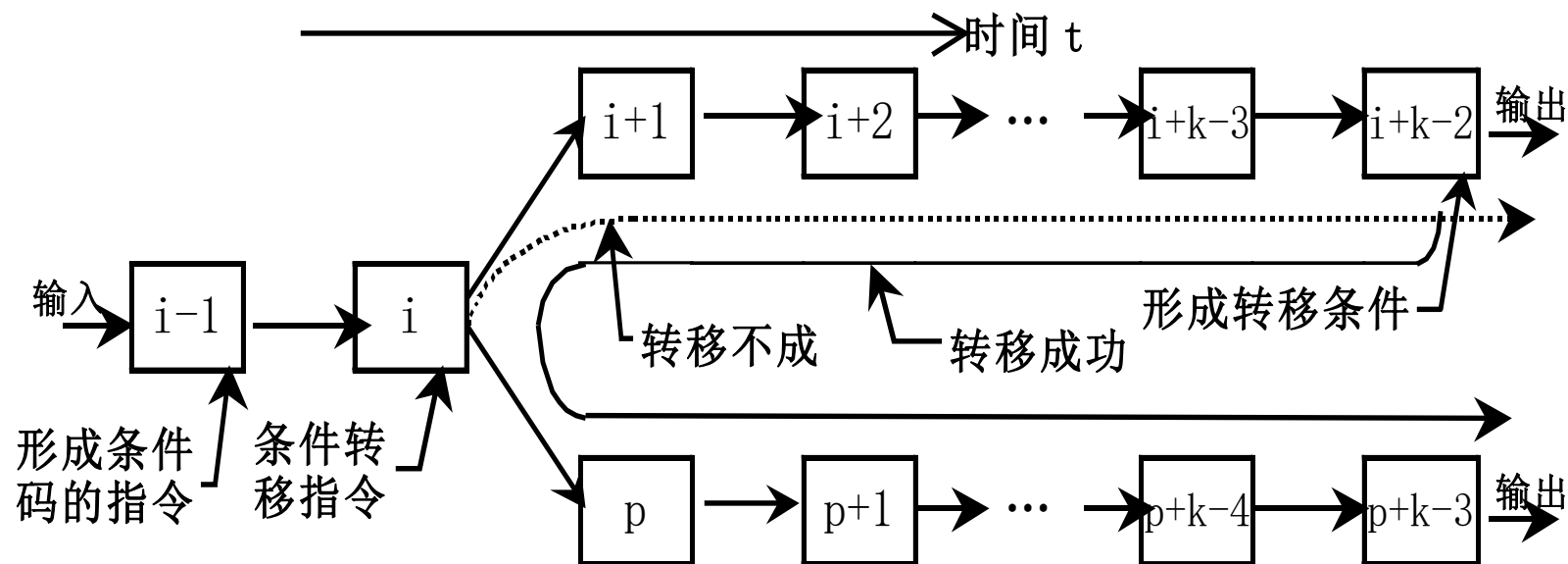
5.2.6 全局相关

- 由条件转移或程序中断引起的相关成为全局相关
- 处理好条件转移和中断的关键问题有两个：
 - 要确保流水线能够正常工作
 - 减少因“断流”引起的吞吐率和效率的下降

1. 条件分支在流水线中的执行过程

- 因为第 i 条指令所需要的条件码由第 $i-1$ 条指令给出；在一条由 k 个功能段的流水线中，第 $i-1$ 条指令要等到第 $i+k-2$ 条指令进入流水线时才能形成条件码。
- 转移不成功，猜测正确，流水线的吞吐率和效率没有降低，
- 转移成功，猜测错误，要先作废流水线中已经执行的 $i+1$ 、 $i+2$ 、……、 $i+k-2$ 指令；然后再从分支点开始执行第 p 、 $p+1$ 、……指令。一条 k 段流水线有 $k-2$ 个功能段是浪费的。

- 条件分支指令在流水线中的执行过程



- 当分支的执行方向猜测错误时，可能造成程序执行结果发生错误。
 - 例如，若第 $i+1$ 条指令是： $(R1 + (R2)) \rightarrow R1$ ，寄存器 $R1$ 中内容就被破坏，整个程序执行的结果是错误的。
- 目前的处理机有两种做法：
 - 一种方法是只进行指令译码和准备好运算所需要的操作数，在转移条件没有形成之前不执行运算；
 - 另一种方法是一直执行到运算完成，但不送回运算结果。

2. 条件分支对流水线性能的影响

- 假设条件转移指令在一般程序中所占的比例为 p ，转移成功的概率为 q 。
- n 条指令的总的执行时间是：

$$T_{K-IF} = (n + k - 1)\Delta t + npq(k - 1)\Delta t$$

- 有条件转移影响的流水线吞吐率为：

$$TP_{IF} = \frac{n}{(n + k - 1)\Delta t + npq(k - 1)\Delta t}$$

- 有条件转移影响的流水线最大吞吐率为：

$$TP_{MAX-IF} = \frac{1}{(1 + pq(k - 1))\Delta t}$$

- 流水线吞吐率下降的百分比为：

$$D = \frac{TPMAX - TP_{MAX} - IF}{TPMAX} = \frac{pq(k-1)}{1 + pq(k-1)}$$

- 在典型程序中，转移指令占的比例为 $p=20\%$ ，转移成功的概率为 $q=60\%$ 。
- 对于一条8功能段的指令流水线，由于条件转移指令的影响，流水线的最大吞吐率要下降：

$$D_8 = \frac{0.20 \times 0.60 \times (8-1)}{1 + 0.20 \times 0.60 \times (8-1)} = 46\%$$

- 如果指令流水线的功能段数为10，由于条件转移指令的影响，流水线的最大吞吐率将下降一半以下：

$$D_{10} = \frac{0.20 \times 0.60 \times (10-1)}{1 + 0.20 \times 0.60 \times (10-1)} = 53\%$$

3. 条件分支的处理方法

- 条件转移指令对流水线的影响很大，必须采取措施来减少这种影响。
- 可能的措施有：

(1) 延迟转移技术和指令取消技术

- 只能用于单流水线处理机中，且流水线的级数不能太多；
- 据统计，编译器调度一条指令成功的概率在90%以上，而调度两条指令成功的概率只有40%左右。
- 当没有合适的指令可调度时，编译器只能插入空操作。

(2) 动态分支预测技术

- 根据近期转移是否成功的记录来预测下一次转移的方向。
- 所有的动态转移预测方法都能够随程序的执行过程动态地改变转移的预测方向。

(3) 静态分支预测技术

- 转移预测的方向是确定的，或者预测转移不成功，或者预测转移成功，
- 在程序实际执行过程中，转移预测的方向不能改变。
- 静态转移预测可以只用软件实现，也可用硬件来实现，还可以在转移的两个方向上都预取指令。
- TI公司的SuperSPARC处理机采用了静态转移预测技术，而且设置有转移目标缓冲栈，在两个方向上都预取指令。

(3) 提前形成条件码

4 动态分支预测技术

➤ 动态转移预测技术的两个关键问题：

- 如何记录转移历史信息
- 如何根据记录的转移历史信息预测转移方向

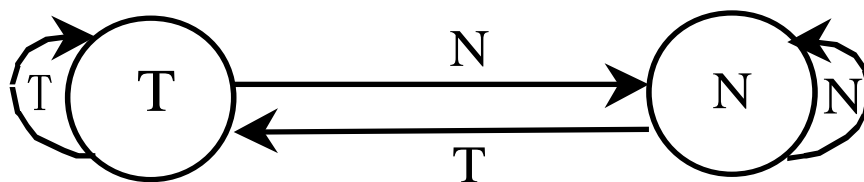
➤ 记录转移历史信息的方法有三种：

- (1) 最近一次或几次转移是否成功的信息记录在指令cache中
- (2) 用一个高速缓冲栈保存条件转移指令的转移目标地址
- (3) 用Cache保存转移目标地址之后的n条指令

4 动态分支预测技术

1) 在指令Cache中记录转移历史信息

- 在指令Cache中专门设置一个字段，称为“转移历史表”。
- 在执行转移指令时，把转移成功或不成功的信息记录在这个表中。
- 当下次再执行到这条指令时，转移预测逻辑根据“转移历史表”中记录的信息预测转移成功或不成功。



T: 转移成功, N: 转移不成功

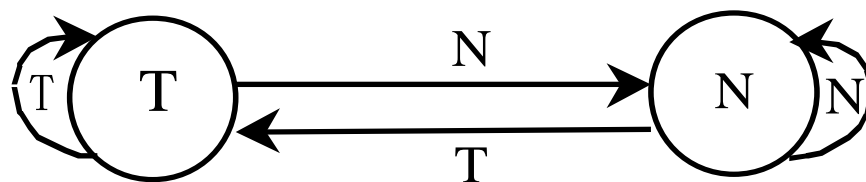
4 动态分支预测技术

1) 在指令Cache中记录转移历史信息

➤ 只记录最近一次转移是否成功的历史信息

➤ 如果“转移历史表”中记录的内容是“T”，则预测转移成功，如果记录的是“N”，则按照转移不成功的方向继续取指令。

➤ 并用实际转移是否成功的信息来修改“转移历史表”



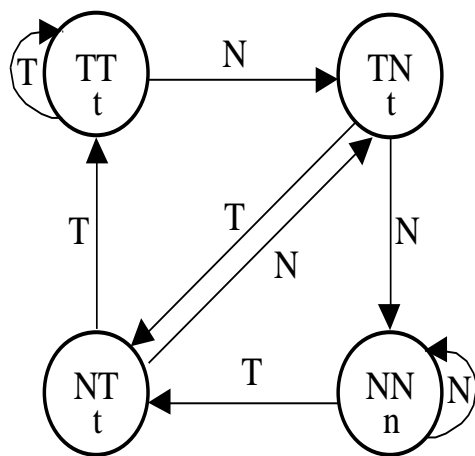
T: 转移成功, N: 转移不成功

4 动态分支预测技术

1) 在指令Cache中记录转移历史信息

➤ 记录最近两次转移是否成功的历史信息

- 图中采用偏向成功的预测策略：只有历史上最近两次执行这条转移指令时转移都没有成功，本次才预测转移不成功
- 也可以采用其他预测策略



TT: 最近两次转移都成功
NT: 最近一次转移成功，前一次转移不成功
TN: 最近一次转移不成功，前一次转移成功
NN: 最近两次转移都不成功
t: 本次预测转移成功
n: 本次预测转移不成功
T: 本次实际转移成功
N: 本次实际转移不成功

4 动态分支预测技术

1) 在指令Cache中记录转移历史信息

- “转移历史表”的修改规则和转移预测规则可以有多种多样，记录转移预测是否成功的信息。
- 用最近预测是否成功的信息作为是否转移的依据
- 当“转移历史表”是空白时，可以有两种做法：
 - 在“转移历史表”中预置转移历史信息。
 - 根据指令本身的偏移字段的符号来预测转移的方向，如果偏移字段为负，则预测转移成功，否则预测转移不成功

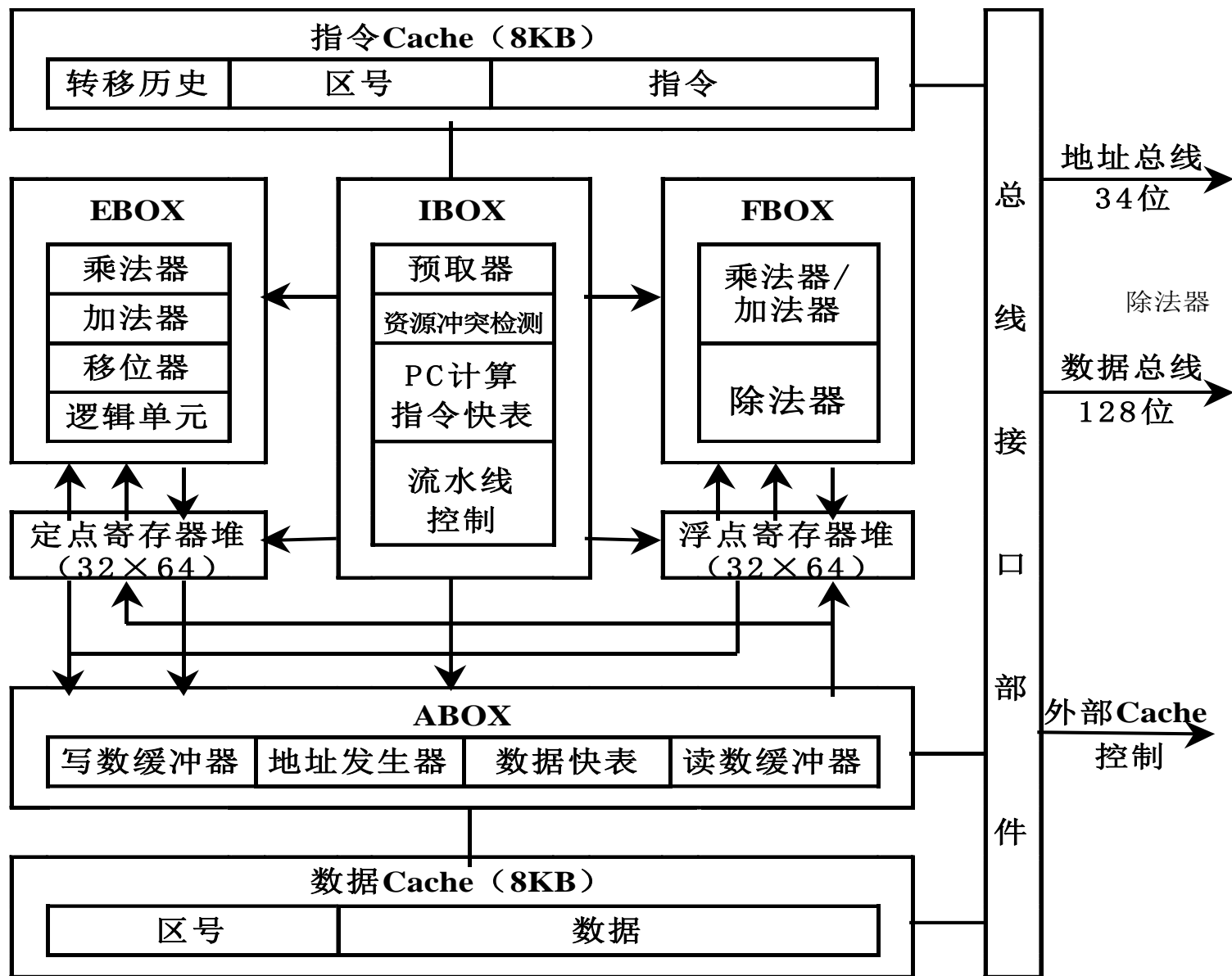
4 动态分支预测技术

1) 在指令Cache中记录转移历史信息

➤ 主要优点：

- 不必专门设置转移缓冲栈，
- 所记录的转移历史信息比较少。

➤ 例如：DEC公司的Alpha 21064处理机就采用了这种转移预测方法，在它的一级指令Cache中有一个专门的“转移历史表”字段。



Alpha 21064 处理器结构

4 动态分支预测技术

2) 设置转移目标地址缓冲栈

- 用高速缓冲栈保存最近k条转移指令的“转移历史表”和转移目标地址
- 当前指令地址与转移目标缓冲栈中的所有转移指令地址进行比较；如果发现相等的，则根据所记录的历史信息预测本次转移方向。
- 根据某种规则修改“转移历史表”。

I_0	T_0	P_0
I_1	T_1	P_1
\vdots	\vdots	\vdots
I_{k-1}	T_{k-1}	P_{k-1}
转移指令地址	转移历史表	转移目标地址

4 动态分支预测技术

3) 设置转移目标指令缓冲栈

- 把上面方法中的“转移目标地址”字段改为存放转移目标地址之后的n条指令。
- 预测转移方向的规则和修改“转移历史表”的规则与上面的方法相同。

A_0	T_0	$I_{0,0}$	$I_{0,1}$	$I_{0,n-1}$
A_1	T_1	$I_{1,0}$	$I_{1,1}$	$I_{1,n-1}$
A_2	T_2	$I_{2,0}$	$I_{2,1}$	$I_{2,n-1}$
\vdots	\vdots	\vdots			
A_{k-1}	T_{k-1}	$I_{k-1,0}$	$I_{k-1,1}$	$I_{k-1,n-1}$
转移指令地址	转移历史表	转移目标地址之后的 n 条指令			

5 提前形成条件码

- 必要性：对提高流水线的性能非常有效
- 可能性：可在运算开始或中间产生条件码
- 对于乘除法，两个源操作数的符号相同结果为正，符号相反结果为负
- 对于乘法，有一个操作数为0，则乘积为0
- 被除数为0，商为0；除数为0，除法结果溢出
- 同号加或异号减，结果符号与第一操作数相同
- 异号加或同号减，结果的符号与绝对值大的操作数相同
- 溢出及是否为0可以通过一个比较器提前产生

5 提前形成条件码

- 只要在一个时钟周期之内产生条件码，流水线就不会“断流”。
- 如Amdahl470V/6在运算部件的入口处设置一个LOCK部件，提前形成条件码
- 把产生条件码与使用条件码的指令分开

LOAD R1, NUM ;循环次数初值装入R1

LOOP: ;循体开始

.....

DEC R1 ;循环次数减“1”

BNE LOOP ;测试循环是否则结束

HALT ;程序结束

NUM: n

可以编译成如下程序：

LOAD R1, NUM ;循环次数装入R1中

LOOP: LDEC R1 ;一条专用的循环次数减1指令

..... ;循体开始

.....

LBNE LOOP ;一条专用的测试循环是否结束的指令

HALT ;程序结束

NUM: n ;循环次数

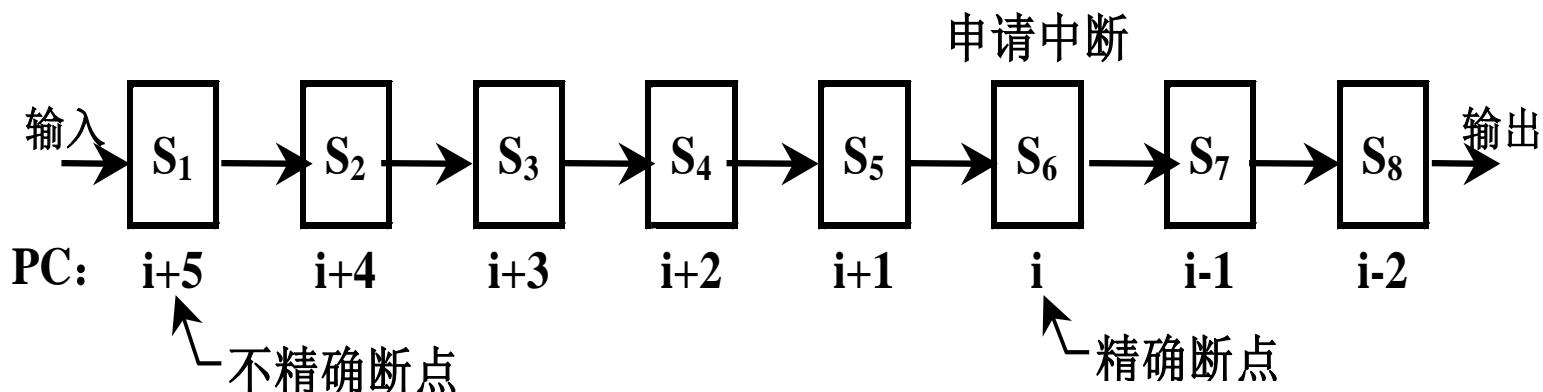
指令LDEC和LBNE使用专用的条件码寄存器

6 精确断点与不精确断点

- 对于输入输出设备的中断服务，实际上不需要有精确断点。比较简单的处理方法是：让已经进入流水线的所有指令都执行完成，断点就是最后进入流水线的那条指令的地址。
- 对于程序性错误和机器故障等引起的中断，它们出现的概率很低，处理原则：不在于缩短时间，关键是要正确保存现场和正确恢复断点。

6 精确断点与不精确断点

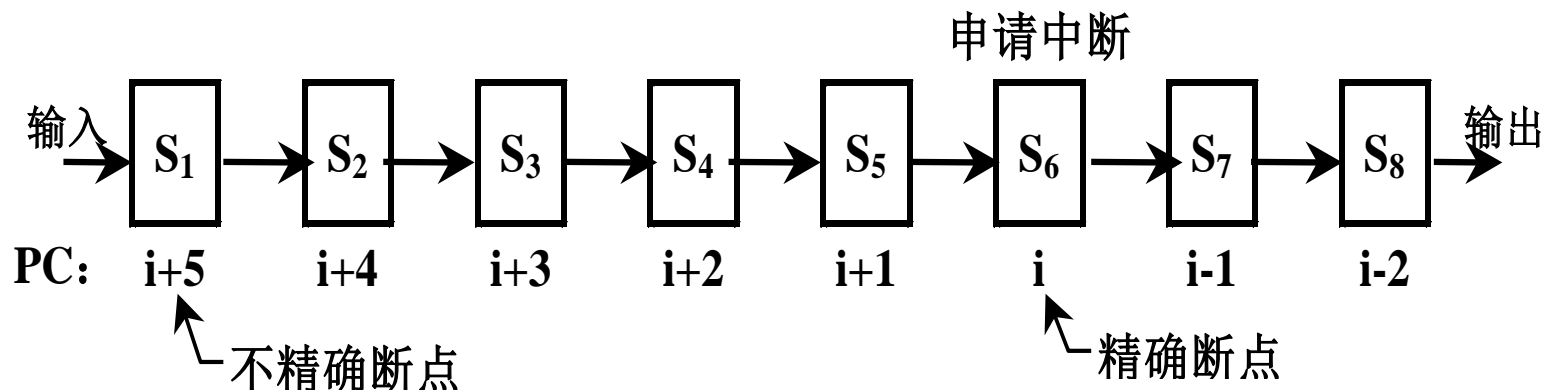
- 对于输入输出设备的中断服务，实际上不需要有精确断点比较简单的处理方法是：让已经进入流水线的指令都执行完成，断点就是最后进入流水线的那条指令的地址。
- 对于程序性错误和机器故障等引起的中断，它们出现的概率很低，处理原则：不在于缩短时间，关键是要正确保存现场和正确恢复断点。



早期的流水线处理机，多采用不精确断点法；近期的流水线处理机一般都采用精确断点法

6 精确断点与不精确断点

- 不精确断点 (Imprecise)，流水线可以不断流，需要的硬件比较少，控制逻辑比较简单，中断响应时间加长
- 采用不精确断点法可能会发生如下两个问题：
 - (1) 程序的调试困难
 - (2) 程序执行的结果可能出错



6 精确断点与不精确断点

► 采用不精确断点法可能会发生如下两个问题：

(1) 程序的调试困难

(2) 程序执行的结果可能出错

i: FADD R1, R2 ; $(R1) + (R2) \rightarrow R1$

i+1: FMUL R3, R1 ; $(R3) \times (R1) \rightarrow R3$

- 当第i条指令执行到S₆段时发现浮点加法结果溢出，于是发出中断服务申请。由于采用不精确断点法，已经进入流水线的第i+1条指令将执行完成；因为第i+1条指令使用了不正确的R1，所以浮点乘法的执行结果是不正确的。

6 精确断点与不精确断点

➤ 采用不精确断点法可能会发生如下两个问题：

(1) 程序的调试困难

(2) 程序执行的结果可能出错

➤ 采用精确(Precise)断点法，要设置一定数量的后援寄存器，把整个流水线中所有指令的执行结果和现场都保存下来。

5.3 超标量处理机

5.3.1 基本结构

5.3.2 单发射与多发射

5.3.3 多流水线调度

5.3.4 资源冲突

5.3.5 超标量处理机性能

三种高性能主流处理机：

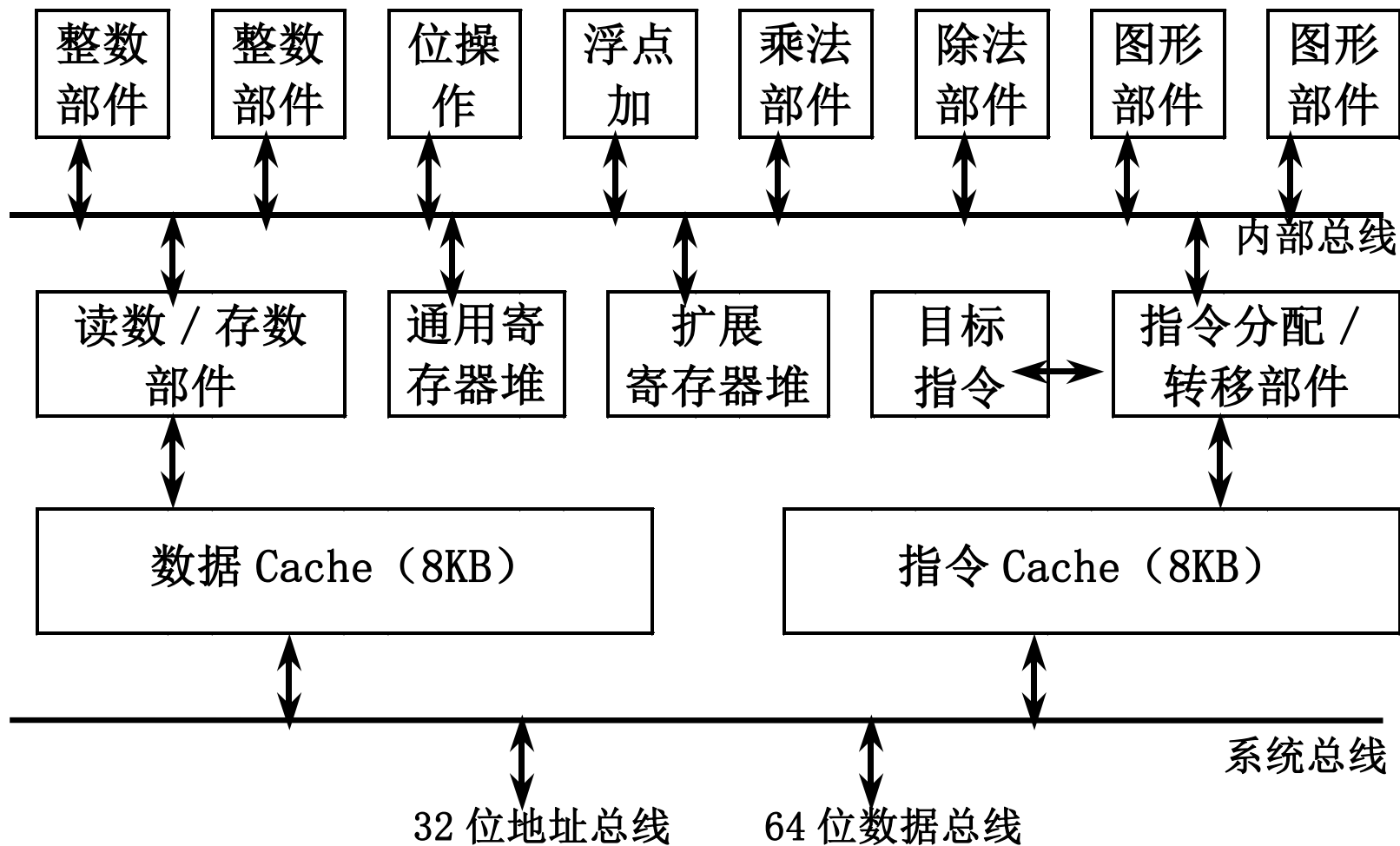
- 超标量处理机（superscalar processor）
- 超流水线处理机（superpipelining processor）
- 超标量超流水线处理机（superpipelining superscalar processor）

以一台 k 段流水线的普通标量处理机为基准
 超标量处理机、超流水线处理机和超标量超流水线处理机的主要性能：

机器类型	k 段流水线 标量处理机	m 度 超标量处理机	n 度 超流水线处理机	(m,n)度超标量 超流水线处理机
机器流水线周期	1 个时钟周期	1	$1 / n$	$1 / n$
同时发射指令条数	1 条	m	1	m
指令发射等待时间	1 个时钟周期	1	$1 / n$	$1 / n$
指令级并行度 ILP	1	m	n	$m \times n$

5.3.1 基本结构

- 普通标量流水线处理机：一条指令流水线，一个多功能操作部件，每个时钟周期平均执行指令的条数小于1。
- 多操作部件标量处理机：一条指令流水线，多个独立的操作部件，指令级并行度小于1。
- 超标量处理机典型结构：多条并行工作的指令流水线，多个独立的操作部件，指令级并行度（ILP）大于1。



超标量处理机 MC88110 的结构

Motorola公司的MC88110

- 有10个操作部件
- 两个寄存器堆：
 - 整数部件通用寄存器堆，32个32位寄存器
 - 浮点部件扩展寄存器堆，32个80位寄存器
- 缓冲深度为4的先行读数栈
- 缓冲深度为3的后行写数栈
- 两个独立的高速Cache中，各为8KB，采用两路组相联方式
- 转移目标指令Cache，用于存放另一条分支上的指令

5.3.2 单发射与多发射

1. 单发射处理机：

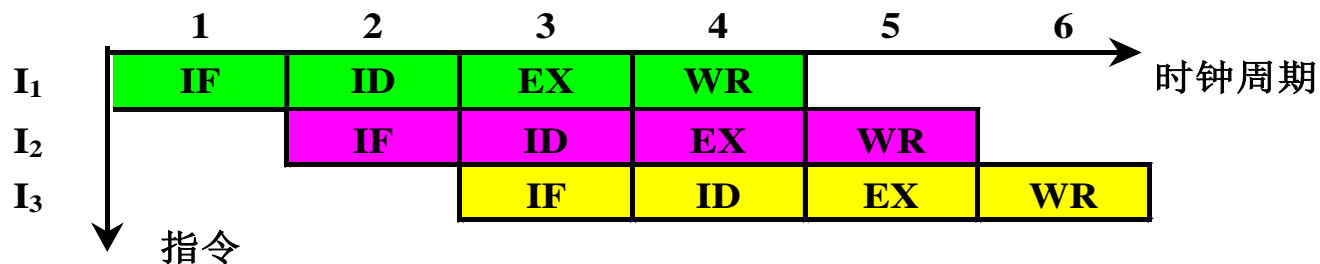
- 每个周期只取一条指令、只译码一条指令，只执行一条指令，只写回一个运算结果。
- 取指令部件和指令译码部件各设置一套；
- 只设置一个多功能操作部件或设置多个独立的操作部件；
- 操作部件中可以采用流水线结构，也可以不采用流水线结构。
- 目标是每个时钟周期平均执行一条指令，ILP的期望值为1。

5.3.2 单发射与多发射

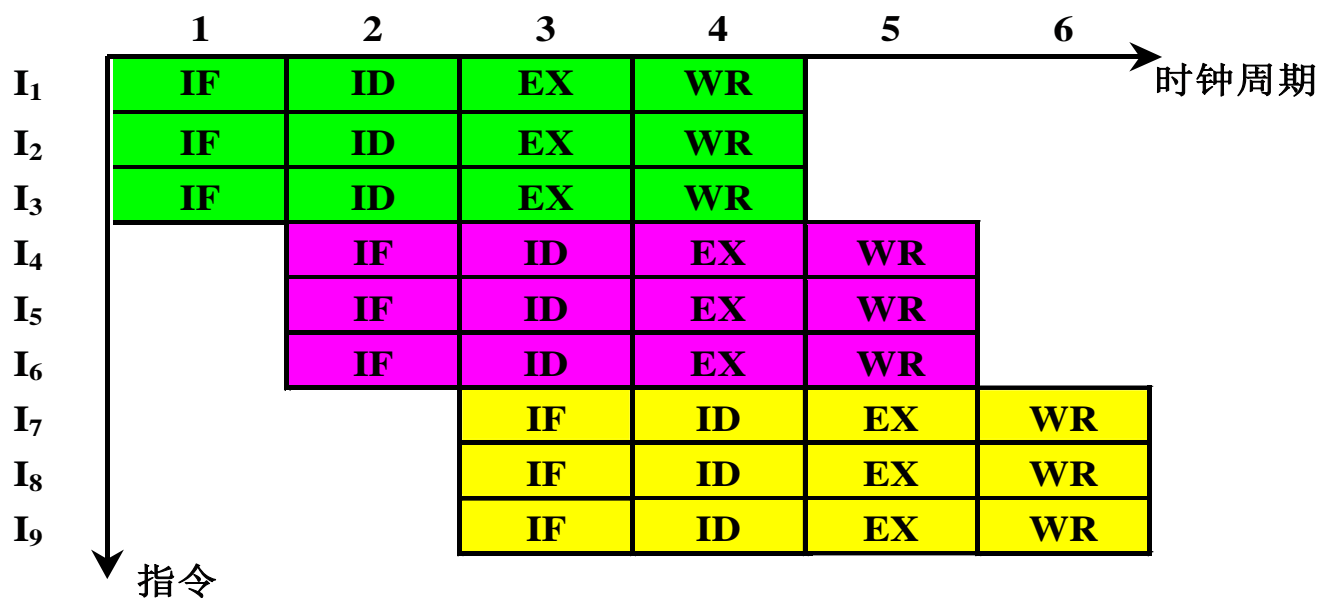
2. 多发射处理机:

- 每个周期同时取多条指令、同时译码多条指令，同时执行多条指令，同时写回多个运算结果。
- 多个取指令部件，多个指令译码部件和多个写结果部件。
- 设置多个指令执行部件，有些指令执行部件采用流水线结构。
- 目标是每个时钟周期平均执行多条指令，ILP的期望值大于1。

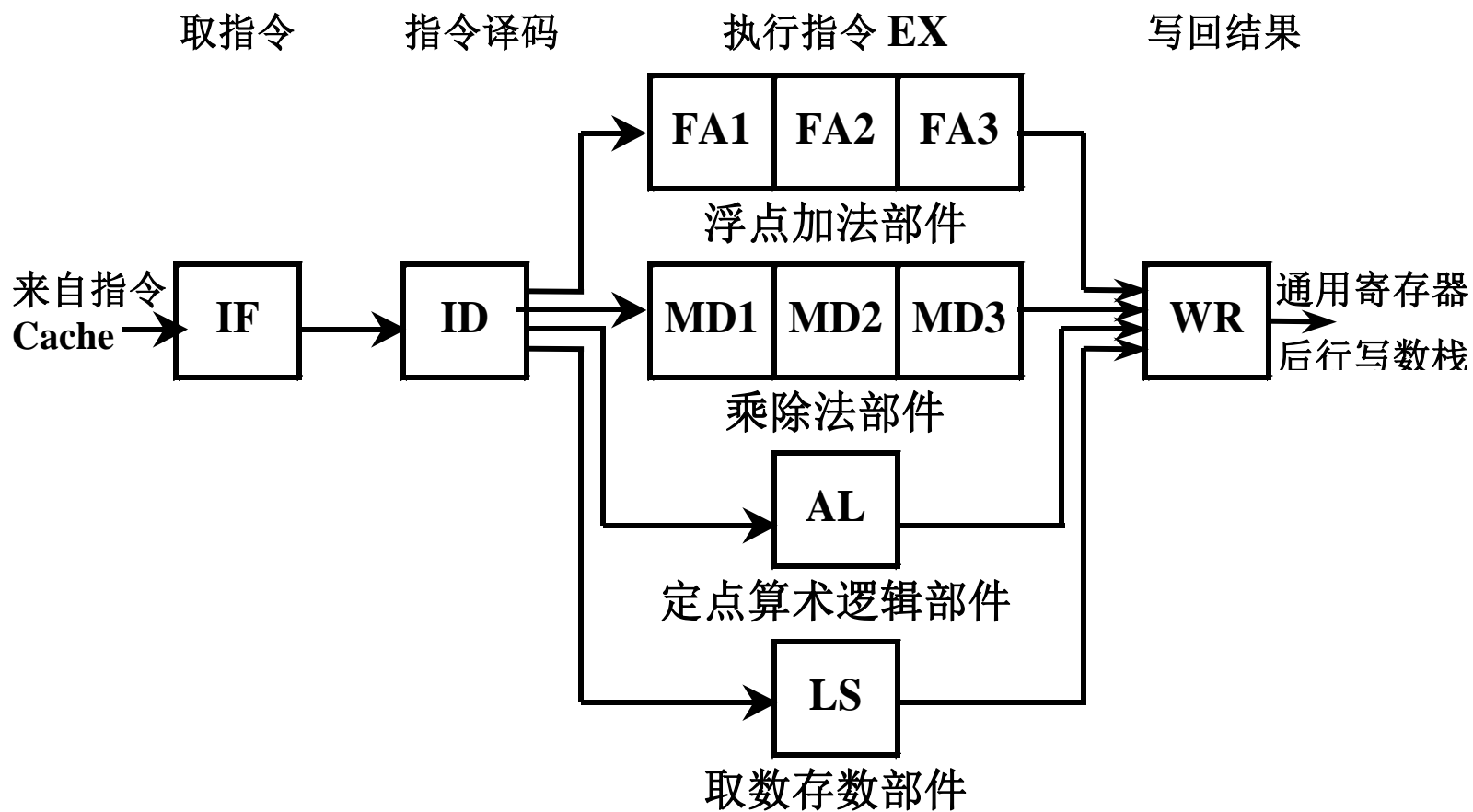
单发射处理机的指令流水线时空图



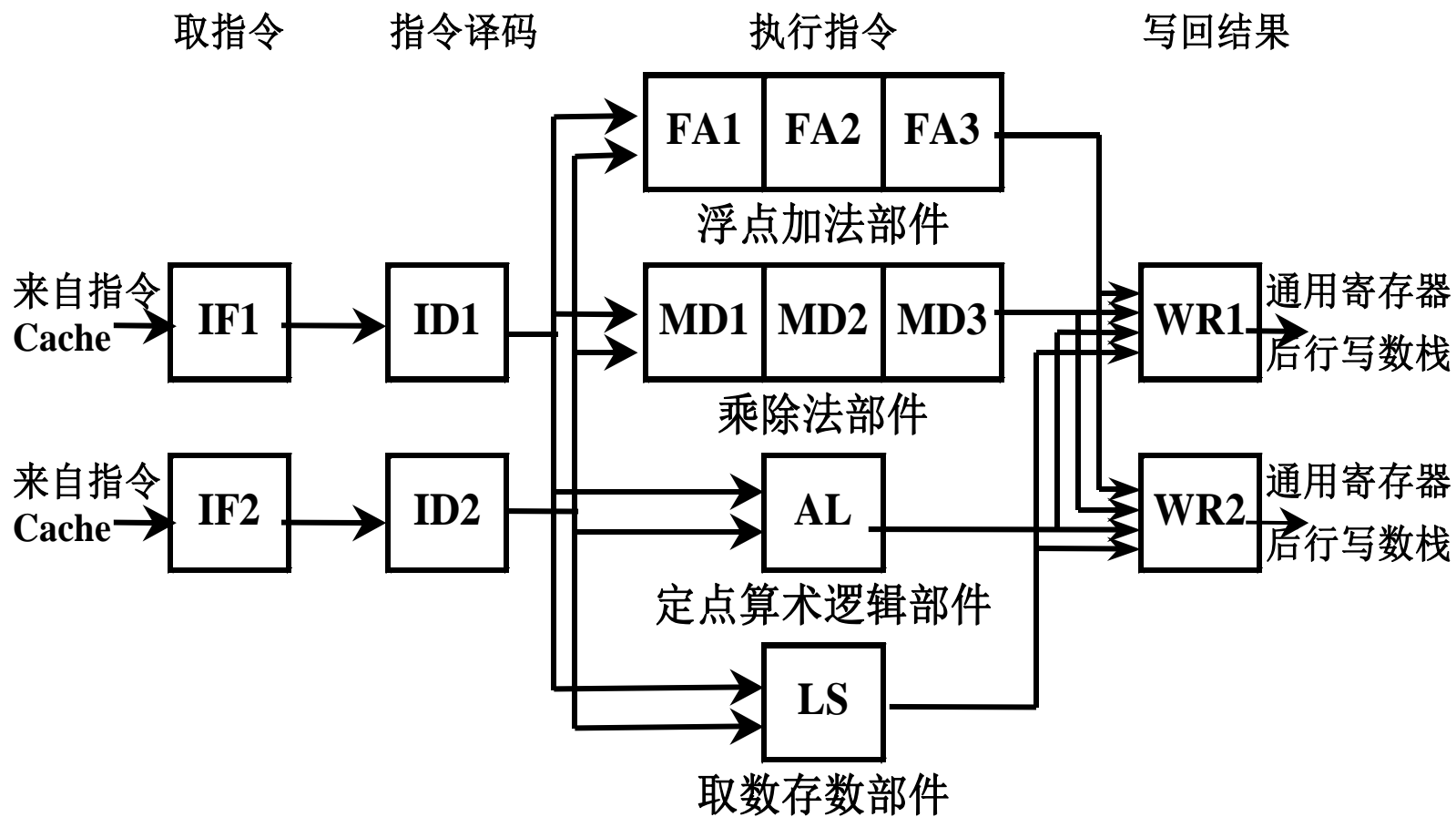
多发射处理机的指令流水线时空图



单发射处理机的指令流水线



同时发射两条指令的多发射处理机的指令流水线

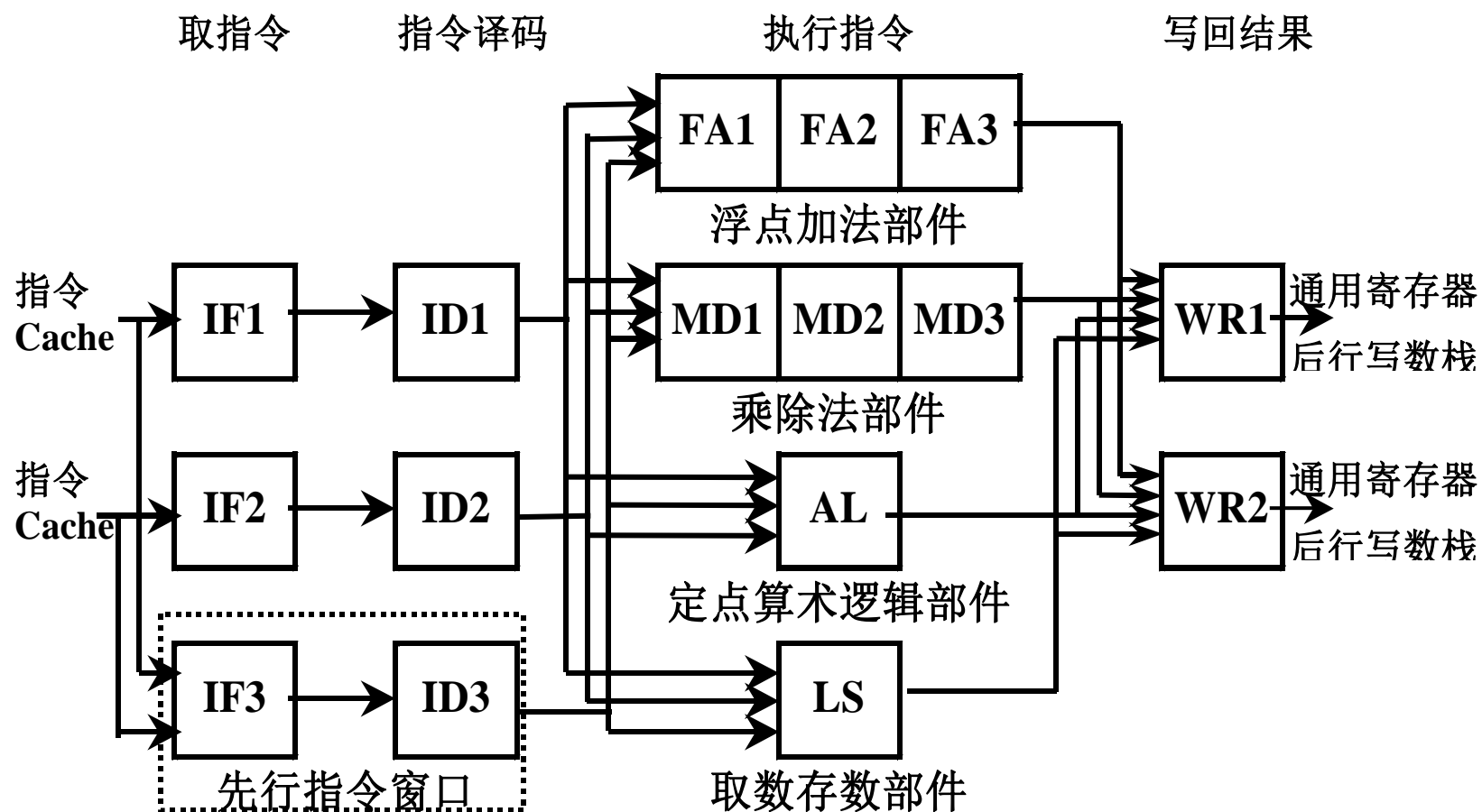


5.3.2 单发射与多发射

3. 超标量处理机：

- 有两条或两条以上能同时工作的指令流水线
- 先行指令窗口：能够从指令Cache中预取多条指令，能够对窗口内的指令进行数据相关性分析和功能部件冲突检测。
- 例如：Intel公司的i860、i960、Pentium，Motolora公司的MC88110，IBM公司的Power 6000, TI公司生产SuperSPARC等
- 操作部件的个数一般多于每个周期发射的指令条数。通常为4 个至16个操作部件。
- 超标量处理机的指令级并行度： $1 < \text{ILP} < m$

有先行指令窗口的超标量处理机的流水线结构



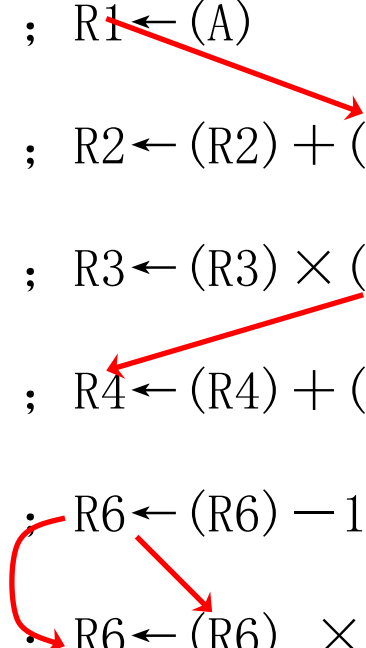
FA: 浮点加减法运算, MD: 乘除法运算, AL: 定点算术逻辑运算, LS 取数存数

5.3.3 多流水线调度

- 顺序发射(in-order issue)与乱序发射(out-order issue)：指令发射顺序是按照程序中指令排列顺序进行的称为顺序发射
- 顺序完成(in-order completion)与乱序完成(out-order completion)：指令完成顺序是按照程序中指令排列顺序进行的称为顺序完成
- 多流水线的调度主要有三种方法：
 - 顺序发射顺序完成
 - 顺序发射乱序完成
 - 乱序发射乱序完成

➤ 以如下6条指令组成的程序为例，说明这三种调度方法

I_1 : LOAD R1, A ; $R1 \leftarrow (A)$
 I_2 : FADD R2, R1 ; $R2 \leftarrow (R2) + (R1)$
 I_3 : FMUL R3, R4 ; $R3 \leftarrow (R3) \times (R4)$
 I_4 : FADD R4, R5 ; $R4 \leftarrow (R4) + (R5)$
 I_5 : DEC R6 ; $R6 \leftarrow (R6) - 1$
 I_6 : FMUL R6, R7 ; $R6 \leftarrow (R6) \times (R7)$



➤ 6条指令中有4个数据相关，包括2个写读相关，1个读写相关和1个写写相关。

1. 顺序发射顺序完成

共用10个时钟周期完成

还有8个空闲的时钟周期

I₁: LOAD R1, A ; R1←(A)

I₂: FADD R2, R1 ; R2←(R2)+(R1)

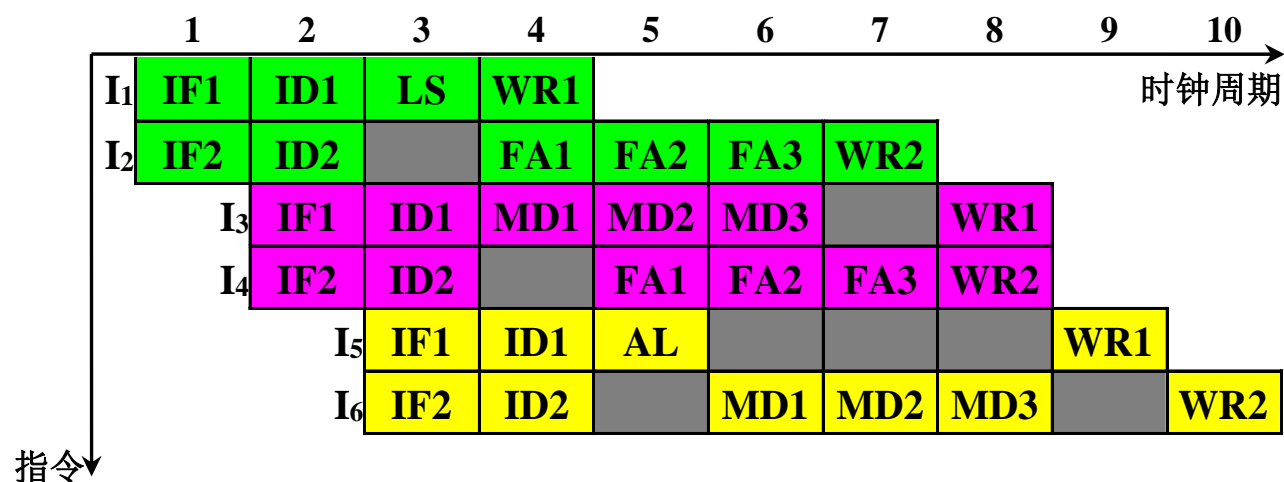
I₃: FMUL R3, R4 ; R3←(R3)×(R4)

I₄: FADD R4, R5 ; R4←(R4)+(R5)

I₅: DEC R6 ; R6←(R6)−1

I₆: FMUL R6, R7 ; R6←(R6)×(R7)

顺序发射顺序完成的指令流水线时空图



IF: 取指令, ID: 指令译码, LS 取数存数, FA: 浮点加减法运算,
MD: 乘除法运算, AL: 定点算术逻辑运算 WR: 写回运算结果

2. 顺序发射乱序完成

总的执行时间为9个时钟周期，

节省了一个时钟周期。少了5个空闲时钟周期

I_1 : LOAD R1, A ; $R1 \leftarrow (A)$

I_2 : FADD R2, R1 ; $R2 \leftarrow (R2) + (R1)$

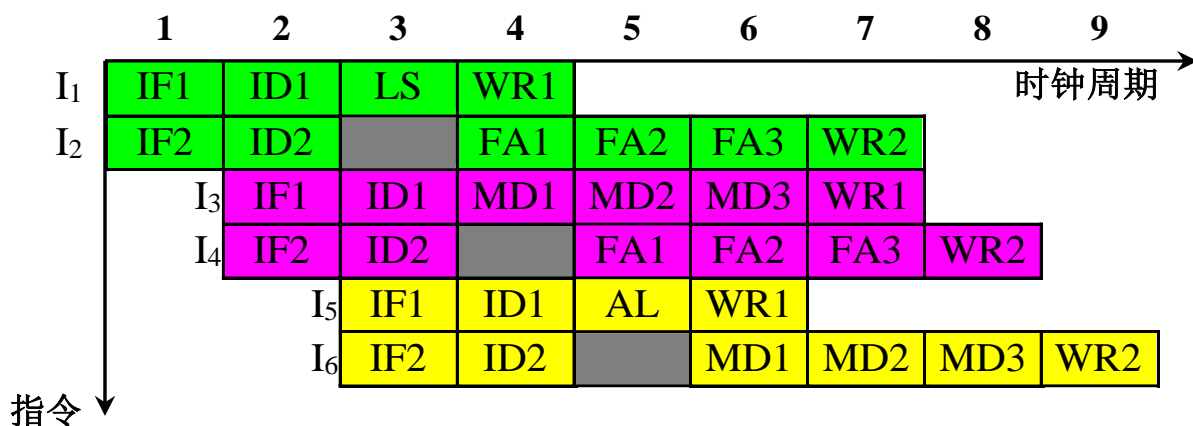
I_3 : FMUL R3, R4 ; $R3 \leftarrow (R3) \times (R4)$

I_4 : FADD R4, R5 ; $R4 \leftarrow (R4) + (R5)$

I_5 : DEC R6 ; $R6 \leftarrow (R6) - 1$

I_6 : FMUL R6, R7 ; $R6 \leftarrow (R6) \times (R7)$

顺序发射乱序完成的流水线时空图



顺序发射乱序完成的指令完成次序

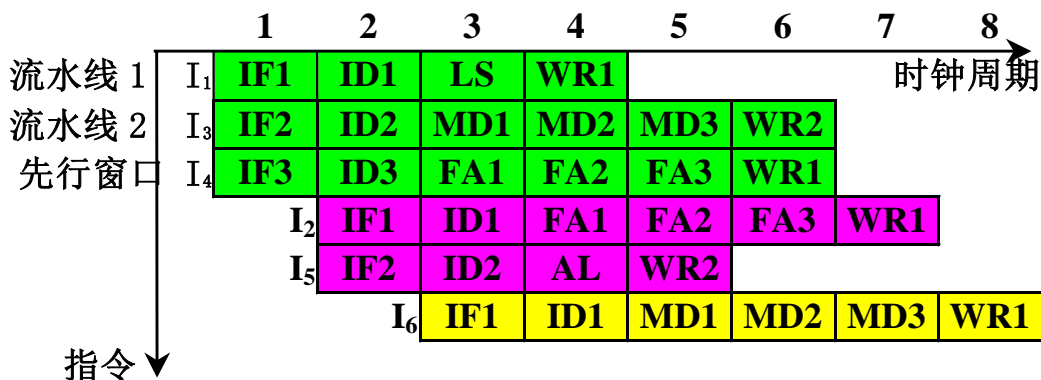
时钟周期	4	5	6	7	8	9
流水线 1	I_1		I_5	I_3		
流水线 2				I_2	I_4	I_6

3. 乱序发射乱序完成

- 没有空闲周期，功能部件得到充分利用
- 总的执行时间为8个周期，节省1个周期

I_1 : LOAD R1, A ; $R1 \leftarrow (A)$
 I_2 : FADD R2, R1 ; $R2 \leftarrow (R2) + (R1)$
 I_3 : FMUL R3, R4 ; $R3 \leftarrow (R3) \times (R4)$
 I_4 : FADD R4, R5 ; $R4 \leftarrow (R4) + (R5)$
 I_5 : DEC R6 ; $R6 \leftarrow (R6) - 1$
 I_6 : FMUL R6, R7 ; $R6 \leftarrow (R6) \times (R7)$

乱序发射乱序完成调度方法的流水线时空图



指令在流水线中的发射次序

时钟周	1	2	3
流水线 1	I_1	I_2	I_6
流水线 2	I_3	I_5	
先行窗口	I_4		

指令在流水线中的完成次序

时钟周期	4	5	6	7	8
流水线 1	I_1		I_4	I_2	I_6
流水线 2		I_5	I_3		

5.3.4 资源冲突

- 如果操作部件采用流水线结构，发生资源冲突的可能性很小；
- 如果不采用流水线结构，发生资源冲突的可能性就比较大。
- 下面是一个由4条指令的程序例子：

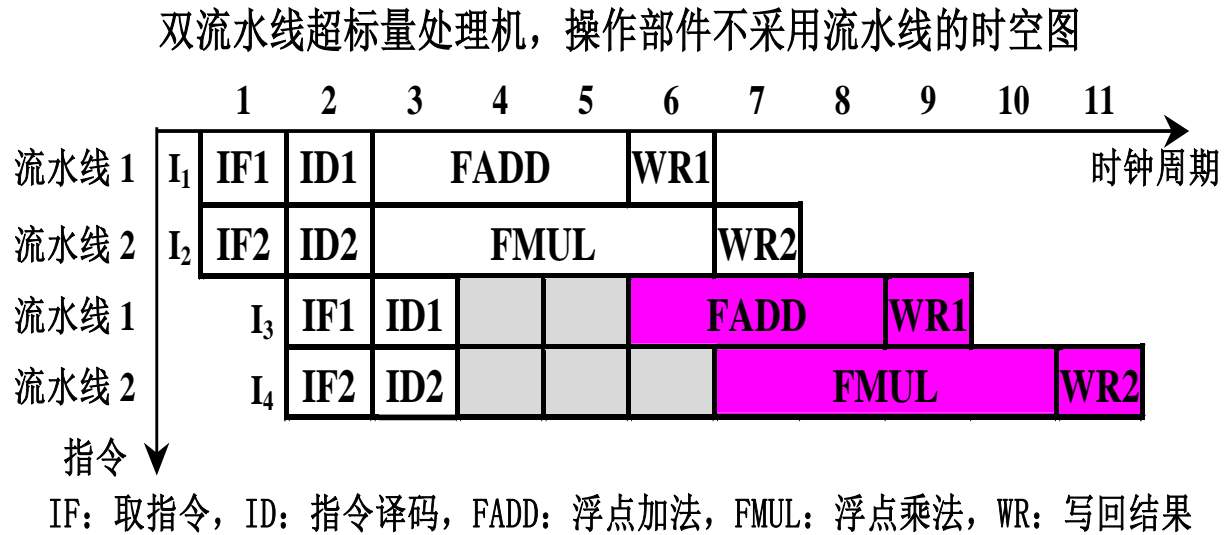
I1: FADD R0, R1 ; $R0 \leftarrow (R0) + (R1)$

I2: FMUL R2, R3 ; $R2 \leftarrow (R2) \times (R3)$

I3: FADD R4, R5 ; $R4 \leftarrow (R4) + (R5)$

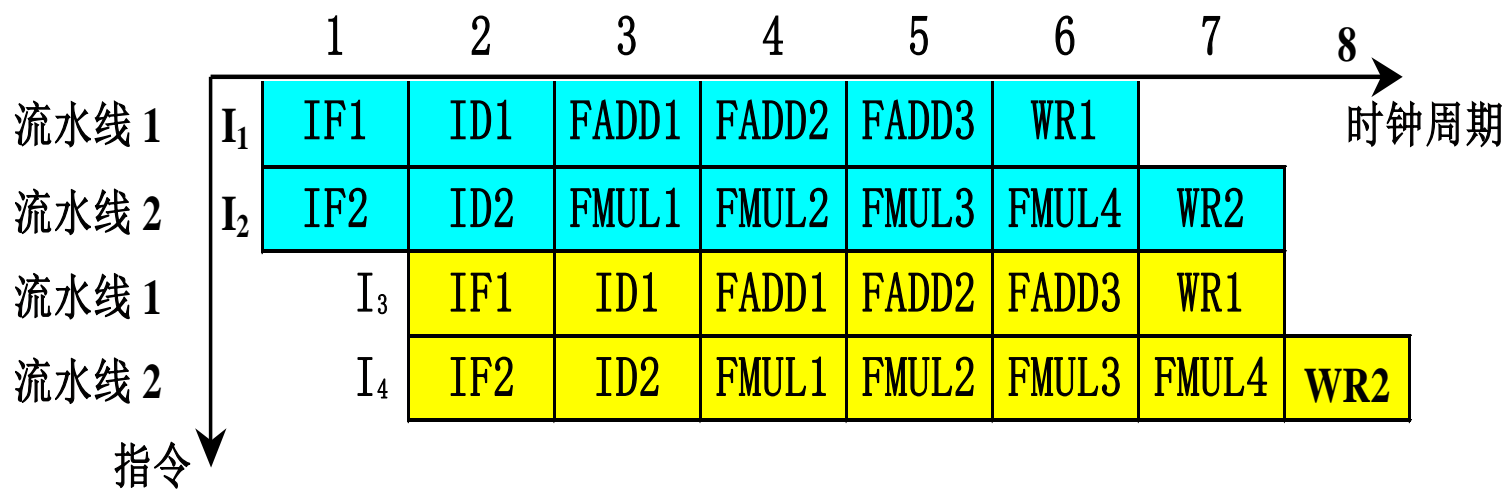
I4: FMUL R6, R7 ; $R6 \leftarrow (R6) \times (R7)$

- 操作部件不采用流水线：做完4条指令总共用了11个周期，有5个空闲周期。



- 操作部件采用流水线：做完4条指令共用8个周期，少用3个周期。

双流水线超标量处理机，操作部件采用流水线的时空图



IF: 取指令, ID: 指令译码, FADD: 浮点加法, FMUL: 浮点乘法, WR: 写回结果

操作部件采用流水线结构的原因分析

- 假设每个周期发射 m 条指令，操作部件的延迟时间为 k 个周期，
- 如果操作部件不采用流水线结构，则使用同一个操作部件的两条指令应该**至少相差 $m \times k$**
- 如果操作部件采用 k 段流水线结构，则使用同一个操作部件的两条指令**只需相差 m 或 m 以上**
- 指令流水线的段数 k 一般在4至10之间，每个时钟周期发射的指令条数 m 在2至4之间。取中间值， $k=7$ ， $m=3$

- 为了不发生资源冲突，如果操作部件不采用流水线结构，两条使用同一个功能部件的指令序号必须相差21或21以上。
- 如果操作部件采用流水线结构，两条使用同一个功能部件的指令序号只需要相差3或3以上。
- 因此，在超标量处理机中，操作部件一般要采用流水线结构。
- 如果由于某种原因，操作部件不能采用流水线结构，则必须设置多个相同种类的操作部件

- 普通标量处理机，希望相同操作连续出现。
- 只有连续出现相同操作的指令序列时，流水线的效率才能得到充分发挥。
- 超标量处理机则正好相反，希望相同操作不要连续出现。
- 相同操作的指令序列连续出现时，会发生资源冲突；要求相同操作的指令能够相对均匀地分布在程序中。
- 超标量处理机的这种要求正好符合一般标量程序的特点。

5.3.5 超标量处理机性能

- 单流水线普通标量处理机的指令级并行度记作(1, 1),
- 超标量处理机的指令级并行度记作(m, 1),
- 超流水线处理机的指令级并行度记作(1, n),
- 而超标量超流水线处理机的指令级并行度记作(m, n)。
- 在理想情况下, N条指令在单流水线标量处理机上的执行时间为:

$$T(1, 1) = (k + N - 1)\Delta t$$

- 在每个周期发射 m 条指令的超标量处理机上执行的时间为:

$$T(m, 1) = (k + \frac{N - m}{m})\Delta t$$

- 超标量处理机相对于单流水线标量处理机的加速比为:

$$S(m, 1) = \frac{T(1,1)}{T(m, 1)} = \frac{m(k + N - 1)}{N + m(k - 1)}$$

- 超标量处理机的加速比的最大值为

$$S(m, 1)_{max} = m$$

$$\begin{aligned} & \frac{(k + N - 1)\Delta t}{(k + \frac{N - m}{m})\Delta t} \\ &= \frac{k + N - 1}{k + \frac{N - m}{m}} \\ &= \frac{k + N - 1}{\frac{km + N - m}{m}} \end{aligned}$$

5.4 超流水线处理机

5.4.1 指令执行时序

5.4.2 典型处理机结构

5.4.3 超流水线处理机性能

➤ 超流水线处理机的两种定义：

- 在一个周期内**分时**发射多条指令的处理机
- 指令流水线的段数大于等于8的流水线处理机

➤ 提高处理机性能的两种方法：

- 通过增加硬件资源来提高处理机性能
- 通过各部分硬件的重叠工作来提高处理机性能

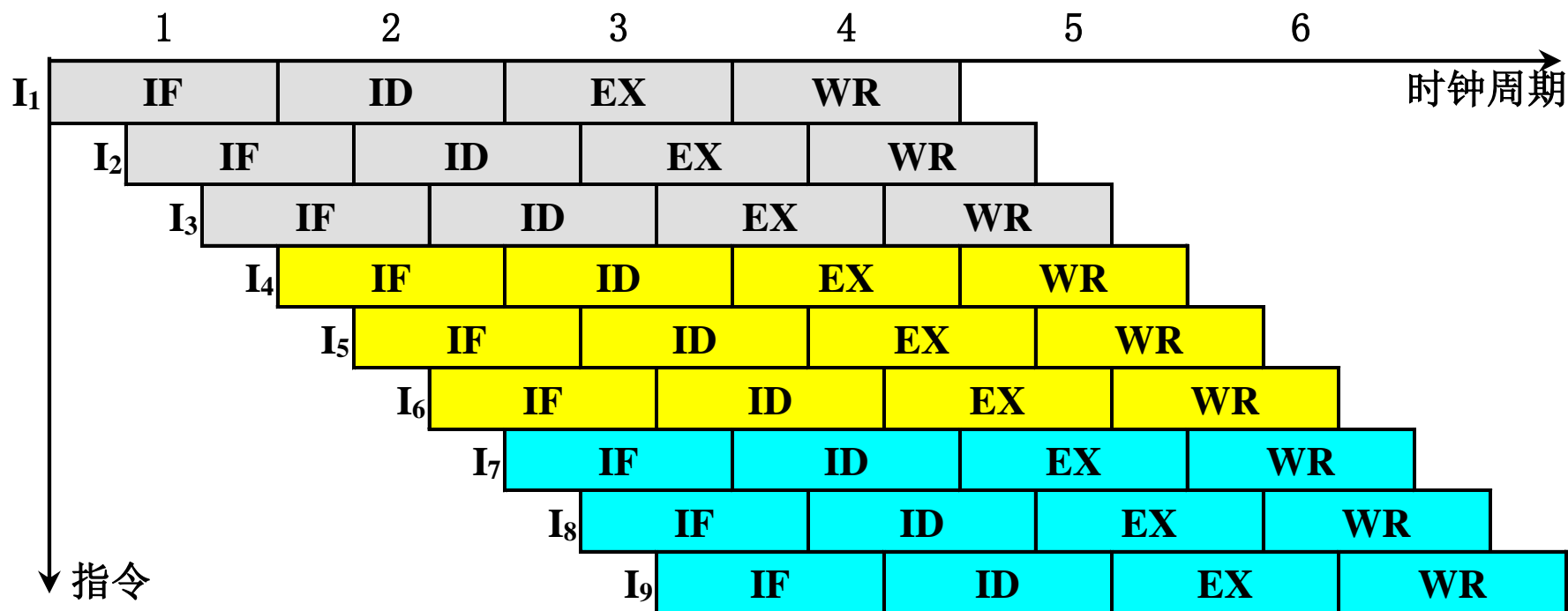
➤ 两种不同并行性：

- 超标量处理机采用的是空间并行性。
- 超流水线处理机采用的是时间并行性。

5.4.1 指令执行时序

- 每隔 $1/n$ 个时钟周期发射一条指令，即处理机的流水线周期为 $1/n$ 个时钟周期。

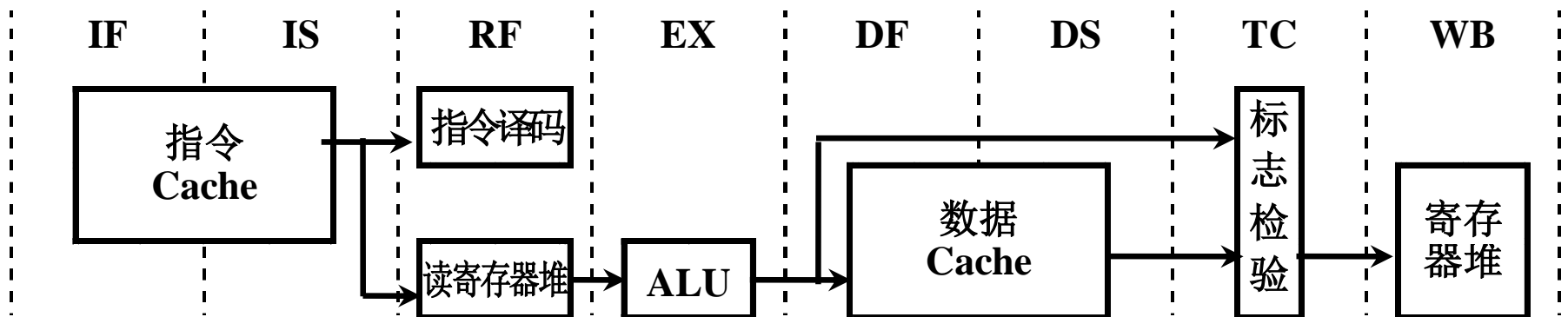
每个时钟周期分时发射 3 条指令的超流水线处理机的指令执行时空图



5.4.2 典型处理机结构

- MIPS R4000处理机：每个时钟周期包含两个流水段
- 是一种很标准的超流水线处理机结构。
- 读数据指令流水线有8个流水段。
- 指令Cache和数据Cache的容量各8KB，
- 每个时钟周期可以访问Cache两次，
- 在一个时钟周期内可以从指令Cache中读出指令前后部分，从数据Cache中读出或写入数据的前后部分。
- 主要运算部件有整数部件和浮点部件。

MIPS R4000 处理机的流水线操作



IF: 取指令前半部分; **IS:** 取指令后半部分; **RF:** 读寄存器堆, 指令译码;
EX: 执行指令; **DF:** 取数据前半部分; **DS:** 取数据后半部分; **TC:** 数据标志检验;
WB: 写回结果

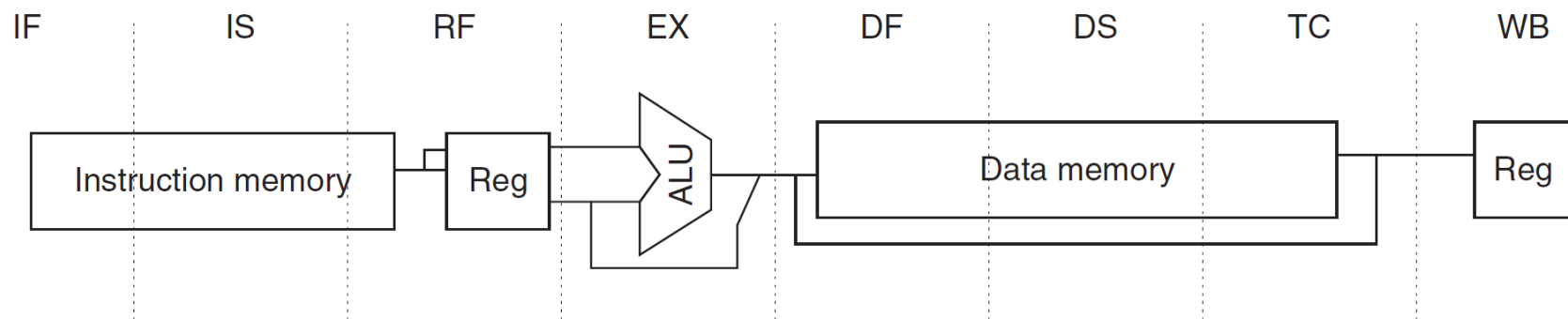
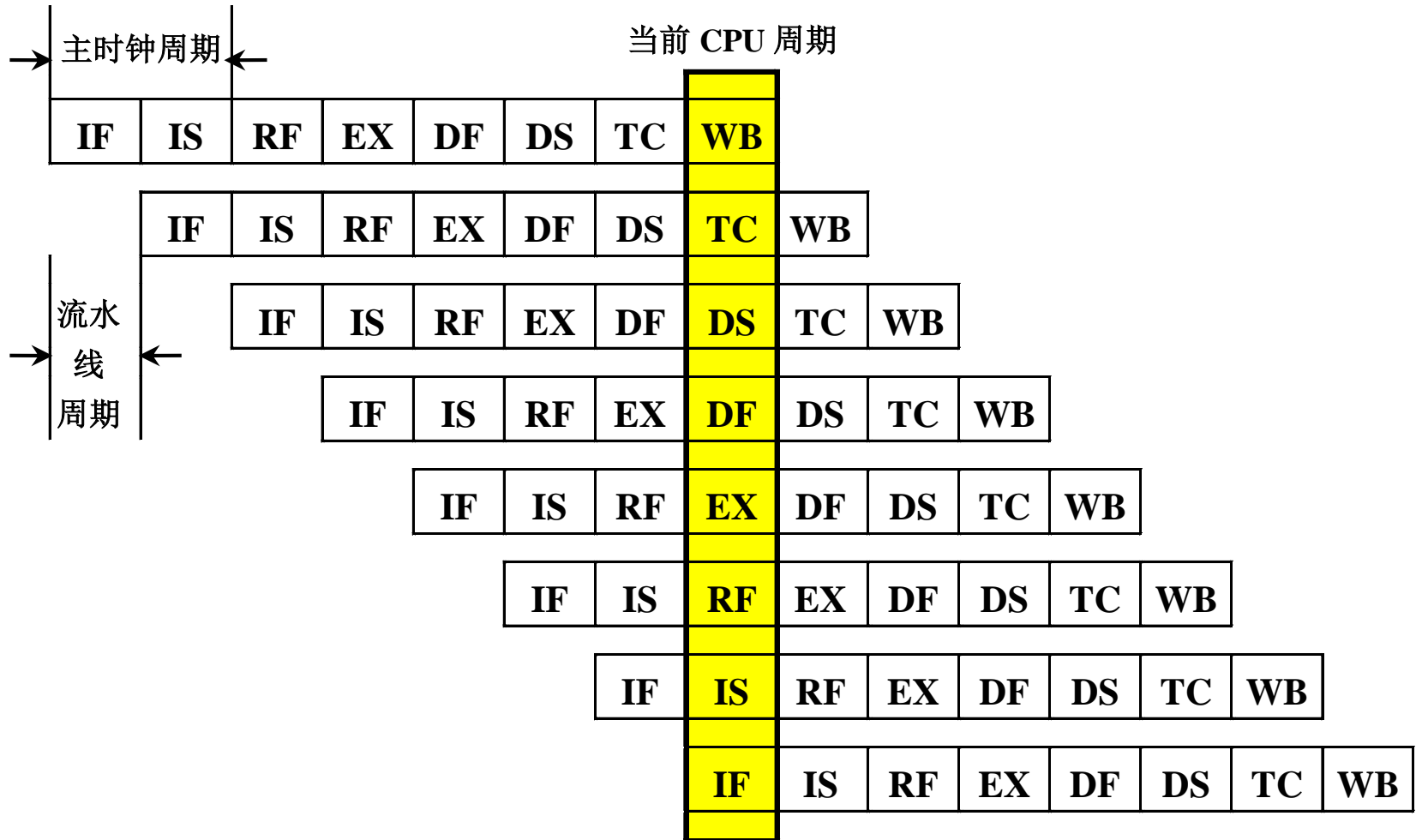


Figure C.36 The eight-stage pipeline structure of the R4000 uses pipelined instruction and data caches. The pipe stages are labeled and their detailed function is described in the text. The vertical dashed lines represent the stage boundaries as well as the location of pipeline latches. The instruction is actually available at the end of IS, but the tag check is done in RF, while the registers are fetched. Thus, we show the instruction memory as operating through RF. The TC stage is needed for data memory access, because we cannot write the data into the register until we know whether the cache access was a hit or not.

- IF—First half of instruction fetch; PC selection actually happens here, together with initiation of instruction cache access.
- IS—Second half of instruction fetch, complete instruction cache access.
- RF—Instruction decode and register fetch, hazard checking, and instruction cache hit detection.
- EX—Execution, which includes effective address calculation, ALU operation, and branch-target computation and condition evaluation.
- DF—Data fetch, first half of data cache access.
- DS—Second half of data fetch, completion of data cache access.
- TC—Tag check, to determine whether the data cache access hit.
- WB—Write-back for loads and register-register operations.

MIPS R4000 正常指令流水线工作时序



IF: 取指令前半部分; IS: 取指令后半部分; RF: 读寄存器堆, 指令译码;
 EX: 执行指令; DF: 取数据前半部分; DS: 取数据后半部分; TC: 数据标志检验;
 WB: 写回结果

- 如果在LOAD指令之后的两条指令中，任何一条指令要在它的EX流水级使用这个数据，则指令流水线要暂停一个时钟周期。

		指令	运行	暂停	暂停	运行	运行	运行	运行	运行	运行	运行
LOAD 指令 使用 LOAD 数据	I1	DF	DS	TC	WB							
	I2	EX	DF	DS	TC	WB						
	I3	RF			EX	DF	DS	TC	WB			
	I4	IS			RF	EX	DF	DS	TC	WB		
	I5	IF			IS	RF	EX	DF	DS	TC	WB	
	I6				IF	IS	RF	EX	DF	DS	TC	WB

5.4.3 超流水线处理机性能

- 指令级并行度为(1,n)的超流水线处理机，执行n条指令所的时间为：

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

- 超流水线处理机相对于单流水线普通标量处理机的加速比为：

$$S(1, n) = \frac{T(1, 1)}{T(1, n)} = \frac{(k + N - 1)\Delta t}{(k + \frac{N-1}{n})\Delta t} = \frac{n(k + N - 1)}{nk + N - 1}$$

- 加速比的最大值为： $S(1, n)_{max} = n$

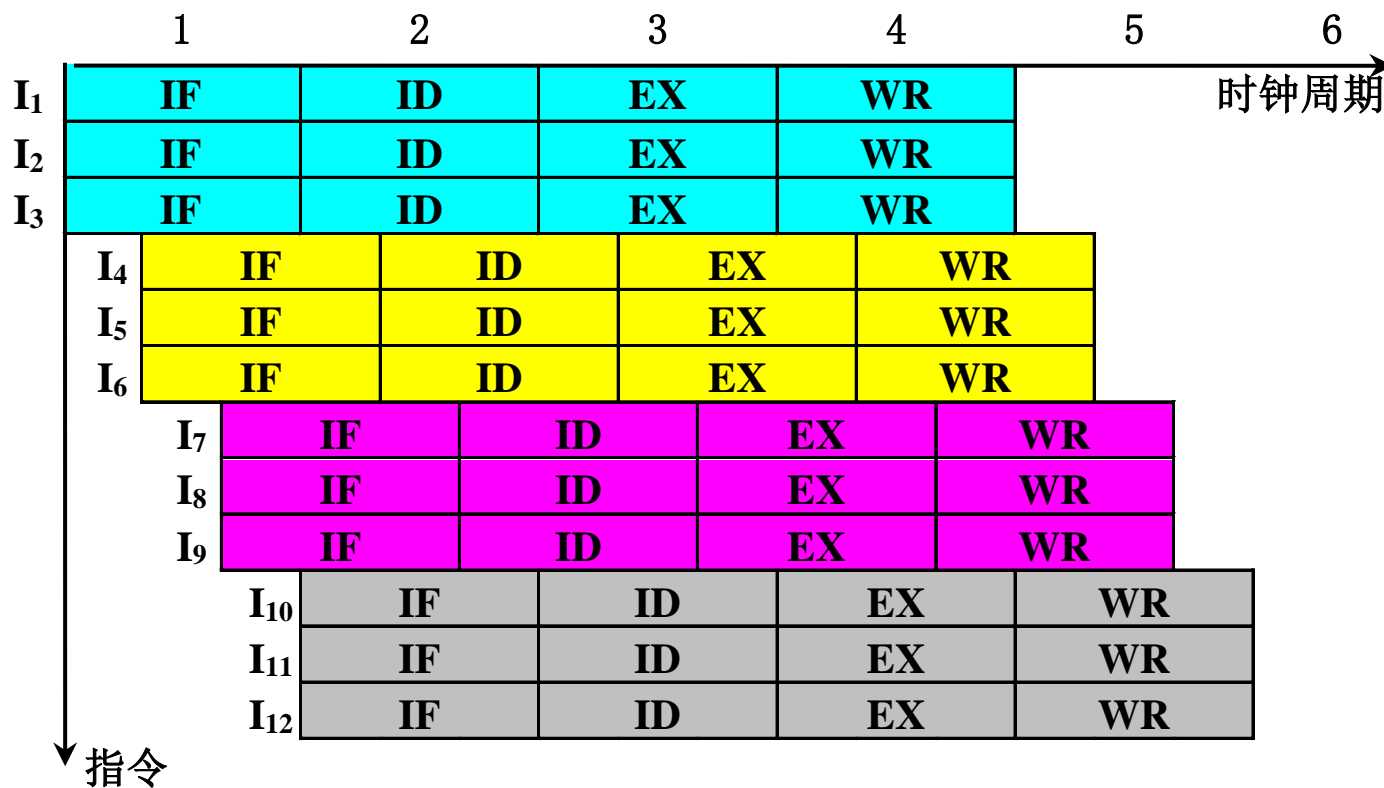
5.5 超标量超流水线处理机

一个时钟周期发射 m 次，每次发射 n 条指令

- 5.5.1 指令执行时序
- 5.5.2 典型处理机结
- 5.5.3 超标量超流水线处理机性能
- 5.5.4 三种处理机的性能比较

5.5.1 指令执行时序

每个时钟周期发射 3 次，每次同时发射 3 条指令的
超标量超流水线处理机的指令执行时空图



IF: 取指令, ID: 指令译码, EX: 执行指令, WR: 写回结果

5.5.2 典型处理机结构

- DEC公司的Alpha处理机为典型的**超标量超流水线结构**。
- 主要由四个功能部件和两个Cache组成：
 - 整数部件EBOX
 - 浮点部件FBOX
 - 地址部件ABOX
 - 中央控制部件IBOX
 - 指令Cache和数据Cache
- 在EBOX内还有多条专用数据通路，可以把运算结果直接送到执行部件。

- 中央控制部件IBOX能够同时完成：
 - 同时读出两条指令；
 - 同时对两条指令进行译码，并作相关性检测；
 - 如果资源和相关性允许，IBOX就把两条指令同时发射给EBOX、ABOX和FBOX三个执行部件中的两个。
- 指令流水线的控制方式：采用顺序发射乱序完成。
- 在指令Cache中有一个转移历史表，实现条件转移的动态预测。

- Alpha 21064处理机共有三条指令流水线：
 - 1) 整数操作流水线为7个流水段，其中，取指令2个流水段、分析指令2个流水段、运算2个流水段、写结果1个流水段。
 - 2) 访问存储器流水线为7个流水段。
 - 3) 浮点操作流水线分为10个流水段，其中，浮点执行部件FBOX的延迟时间为6个流水段。
- 三条指令流水线的平均段数为 $(7+7+10)/3=8$ ，且每个时钟周期发射两条指令。因此，Alpha 21064处理机为超标量超流水线处理机。

7 个流水段的整数操作流水线

(0)	(1)	(2)	(3)	(4)	(5)	(6)
IF	SWAP	I0	I1	A1	A2	WR

IF: 取指令; SWAP: 交换双发射指令, 转移预测; I0: 指令译码;
 I1 访问通用寄存器堆, 发射校验; A1: 计算周期 1, IBOX 计算新的 PC 值;
 A2: 计算周期 2, 查指令快表; WR: 写整数寄存器堆, Cache 命中检测。

7 个流水段的访问存储器流水线

(0)	(1)	(2)	(3)	(4)	(5)	(6)
IF	SWAP	I0	I1	AC	TB	HM

AC: ABOX 计算有效数据地址; TB: 查数据快表;
 HM: 写读数缓冲栈, Cache 命中/不命中检测。

10 个流水段的浮点操作流水线

(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
IF	SWAP	I0	I1	F1	F2	F3	F4	F5	FWR

F1—F5: 浮点计算流水线; FWR: 写回浮点寄存器堆。

5.5.3 超标量超流水线处理机的性能

- 指令级并行度为 (m, n) 的超标量超流水线处理机，连续执行 N 条指令所需要的时间为：

$$T(m, n) = (k + \frac{N - m}{m \cdot n}) \Delta t$$

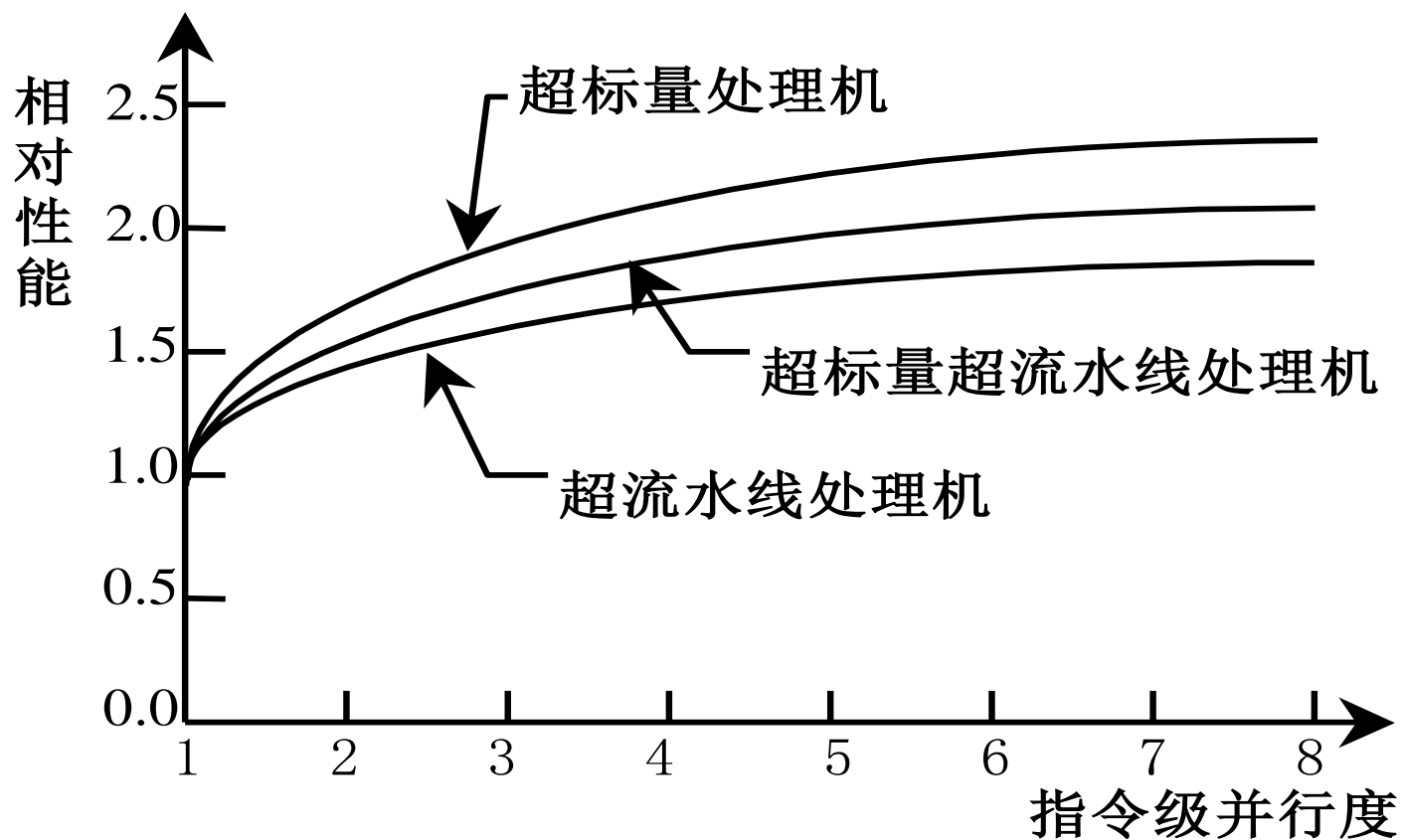
- 超标量超流水线处理机相对于单流水线标量处理机的加速比为：

$$S(m, n) = \frac{S(1, 1)}{S(m, n)} = \frac{(k + N - 1) \Delta t}{(k + \frac{N - m}{mn}) \Delta t} = \frac{mn(k + N - 1)}{mnk + N - m}$$

- 在理想情况下，超标量超流水线处理机加速比的最大值为：

$$S(m, n)_{max} = mn$$

5.7.4 三种标量处理机的性能比较



从三种标量处理机的性能曲线中，可以得出如下结论：

1. 三种处理机的性能关系

超标量处理机的相对性能最高，其次是超标量超流水线处理机，超流水线处理机的相对性能最低，主要原因如下：

- (1) 超标量处理机功能部件的冲突比超流水线处理机小。在指令执行过程中的许多功能段，超标量处理机都重复设置有多个相同的指令执行部件，而超流水线处理机只是把同一个指令执行部件分解为多个流水级。
- (2) 条件转移等操作造成的损失，超流水线处理机要比超标量处理机大。由于超流水线处理机采用深度流水线结构，对条件转移等操作比超标量处理机敏感。
- (3) 超流水线处理机的启动延迟通常要比超标量处理机大。超标量处理机在每个时钟周期的一开始就同时发射多条指令，超流水线处理机把一个时钟周期平均分成多个流水线周期，每个流水线周期只发射一条指令。

2. 实际指令级并行度与理论指令级并行度的关系

- 当横坐标给出的理论指令级并行度比较低时，处理机的实际指令级并行度的提高比较快。
- 当理论指令级并行度进一步增加时，处理机实际指令级并行度提高的速度越来越慢。
- 在实际设计超标量、超流水线、超标量超流水线处理机的指令级并行度时要适当，否则，有可能造成花费了大量的硬件，但实际上处理机所能达到的指令级并行度并不高。
- 目前，一般认为， m 和 n 都不要超过 4。

3. 最大指令级并行度

- 一个特定程序由于受到本身的数据相关和控制相关的限制，它的指令级并行度的最大值是有限的，是有个确定的值。
- 最大指令级并行度由程序自身的语义决定，与这个程序运行在那一种处理机上及采用何种方法开发并行性无关。
- 对于某一个特定的程序，图中的三条曲线最终都要收拢到同一个点上。
- 对于各个不同程序，这个收拢点的位置也是不同的。

本章重点：

1. 线性流水线的性能分析及计算
2. 非线性流水线的调度方法
3. 数据相关的种类，发生的情况及解决的办法
4. 分支预测技术
5. 乱序流动方式中的数据相关及解决办法
6. 单发射、多发射与先行指令窗口
7. 超标量、超流水线处理机的结构及性能分析