

# 第5章

# 流水线和向量处理机

郑宏 副教授  
计算机学院  
北京理工大学

# 学习内容

- **5.1 重叠方式**
- **5.2 流水方式**
- **5.3 向量的流水处理与向量处理机**
- **5.4 指令级高度并行的超级处理机**
- **5.5 ARM流水线处理器举例**

# 5.1 重叠方式

- 为加快机器语言程序的执行，要从系统整体的角度来考虑。可以从两个方面来考虑：
  - 提高每一条指令的执行速度
  - 提高多条指令的执行速度
- 为此，
  - **一方面**需要选用更高速的器件、采取更好的算法，提高指令内部各微操作的并行度
  - **另一方面**需要通过控制机构采用同时解释两条、多条甚至整段程序的控制方式

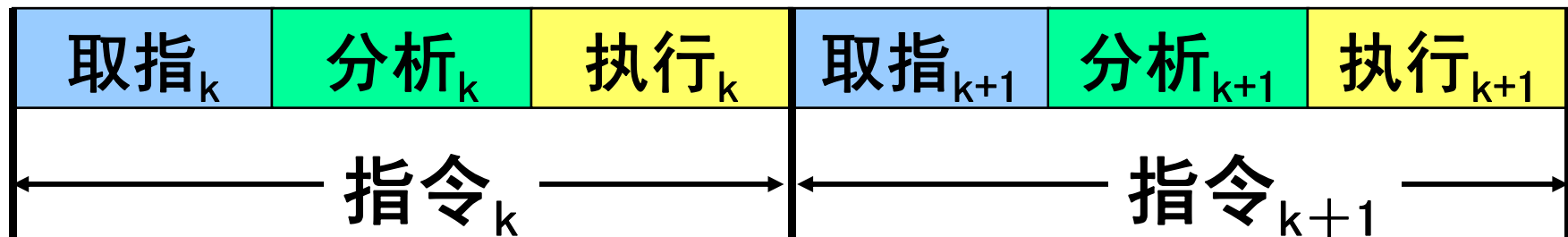
## 5.1.1 重叠原理和一次重叠

- 基于时间重叠技术
- 重叠解释执行两条或多条指令，加快程序的执行

# 顺序解释执行方式

## ■ 特点

- 指令之间串行执行，其指令内微操作也串行执行
- 如果把解释一条机器指令的微操作归结成**取指**、**分析**、**执行**三个步骤，那么，指令的顺序解释执行方式如图所示



# 顺序解释执行方式

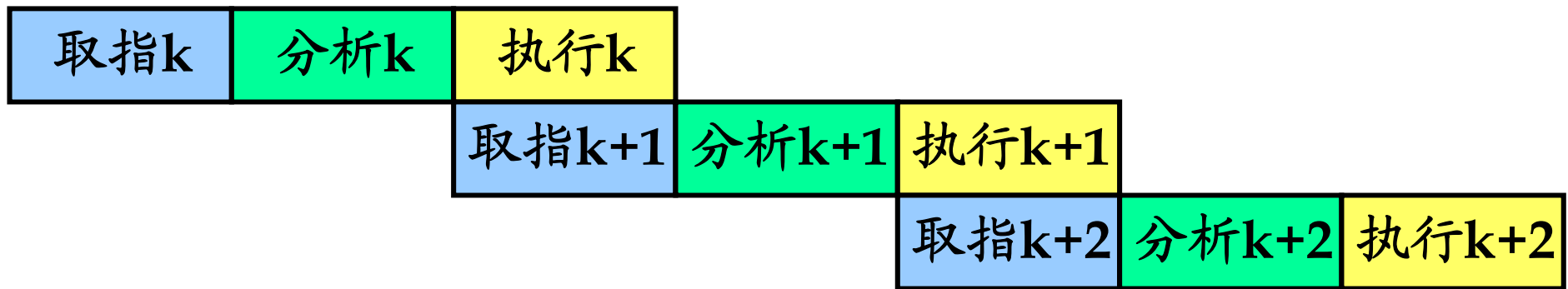
- 执行n条指令所用的时间为：

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

- **优点：**简单、易于实现
- **缺点：**速度难以提高，部件利用率低
- 为此提出了让不同机器指令的解释在时间上重叠进行的重叠解释方式

# 一次重叠执行方式

- 一种最简单的流水线方式
- 每次只重叠执行**两条**指令，故称为**一次重叠**
- **特点：**在第**K**条指令完成之前就开始处理第**K+1**条指令（重叠执行两条指令）



如果三个过程的时间相等，都为 $t$ ，则执行 $n$ 条指令的时间为： $T=(1+2n)t$

# 一次重叠执行方式

## ■ 优点:

- 虽不能加快一条指令的执行，但能加快相邻两条指令，以至一段程序的执行加快相邻指令或一段程序的执行。指令的执行时间缩短了近二分之一
- 提高了部件利用率

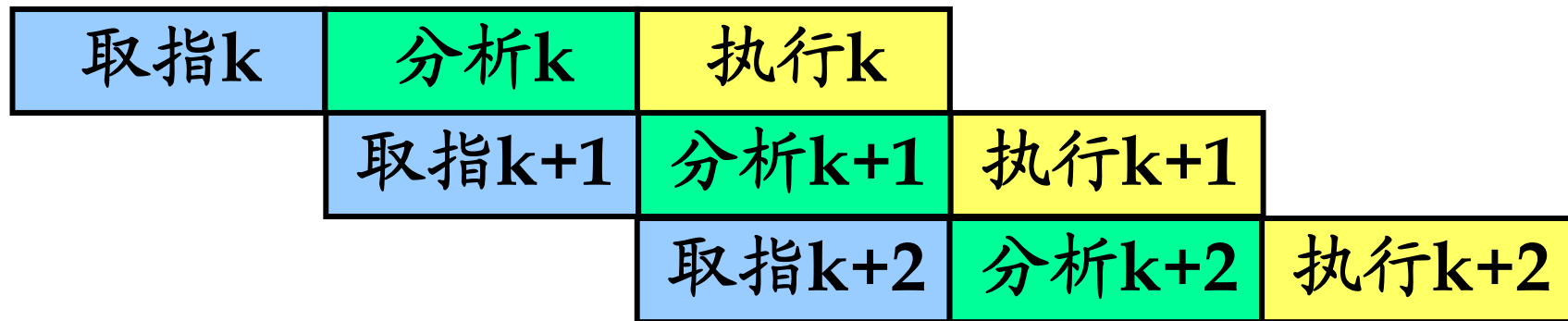
## ■ 缺点:

- 需要增加一些硬件
- 控制过程稍复杂



# 二次重叠执行方式

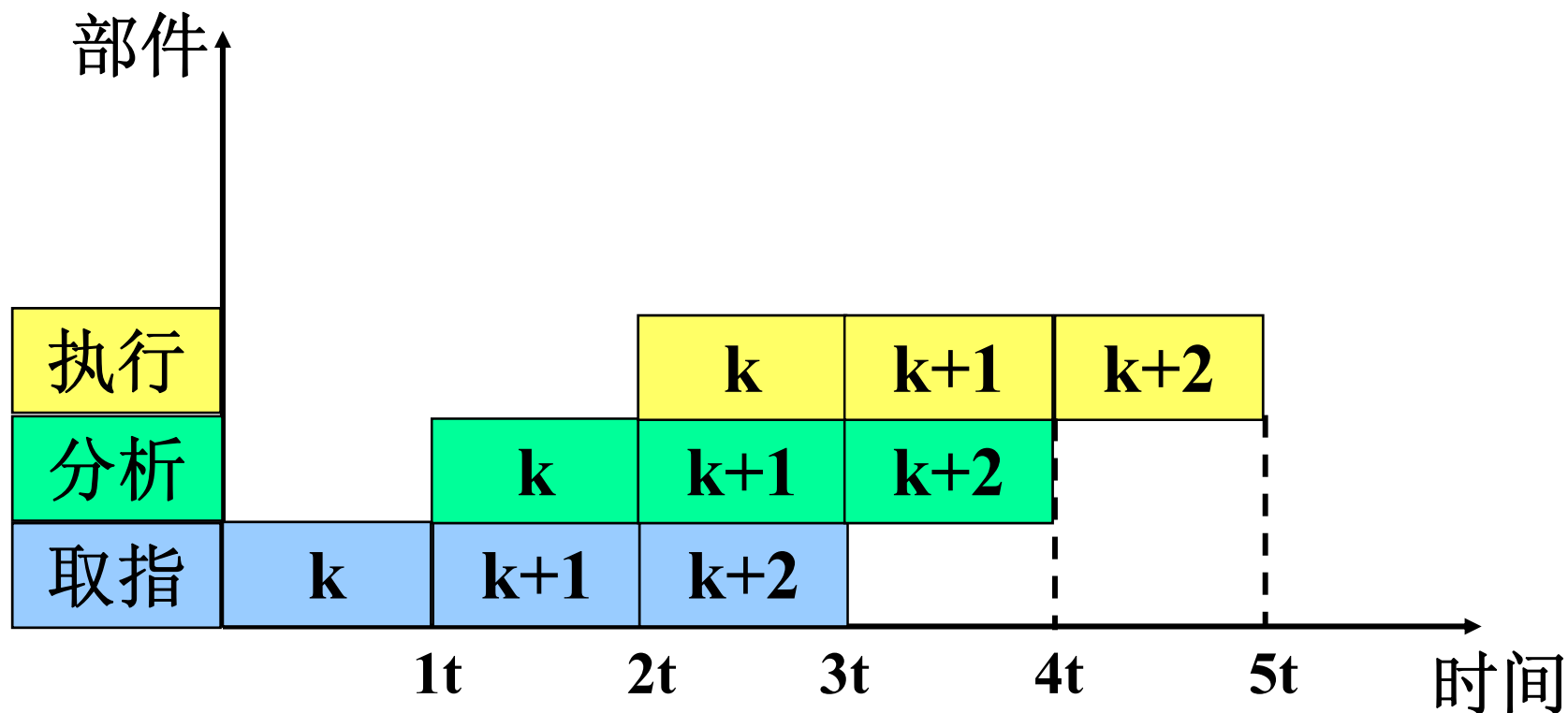
## ■ 每次重叠执行三条指令



## ■ 如果三过程的时间相等，执行n条指令的时间为： $T=(2+n)t$

采用二次重叠执行方式能够使指令的执行时间缩短近三分之二。

# 二次重叠执行方式



二次重叠时-空图

**二次重叠时，执行3条指令只需  $5t$  的时间**

# 重叠执行对计算机组成的要求

## ■ 采用重叠执行方式，须解决以下问题：

● 问题1：功能部件冲突

● 问题2：访存冲突

● 问题3：同步

● 问题4：转移（控制相关）

● 问题5：指令、数据相关 等（数据相关）

“结构相关” 或  
“资源相关”

# 重叠执行对计算机组成的要求

## ■ 问题1：功能部件冲突

- 为实现“执行<sub>k</sub>”与“分析<sub>k+1</sub>”的重叠，必须在硬件上保证有独立的取指、指令分析和指令执行部件。
- 解决方法：
  - ◆ 花费一定的硬件代价，分别设置独立的取指部件、指令分析部件、指令执行部件；
  - ◆ 三个子过程都有独立的控制器：存储控制器、指令控制器、运算控制器。

# 重叠执行对计算机组成的要求

## ■ 问题2：访存冲突

- “取指 $k+1$ ”与“分析 $k$ ”都要访问主存，但主存在同一时间只能访问一个存贮单元。
- 因此必须花费一定的代价，解决好主存访问冲突，否则无法实现“取指 $k+1$ ”与“分析 $k$ ”的重叠。

# 重叠执行对计算机组成的要求

## ■ 问题2：访存冲突（续）

### ● 解决方法：（4种）

- ◆ 1. 采用两个独立编址、可同时访问的的存贮器，分别存放操作数和指令。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，那么，取指令、分析指令和执行指令就可以同时进行。但增加了主存总线控制的复杂性和软件设计的麻烦。

# 重叠执行对计算机组成的要求

## ■ 问题2：访存冲突（续）

### ● 解决方法：（4种）

- ◆ 2. 维持操作数和指令的混存在主存中，但设置两个Cache：

指令Cache、数据Cache

在许多高性能处理机中，有独立的指令Cache和数据Cache。程序空间和数据空间相互独立的系统结构被称为哈佛结构。

# 重叠执行对计算机组成的要求

## ■ 问题2：访存冲突（续）

### ● 解决方法：（4种）

- ◆ 3. 维持操作数和指令的混存，采用多体交叉存贮器，采取低位交叉存取方式，让第 $k$ 条指令的操作数和第 $k+1$ 条指令存放在不同的存贮体。但这种方法有一定的局限性，不能从根本上解决冲突。
- ◆ 如果指令和数据放在同一个存储器，可使用双端口存储器，其中一个端口存取数据，另一个端口取指令。



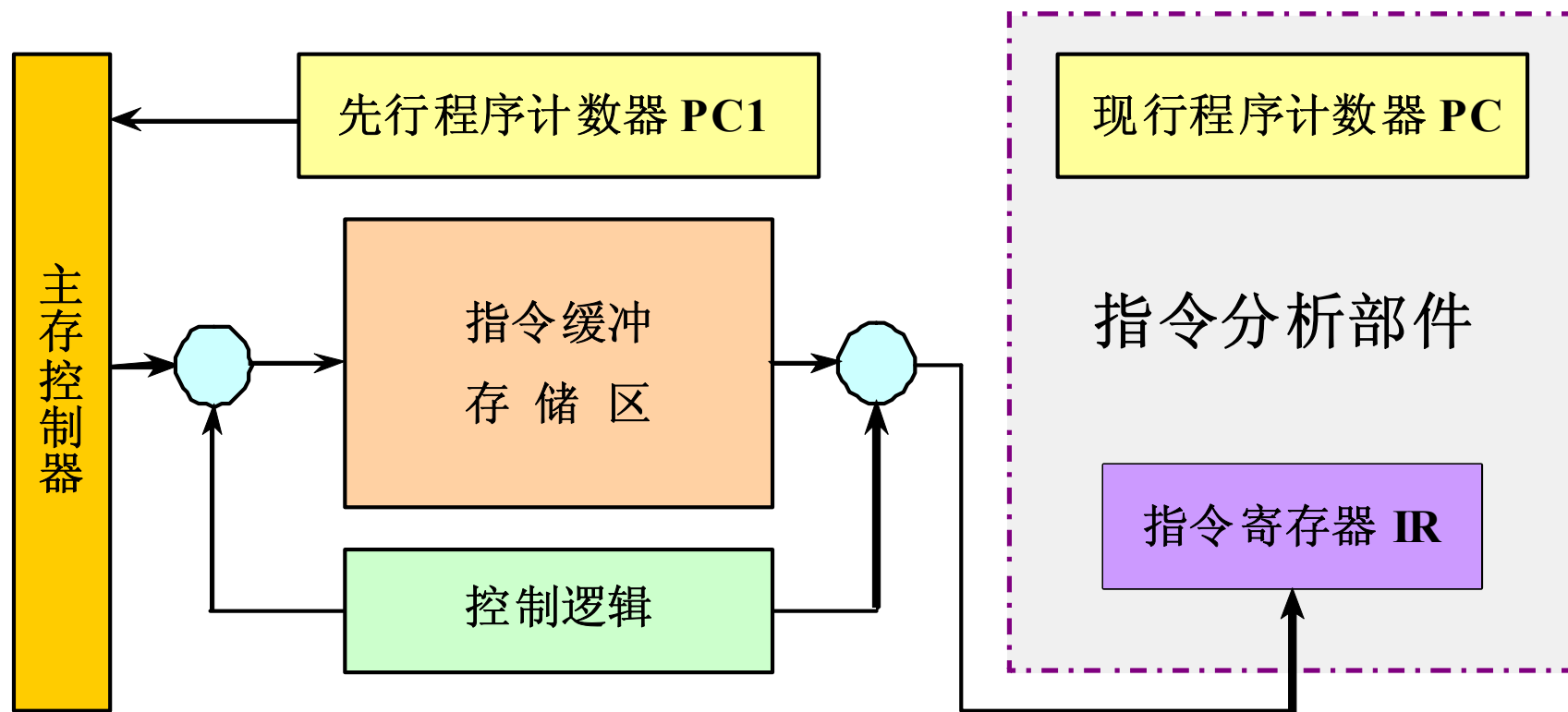
# 重叠执行对计算机组成的要求

## ■ 问题2：访存冲突（续）

### ● 解决方法：（4种）

- ◆ 4. 在主存和指令分析部件之间增设**FIFO**的**指令缓冲寄存器**（又被称为**先行指令缓冲站**）。利用主存的空闲，预先将下一条或下几条指令取入指缓，而不必访问主存储器。这样就能够使取指令、分析指令和执行指令重叠起来执行。

# 先行指令缓冲站



## 先行指令缓冲站的组成

# 先行指令缓冲站

## ■ 指令缓冲存储区和相应的控制逻辑

- 按队列方式工作。
- 只要指令缓冲站不满，它就自动地向主存控制器发取指令请求，不断地预取指令。

## ■ 指令分析部件

- 每分析完一条指令，就自动向指令缓冲站发出取下一条指令的请求。指令取出之后就把指令缓冲站中的该指令作废。
- 指令缓冲站中存放的指令的条数是动态变化的。

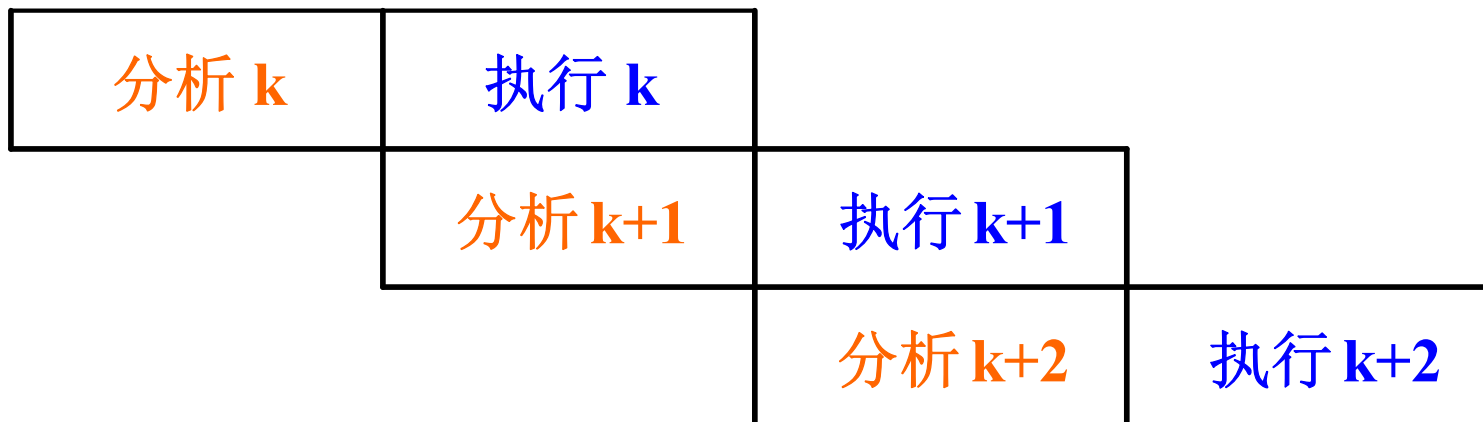
## ■ 两个程序计数器

- 先行程序计数器**PC1**：用于从主存预取指令；
- 现行程序计数器**PC**：用来记录指令分析部件当前正在分析的指令的地址。

# 先行控制方式中的一次重叠

- 若**取指令阶段**的时间很短，可以把这个操作合并到**分析指令**中。
- 上述的二次重叠就演变成了一次重叠
  - 把一条指令的执行过程分为**分析**和**执行**两个阶段；
  - 让前一条指令的**执行**与后一条指令的**分析**重叠进行。

# 先行控制方式中的一次重叠



如果指令分析和指令执行所需要的时间都是  $t$ ，则采用这种方式连续执行  $n$  条指令所需要的时间为：

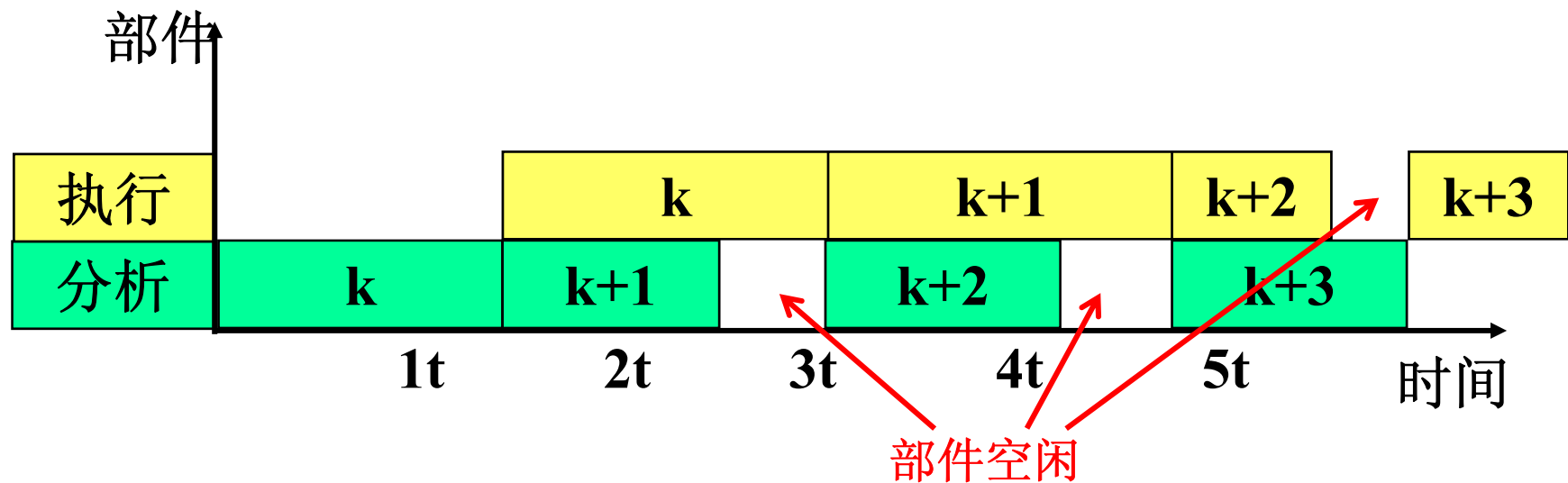
$$T = (1+n) t$$

控制方式比较简单，得到了广泛应用。

# 重叠执行对计算机组成的要求

## ■ 问题3：同步

- “执行”和“分析”所需要的时间常常不完全相同，指令分析部件和指令执行部件经常要相互等待，从而造成功能部件的浪费。



# 重叠执行对计算机组成的要求

先行控制技术最早在IBM公司研制的STRETCH 机器中采用。

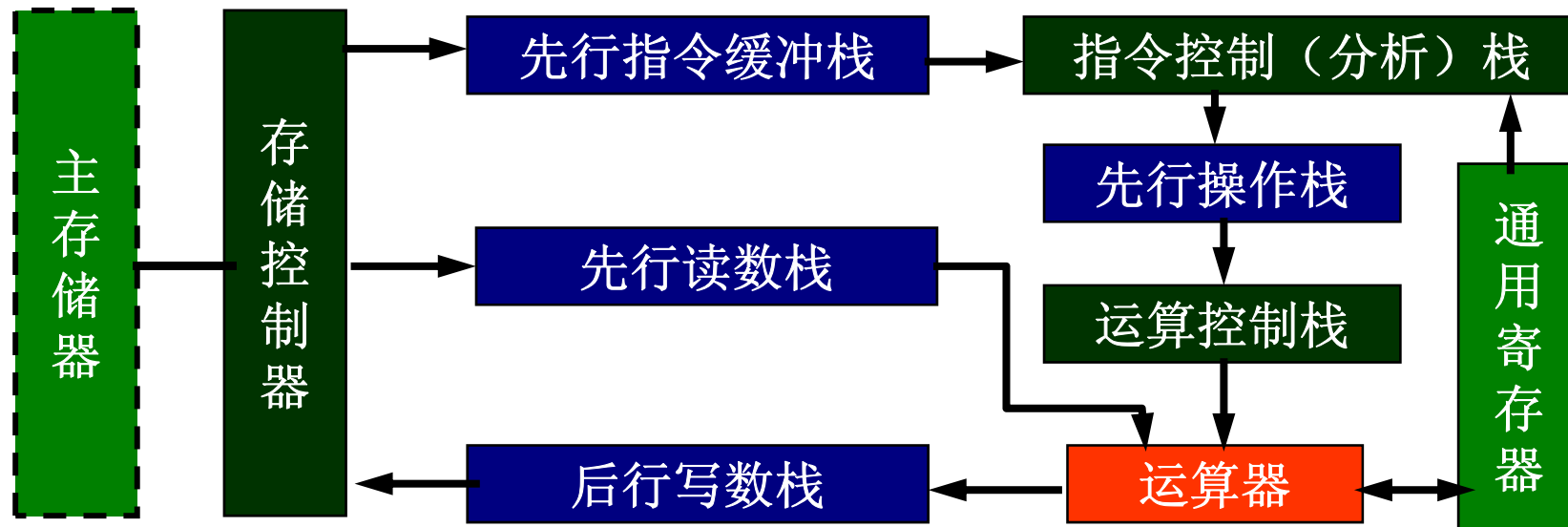
目前，许多处理机中都已经采用了这种技术。

## ■ 问题3：同步（续）

### ● 解决方法：采取先行控制技术

- ◆ 先行控制（**Look-ahead**）技术的**关键**是缓冲技术和预处理技术，以及这两者的相结合。
- ◆ **预处理技术**：把进入运算器的指令都处理成寄存器-寄存器型（**RR型**）指令，它与缓冲技术相结合，为进入运算器的指令准备好所需要的全部操作数；
- ◆ **缓冲技术**：在功能部件之间增设指令缓冲栈、先行读数栈、先行操作栈和后行写数栈等，以平滑它们的工作。

# 先行控制技术 = 缓冲技术 + 预处理技术



## 缓冲部件:

- 先行指令缓冲栈:
- 先行操作栈: 内存预处理过的RR\*型指令
- 先行读数栈: 包括先行地址缓冲寄存器、先行操作数缓冲寄存器和标志字段
- 后行写数栈: 包括一组后行地址缓冲寄存器、后行操作数缓冲寄存器和标志字段

## 预处理部件:

- 指令分析器: 将RS、RI、RX等类型指令变成RR\*型指令。

栈的深度（缓冲寄存器数量）要求:  $D_{\text{指缓}} \geq D_{\text{操作}} \geq D_{\text{读栈}} \geq D_{\text{写栈}}$

采用先行控制（**Look-ahead**，又称预测控制）技术的处理机



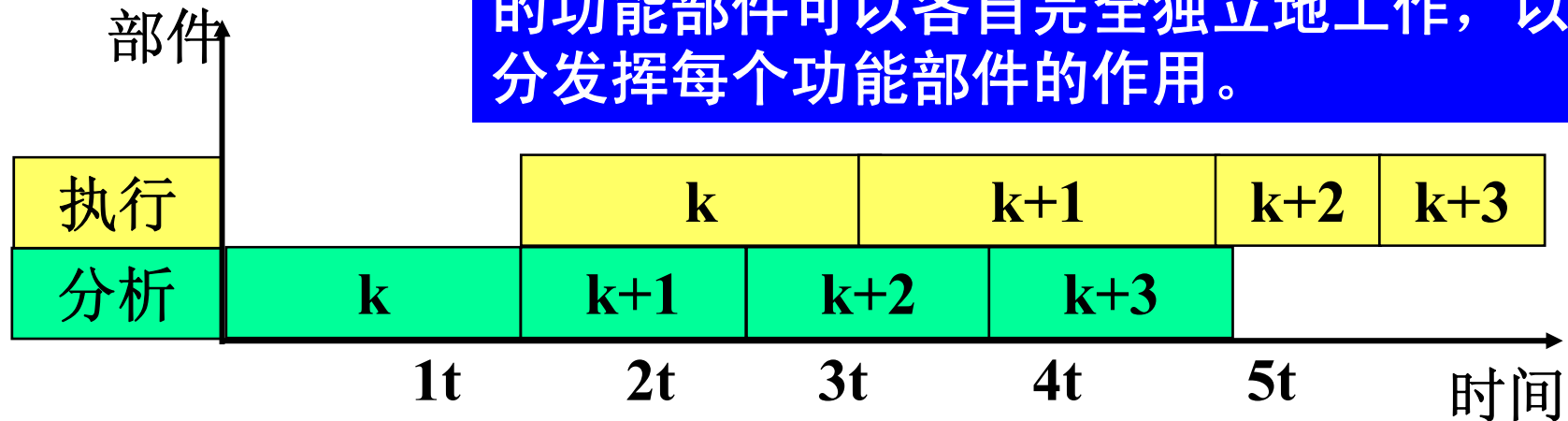
# 先行控制技术 = 缓冲技术 + 预处理技术

在设计先行控制器时，需要确定各个缓冲栈中的缓冲寄存器个数设置多少个，即所谓“**缓冲深度**”问题。

1. 如果缓冲寄存器的个数设置得太少，往往起不到缓冲的作用，指令分析器和指令执行部件不能连续地工作。
2. 相反，如果缓冲寄存器个数设置得太多，不仅浪费设备，而且控制逻辑也复杂。
3. 一般采用静态分析方法，再通过系统模拟来确定各个缓冲栈的缓冲深度。

# 采取先行控制技术

通过设置缓冲栈，两个工作时间长度不相等的功能部件可以各自完全独立地工作，以充分发挥每个功能部件的作用。



采取先行控制技术后可以连续工作

$$T_{\text{重}} = t_{\text{分}1} + \sum_{i=2}^n \max(t_{\text{分}i}, t_{\text{执}i-1}) + t_{\text{执}n}$$
$$T_{\text{先}} = t_{\text{分}1} + \sum_{i=1}^n t_{\text{执}i}$$

# 采取先行控制技术

- **结果：**解决了分析与执行时间不等长问题。
- **与重叠区别：**分析和执行部件可同时处理两条不相邻指令。
- **采用技术：**缓冲技术 + 预处理技术
- **硬件要求**
  - 增设指令缓冲栈，消除取指过程；
  - 增设数据缓冲栈，保证不同指令的读、写操作并行；
  - 增设先行操作栈，保证执行部件能连续执行。

# 采取先行控制技术

- **优点：** 节省硬件，只需设置一套指令分析部件和指令执行部件，同时有助于简化控制；
- **缺点：** 必须注意安排好微操作，使“执行”和“分析”的时间尽可能相等，从而使重叠效率较高。

# 例题

假设指令的解释分取指、分析和执行3步，每步的时间相应为  $t_{\text{取指}}$ 、 $t_{\text{分析}}$ 、 $t_{\text{执行}}$ ，分别计算下列几种情况下执行完100条指令所需时间的一般关系式：

1) 顺序方式

2) 仅“执行<sub>k</sub>”与“取指<sub>k+1</sub>”重叠

3) 仅“执行<sub>k</sub>”、“分析<sub>k+1</sub>”、“取指<sub>k+2</sub>”重叠

4) 分别  $t_{\text{取指}} = t_{\text{分析}} = 2$ ， $t_{\text{执行}} = 1$  及  $t_{\text{取指}} = t_{\text{分析}} = 5$ ， $t_{\text{执行}} = 2$  两种情况下，计算出上述各结果。

# 例题解答

执行完**100**条指令所需的时间:

## 1)顺序方式

$$100 \times (t_{\text{取指}} + t_{\text{分析}} + t_{\text{执行}})$$

## 2)仅“执行<sub>k</sub>”与“取指<sub>k+1</sub>”重叠

$$t_{\text{取指}} + 100t_{\text{分析}} + 99 \times \max\{t_{\text{取指}}, t_{\text{执行}}\} + t_{\text{执行}}$$

## 3)仅“执行<sub>k</sub>”、“分析<sub>k+1</sub>”、“取指<sub>k+2</sub>”重叠

$$t_{\text{取指}} + \max\{t_{\text{取指}}, t_{\text{分析}}\} + 98 \times \max\{t_{\text{取指}}, t_{\text{分析}}, t_{\text{执行}}\} \\ + \max\{t_{\text{分析}}, t_{\text{执行}}\} + t_{\text{执行}}$$

# 例题解答

当 $t_{\text{取指}}=t_{\text{分析}}=2$ ， $t_{\text{执行}}=1$ 时，代入上式，可求得  
**100条指令执行所要的时间：**

1)顺序方式： $100 \times (t_{\text{取指}} + t_{\text{分析}} + t_{\text{执行}}) = 500$

2)仅“执行 $k$ ”与“取指 $k+1$ ”重叠

$$t_{\text{取指}} + 100t_{\text{分析}} + 99 \times \max\{t_{\text{取指}}, t_{\text{分析}}\} + t_{\text{执行}} = 401$$

3)仅“执行 $k$ ”、“分析 $k+1$ ”、“取指 $k+2$ ”重叠

$$t_{\text{取指}} + \max\{t_{\text{取指}}, t_{\text{分析}}\} + 98 \times \max\{t_{\text{取指}}, t_{\text{分析}}, t_{\text{执行}}\} + \max\{t_{\text{分析}}, t_{\text{执行}}\} + t_{\text{执行}} = 203$$

# 例题解答

当 $t_{\text{取指}}=t_{\text{分析}}=5$ ， $t_{\text{执行}}=2$ 时，代入上式，可求得  
**100条指令执行所要的时间：**

1)顺序方式： $100 \times (t_{\text{取指}} + t_{\text{分析}} + t_{\text{执行}}) = 1200$

2)仅“执行 $k$ ”与“取指 $k+1$ ”重叠

$$t_{\text{取指}} + 100t_{\text{分析}} + 99 \times \max\{t_{\text{取指}}, t_{\text{分析}}\} + t_{\text{执行}} = 705$$

3)仅“执行 $k$ ”、“分析 $k+1$ ”、“取指 $k+2$ ”重叠

$$t_{\text{取指}} + \max\{t_{\text{取指}}, t_{\text{分析}}\} + 98 \times \max\{t_{\text{取指}}, t_{\text{分析}}, t_{\text{执行}}\} + \max\{t_{\text{分析}}, t_{\text{执行}}\} + t_{\text{执行}} = 510$$



# 重叠执行对计算机组成的要求

## ■ 问题3：同步（续）

- 多次重叠需要多套指令分析部件和指令执行部件，而且控制复杂，所以重叠方式的机器大多数都采用一次重叠。
- 若仍达不到速度要求，可以采用流水方式。

# 重叠执行对计算机组成的要求

## ■ 采用重叠执行方式，须解决以下问题：

● 问题1：功能部件冲突

● 问题2：访存冲突

● 问题3：同步

● 问题4：转移（控制相关）

● 问题5：指令、数据相关等（数据相关）

“结构相关” 或  
“资源相关”

## 5.1.2 相关处理

- **相关：** 邻近指令之间出现了某种关联，为了避免出错而使得它们不能被同时解释执行的现象

相关	类型	种类	解决方法
	数据相关 (局部相关)	指令相关、操作数相关（主存数据相关、寄存器数据相关）、变址相关	尽可能避免发生数据相关：指令推后分析、设定专用通路
	控制相关 (全局相关)	无条件转移相关、一般条件转移相关、符合条件转移相关、中断、程序调用	增设无条件转移指令分析器、采用转移预测技术、采用短循环程序处理技术

# 指令相关与处理

- **指令相关**：指第 $k+1$ 条指令是经第 $k$ 条指令的执行而形成的。为了避免出错，这两条指令就不能被同时解释。

**K:    STORE   R1, K+1    ; (R1)→K+1**  
**K+1:.....**

**指令 (K+1) = 结果 (K)**

**即：第K+1条指令内容为第K条指令的执行结果。这样，在一次重叠中，第K+1条指令分析必须等第K条指令执行完毕才能开始，否则，取出的第K+1条指令将是错误的！**

# 指令相关与处理

- 在先行控制方式下，若指令缓冲器可以存放多条指令，那么第  $k$  条指令可能会与已经预取进来的多条指令发生相关。
- 若发生相关，已经预取的指令**作废**，需要重新取指，更换指令缓冲器的内容。
- 指令相关的原因：
  - Von Neumann型计算机的指令允许修改

# 指令相关与处理

## 解决方法：

### ■ ①不允许修改指令

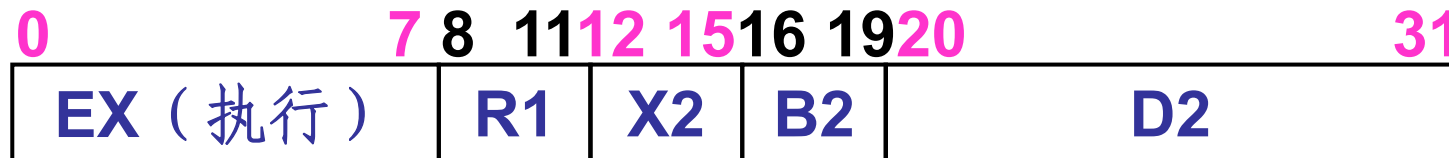
- 优点： 可以实现程序的可再入和递归调用
- 缺点： 程序设计的灵活性差

### ■ ②允许修改指令

- 改变指令的执行方式， 将指令相关转换为数  
相关， 统一按数据相关处理。
- 例如， **IBM 370**设置了“执行”指令。

# 指令相关与处理

## ■ IBM 370 “执行”指令形式：

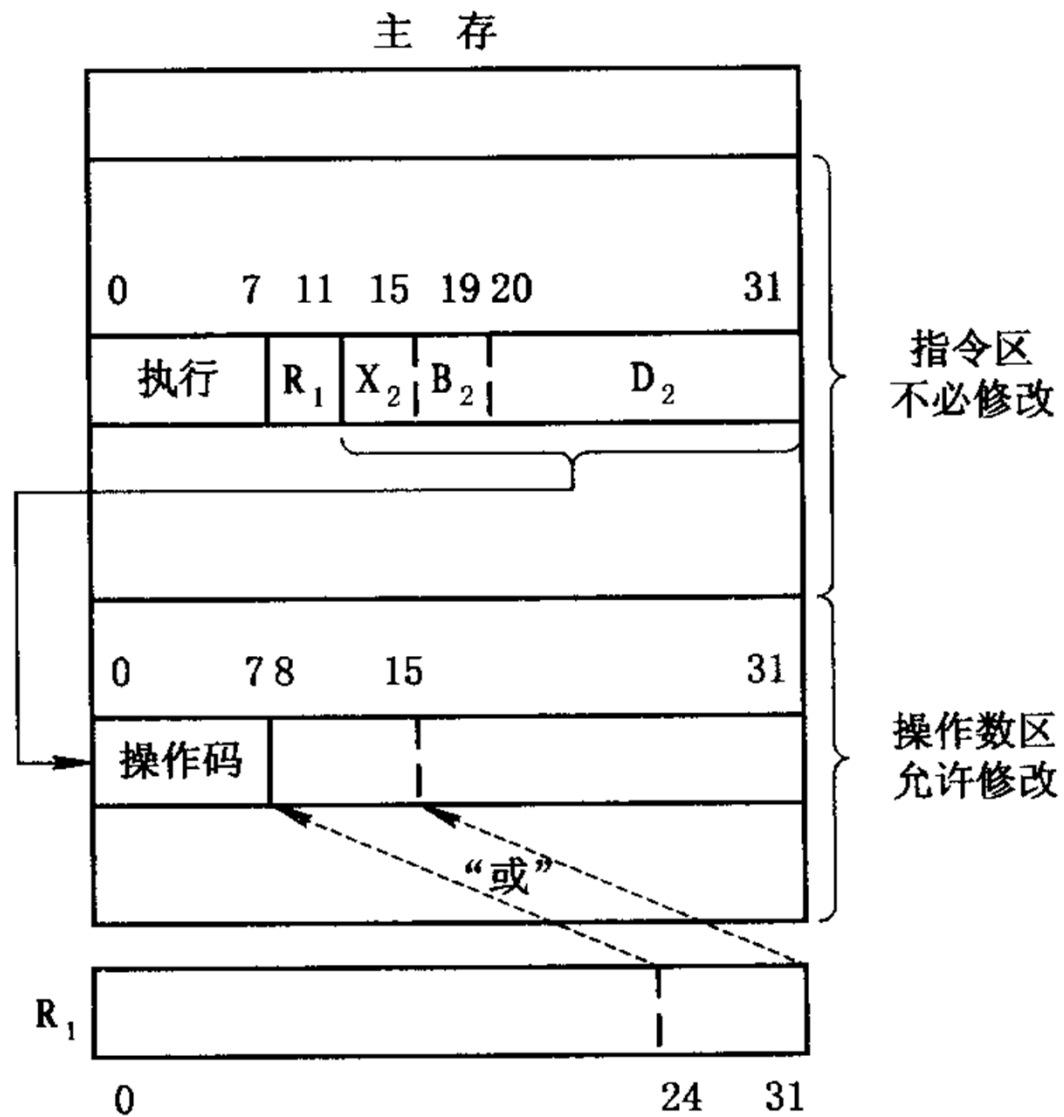


“执行”指令本身并不直接执行，实际指令的指令地址由第二个操作数地址指定：

$$((X2)+(B2)+D2)$$

该地址不在指令区，而是在数据区。程序执行到“执行”时，先修改数据区对应地址的“指令”内容，然后，转去执行该“指令”，因此修改的是数据，而非指令，即通过修改数据而达到修改指令的目的。将指令相关转化成了数相关，统一按数相关进行处理。

# 指令相关与处理





# 主存空间数相关与处理

- **主存空间数相关**：相邻两条指令对同一单元进行“**先写后读**”而出现的关联，使得后一条指令“分析<sub>k+1</sub>”读出的操作数不是前一条指令“执行<sub>k</sub>”应当写入的结果。

K:    OP   A1, A2, A3 ; A1=(A2) OP (A3), 先写  
K+1: OP   A4, A1, A5 ; A4=(A1) OP (A5), 后读

**对于A1主存单元来说，存在“先写后读”相关。**

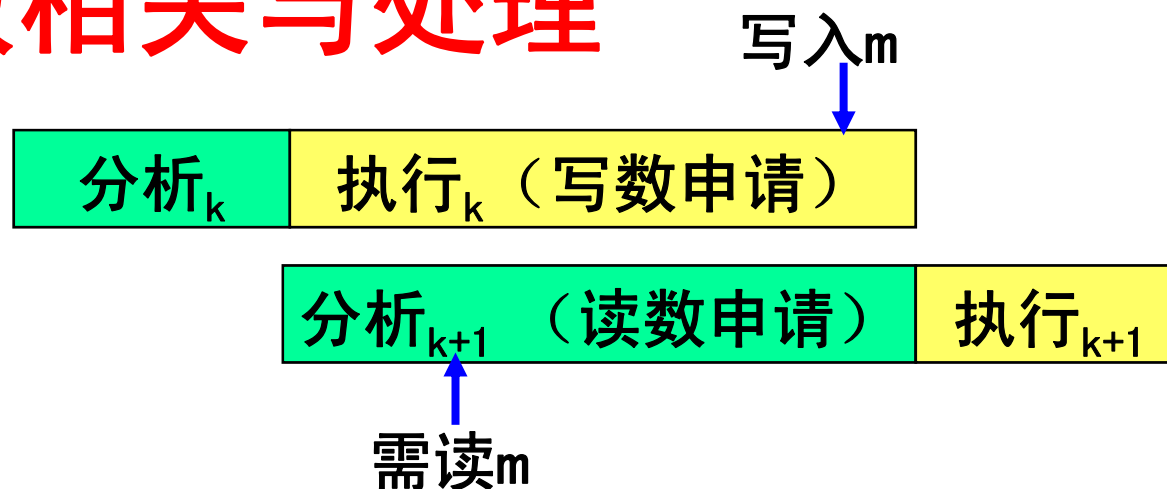
# 主存空间数相关与处理

## 解决方法:

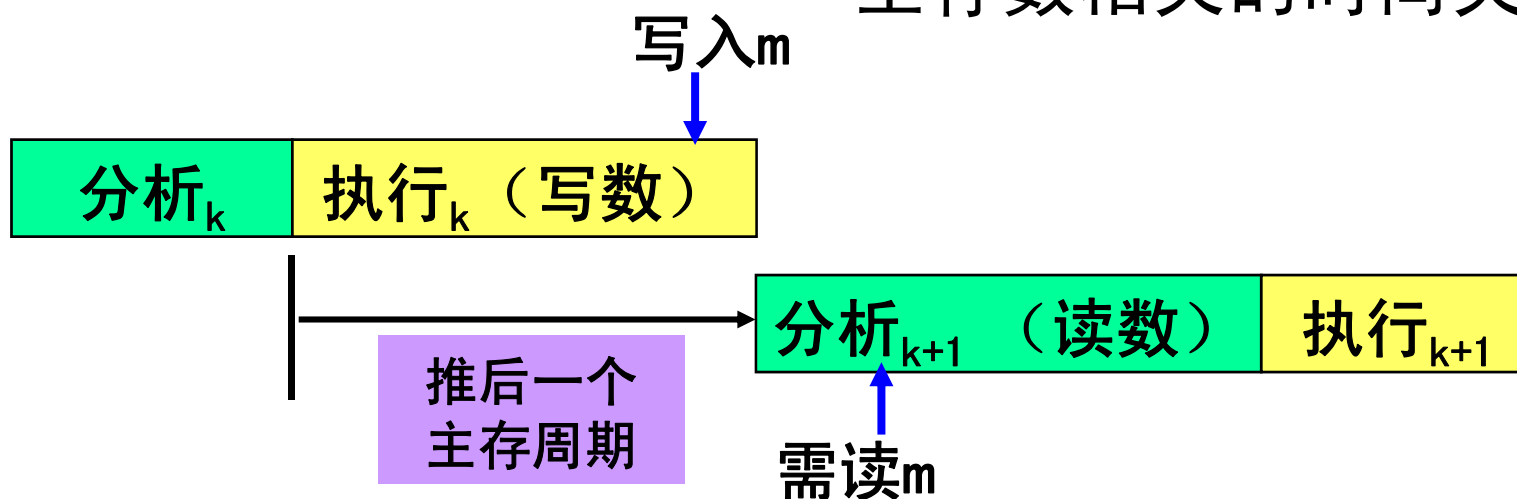
### ■ 推后“分析<sub>k+1</sub>”的读（推后读）

- 无需增加硬件，指令K+1推迟一个周期分析
- 常见的方法是由控存为读数、写数安排不同的访存优先级，让写数优先级高于读数优先级即可。

# 主存空间数相关与处理



主存数相关的时间关系



主存数相关“推后读”的时间关系

# 通用寄存器组数相关的处理

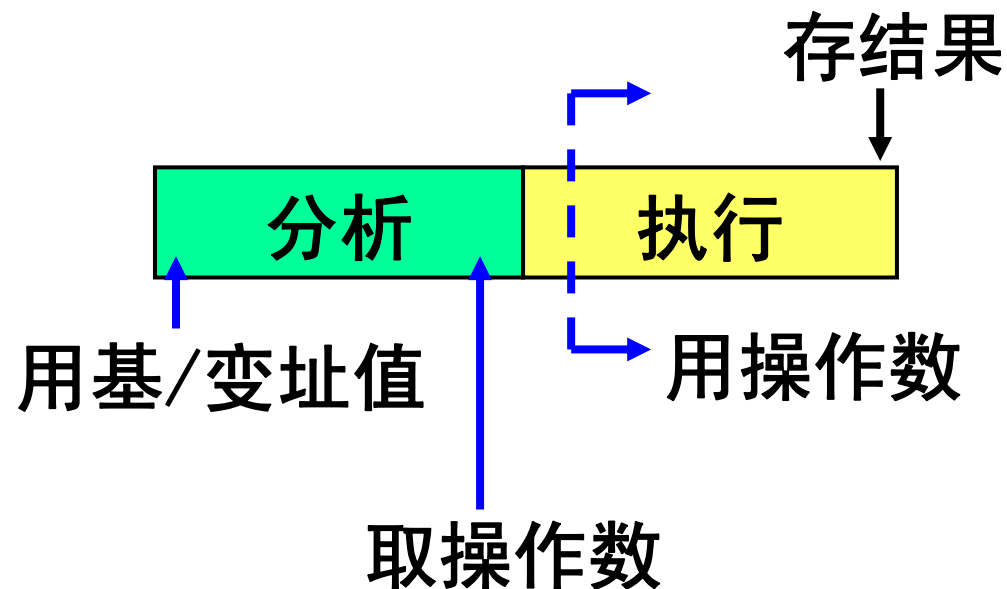
- 通用寄存器可以存放操作数、运算结果，也可以存放操作数物理地址的基址或变址值。
- 假设指令：



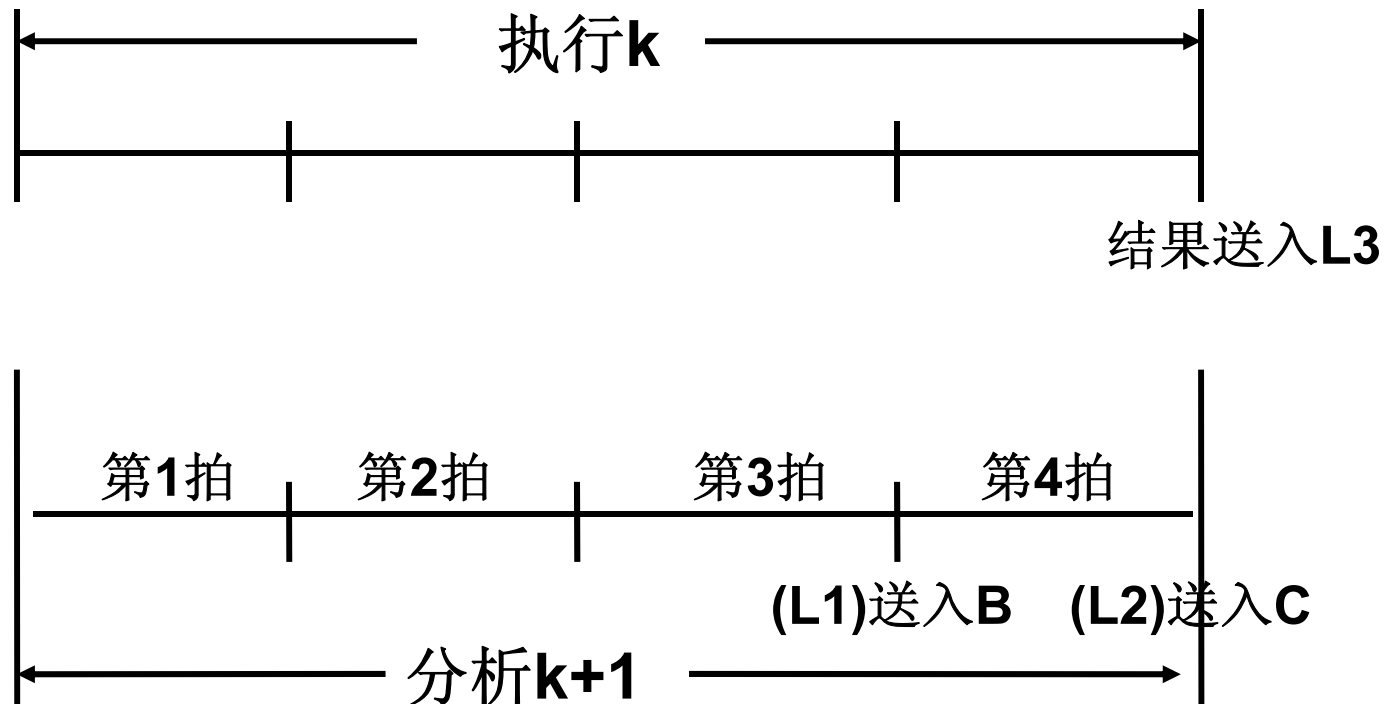
- L1，L2，L3分别指明存放第1操作数、第2操作数和结果数的通用寄存器号；
- B2为形成第二操作数地址的基址值所在的通用寄存器号；
- d2为相对位移量。

# 通用寄存器组数相关的处理

- 但在指令解释过程中，使用通用寄存器作为不同用途所需要的微操作的时间要求并不相同。



# 通用寄存器组数相关的处理



“执行 $k$ ”、“分析 $k+1$ ”重叠时，访问通用寄存器组的时间关系

# 通用寄存器组数相关的处理

## ■ 通用寄存器组数相关分为：

- 通用寄存器组操作数相关
- 通用寄存器组变址值/基址值相关

## ■ 通用寄存器数相关与通用寄存器的基址值或变址值相关的处理是不同的。

# 通用寄存器组操作数相关处理

**K:            OP   R1, A2    ; R1=(R1) OP (A2)**  
**K+1:        OP   R1, R2    ; R1=(R1) OP (R2)**

对于**R1**通用寄存器来说，存在“先写后读”相关。

## 解决方法：

- ①推后“分析<sub>k+1</sub>”的读（**推后读**）
- ②设置“**相关专用通路**”



# 通用寄存器组操作数相关处理

## ■ ①推后“分析<sub>k+1</sub>”的读（推后读）

- 原理同主存数相关
- 优点：基本上不需要增加设备
- 缺点：速度损失较大，控制略为复杂，使重叠效率急剧下降



推迟一个节拍情况：



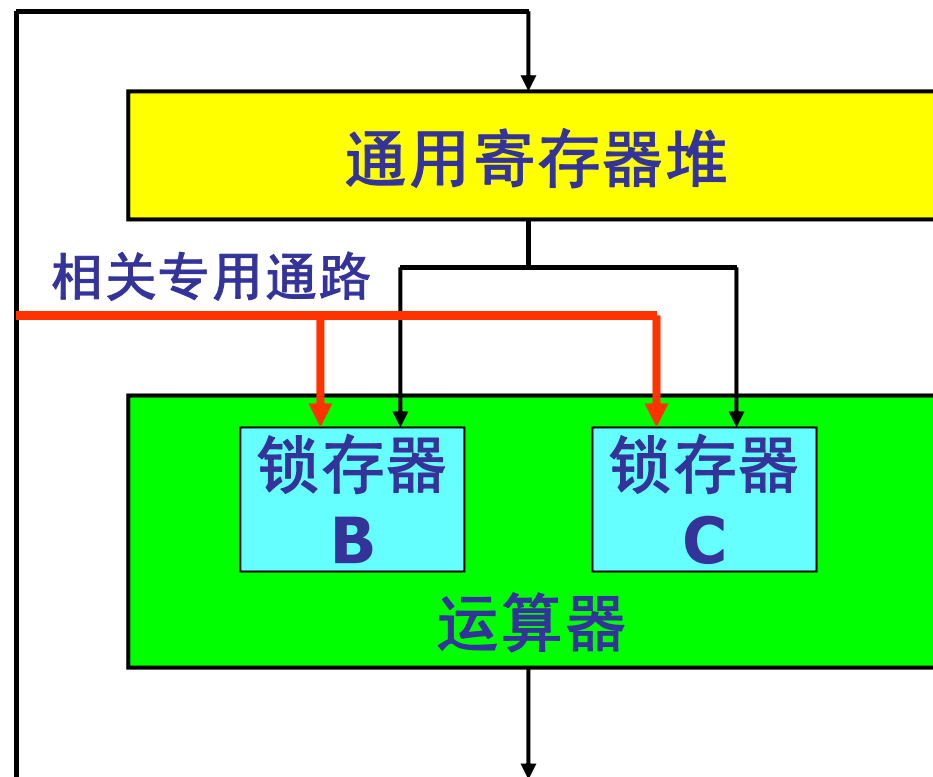
# 通用寄存器组操作数相关处理

## ■ ②设置“相关专用通路”

- 即在运算器和通用寄存器组之间增设一条“相关专用通路”（硬件）；
- 当发生相关时，让相关专用通路接通，在将运算结果送通用寄存器组的同时，直接将运算结果回送到运算器。
- 优点：缩短了传送时间，即可以保证重叠效率不下降，也可以保证重叠解释时不出错
- 缺点：需要增加设备

# 通用寄存器组操作数相关处理

## ■ ②设置“相关专用通路”



# 通用寄存器组基址值/变址值相关处理

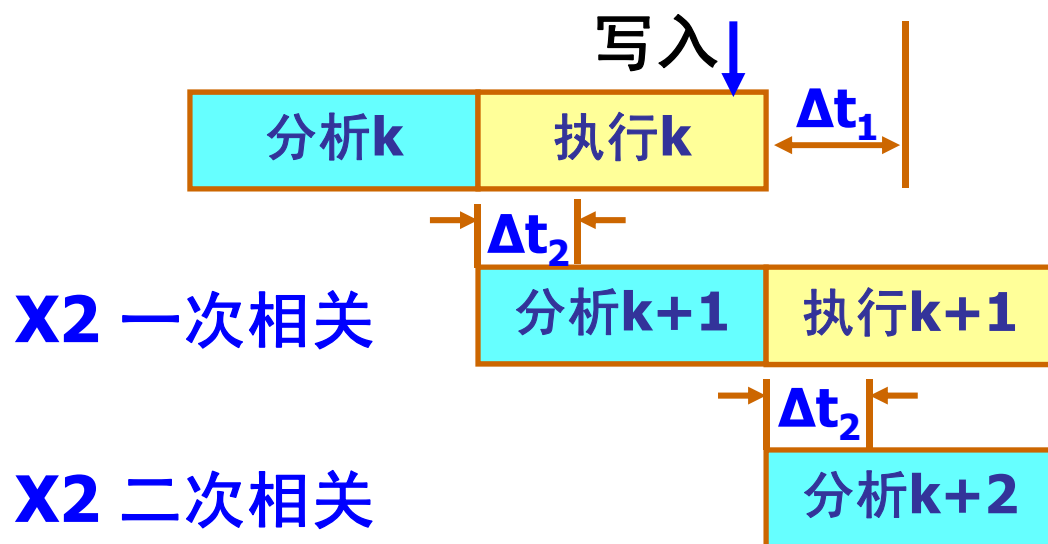
**K: OP R1, R2 ; R1=(R1) OP (R2)**

**K+1: OP R1, A2(X2) ; R1=(R1) OP ((A2) + (X2))**

**K+2: OP R1, A2(X2) ; R1=(R1) OP ((A2) + (X2))**

如果发生:  $X2(K+1) = R1(K)$ , 称为一次(变址)相关。

如果发生:  $X2(K+2) = R1(K)$ , 称为二次(变址)相关。



如果  $\Delta t_1 > \Delta t_2$ , 地址 (变址) 将出错!

# 通用寄存器组基址值/变址值相关处理

## 解决方法:

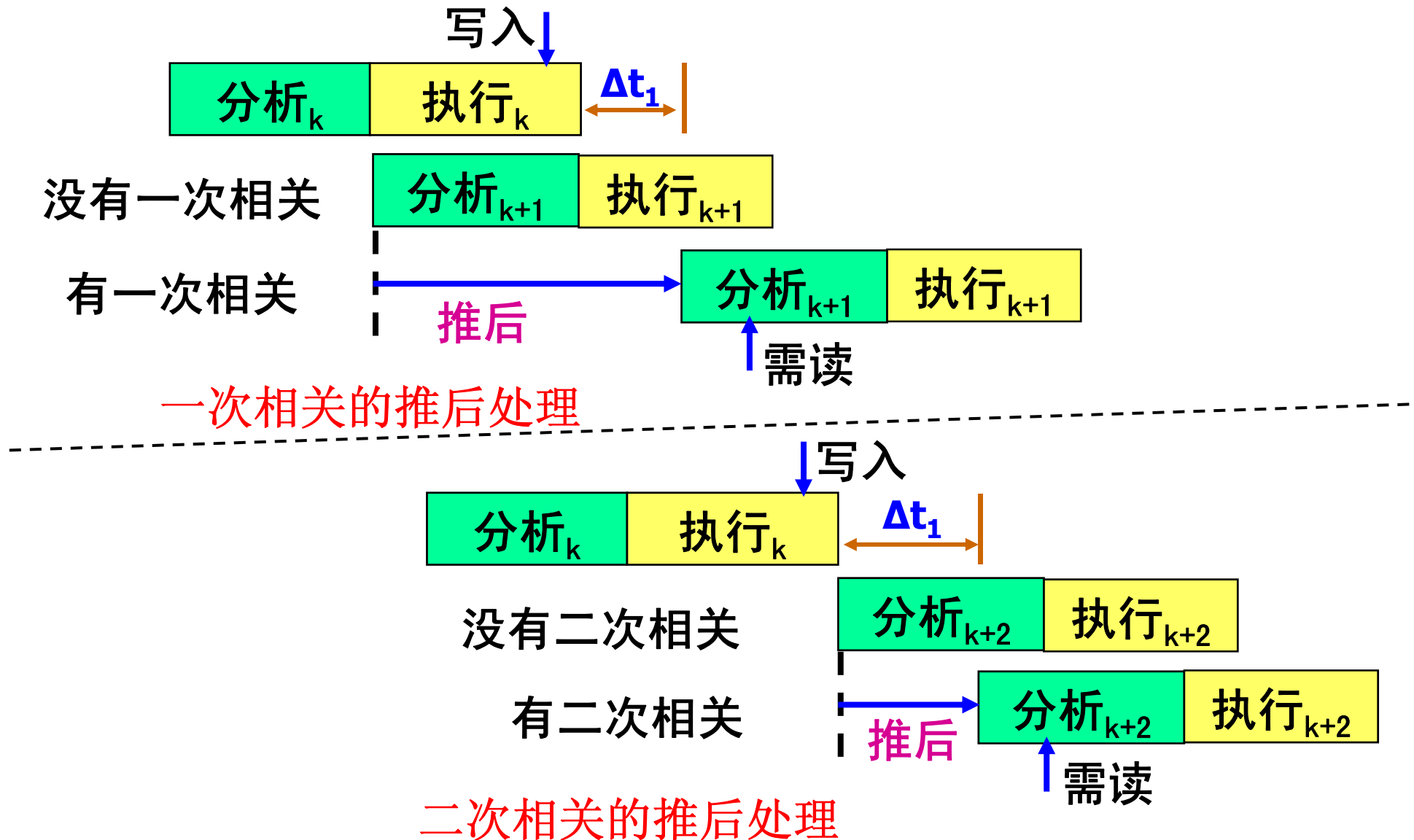
- ①推后“分析<sub>k+1</sub>”的读（推后读）
- ②设置“相关专用通路”

# 通用寄存器组基址值/变址值相关处理

## ■ ①推后“分析<sub>k+1</sub>”的读（推后读）

- 对于一次相关，除推后“分析<sub>k+1</sub>”外，还需要再推后一个“执行”周期；
- 对于二次相关，只需推后“分析<sub>k+2</sub>”的起点，使“执行<sub>k</sub>”送入通用寄存器的结果，在“分析<sub>k+2</sub>”开始时已经出现在通用寄存器的输出总线上即可。

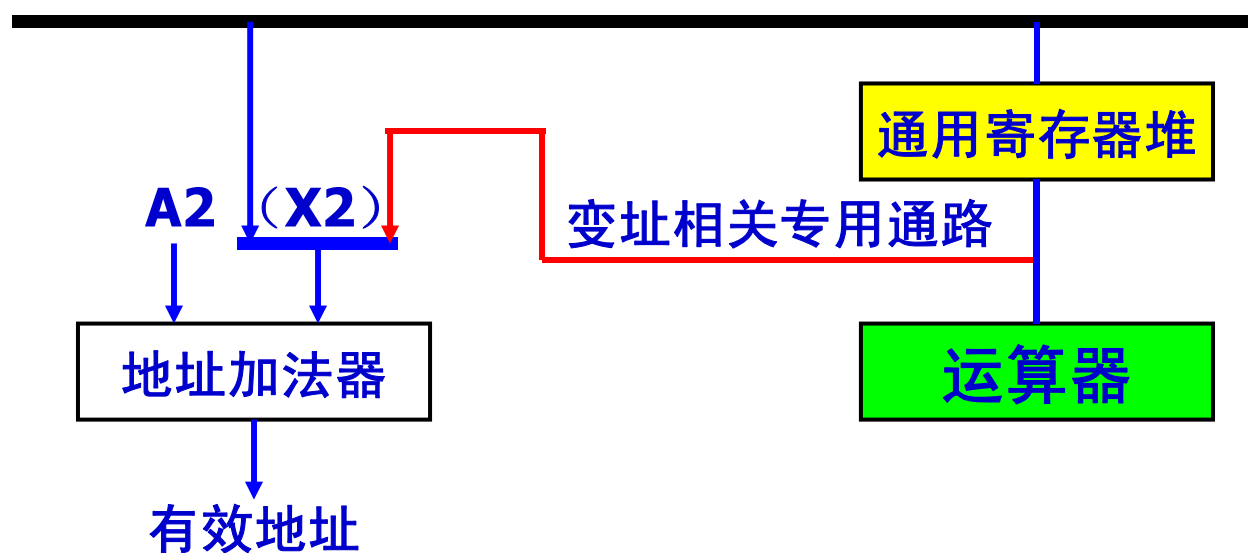
# 通用寄存器组基址值/变址值相关处理



# 通用寄存器组基址值/变址值相关处理

## ■ ②设置“相关专用通路”

- 通过相关专用通路，可以把“执行<sub>k</sub>”得到的运算结果，在送入通用寄存器的同时，直接送入“访存操作数地址”形成机构





# 通用寄存器组基址值/变址值相关处理

## ■ ②设置“相关专用通路”

- 对于一次相关，只需使“分析<sub>k+1</sub>”推后到接着“执行<sub>k</sub>”进行就可以了；
- 对于二次相关，不必推后“分析<sub>k+2</sub>”。

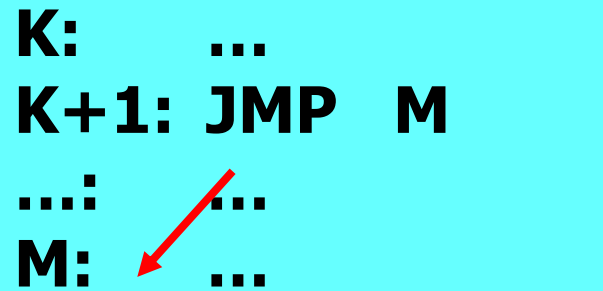
# 控制相关

- 无条件转移
- 一般条件转移
- 复合条件转移
- 子程序调用
- 中断 等

**吸收型指令：**  
在指令分析器中就执行完成的指令，如无条件转移指令、一般条件转移指令。

# 无条件转移

K: ...  
K+1: **JMP M** ;  $M \rightarrow (PC), M \rightarrow (PC1)$   
...:  
M: ...



无条件转移指令

分析<sub>k</sub>

执行<sub>k</sub>

分析<sub>k+1</sub>

转移成功，且指令m在指缓中

分析<sub>m</sub>

执行<sub>m</sub>

转移成功，且指令m不在指缓中

取指<sub>m</sub>

分析<sub>m</sub>

执行<sub>m</sub>

# 一般条件转移

K: ...

K+1: **JMP (CC)** M

...: ...

M: ...

; 置条件码CC

; 如果CC为TRUE, 转向M; 否则继续

产生条件转移

条件转移指令

分析<sub>k</sub>

执行<sub>k</sub>

转移不成功时

分析<sub>k+1</sub>

执行<sub>k+1</sub>

转移成功, 且指令m在指缓中

分析<sub>k+1</sub>

分析<sub>m</sub>

执行<sub>m</sub>

转移成功, 且指令m不在指缓中

分析<sub>k+1</sub>

取指<sub>m</sub>

分析<sub>m</sub>

执行<sub>m</sub>

# 复合条件转移

K: OP M

K+1: ... ↓

...: ...

M: ...

；先执行OP操作，根据结果决定是否转向M

条件转移指令

分析<sub>k</sub>

执行<sub>k</sub>

产生条件转移

转移不成功时

分析<sub>k+1</sub>

执行<sub>k+1</sub>

转移成功，且指令m在指缓中

分析<sub>m</sub>

执行<sub>m</sub>

转移成功，且指令m不在指缓中

取指<sub>m</sub>

分析<sub>m</sub>

执行<sub>m</sub>

# 控制相关

- 转移成功造成的影响比转移不成功造成的影响大得多！
  - 当转移成功时，预取的指令变成无效，重叠方式变成了顺序方式。
- 因此在重叠方式的机器中，应尽可能不使用条件转移指令。
- 若使用应尽可能使用不成功的转移指令。
- 当然，也可以采用**RISC**的延迟转移技术，使重叠效率不至下降。

# 5.1 重叠解释方式小结

## ■ 为实现两条指令的同时解释执行

### ● 首先，要付出空间上的代价

- ◆ 增设数据总线、控制总线、指令缓存器、地址加法器、相关专用通路等；
- ◆ 需要设置单独的指令分析部件和执行部件；
- ◆ 需要将主存采用多体交叉存取等；

# 5.1 重叠解释方式小结

## ■ 为实现两条指令的同时解释执行

- 其次，要处理好指令之间可能存在的相关，包括指令相关、主存数相关、通用寄存器组操作数相关，以及通用寄存器组基址值或变址值相关等。采用的方法有两个：

- ◆ 推后“分析”（推后读）
- ◆ 设置相关专用通路



# 5.1 重叠解释方式小结

## ■ 为实现两条指令的同时解释执行

- 再有，合理安排调配每条指令的微操作，使“分析”和“执行”所需的时间尽可能匹配，以提高重叠效率。

# 学习内容

- 5.1 重叠方式
- 5.2 流水方式
- 5.3 向量的流水处理与向量处理机
- 5.4 指令级高度并行的超级处理机
- 5.5 ARM流水线处理器举例

## 5.2 流水方式

- 流水线的基本概念
- 流水线分类
- 流水线性能
- 流水线相关
- 流水线调度

## 5.2.1 流水线的基本概念

加快机器语言程序的执行可以通过以下两个途径实现：

### ■ 提高每一条指令的执行速度

- 通过选用更高速的器件、采取更好的算法，提高指令内部各微操作的并行度。

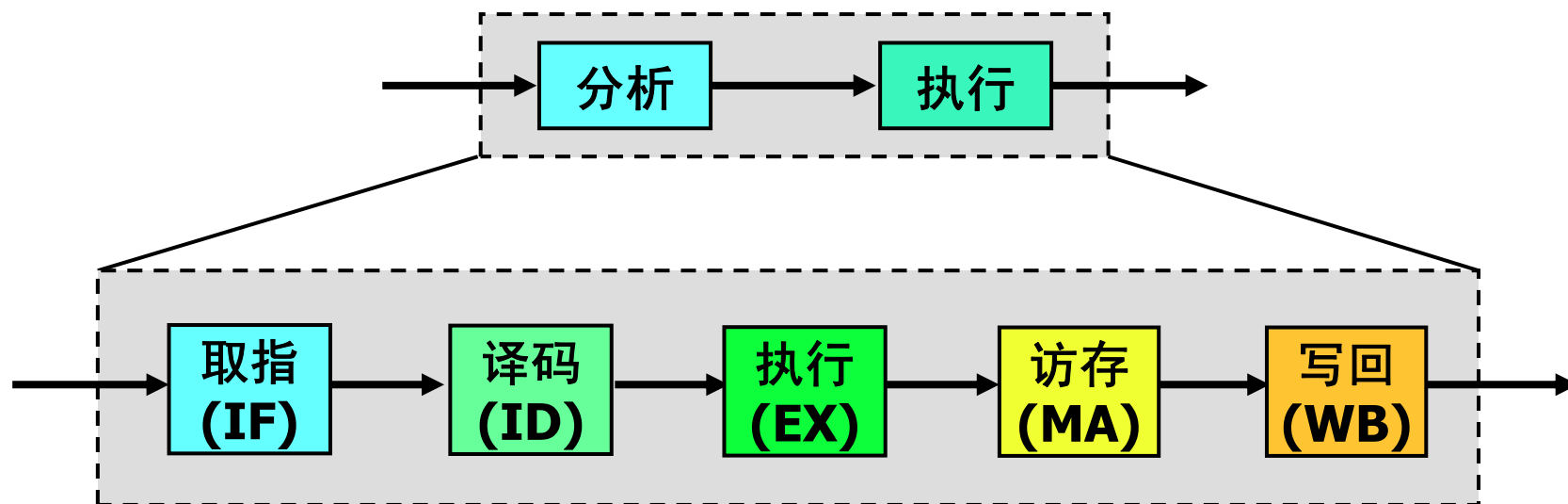
### ■ 提高多条指令的执行速度

- 通过控制机构采用同时解释和执行两条、多条甚至整段程序的控制方式，提高指令间的并行度。
- 流水线技术是目前广泛应用的一项关键技术。

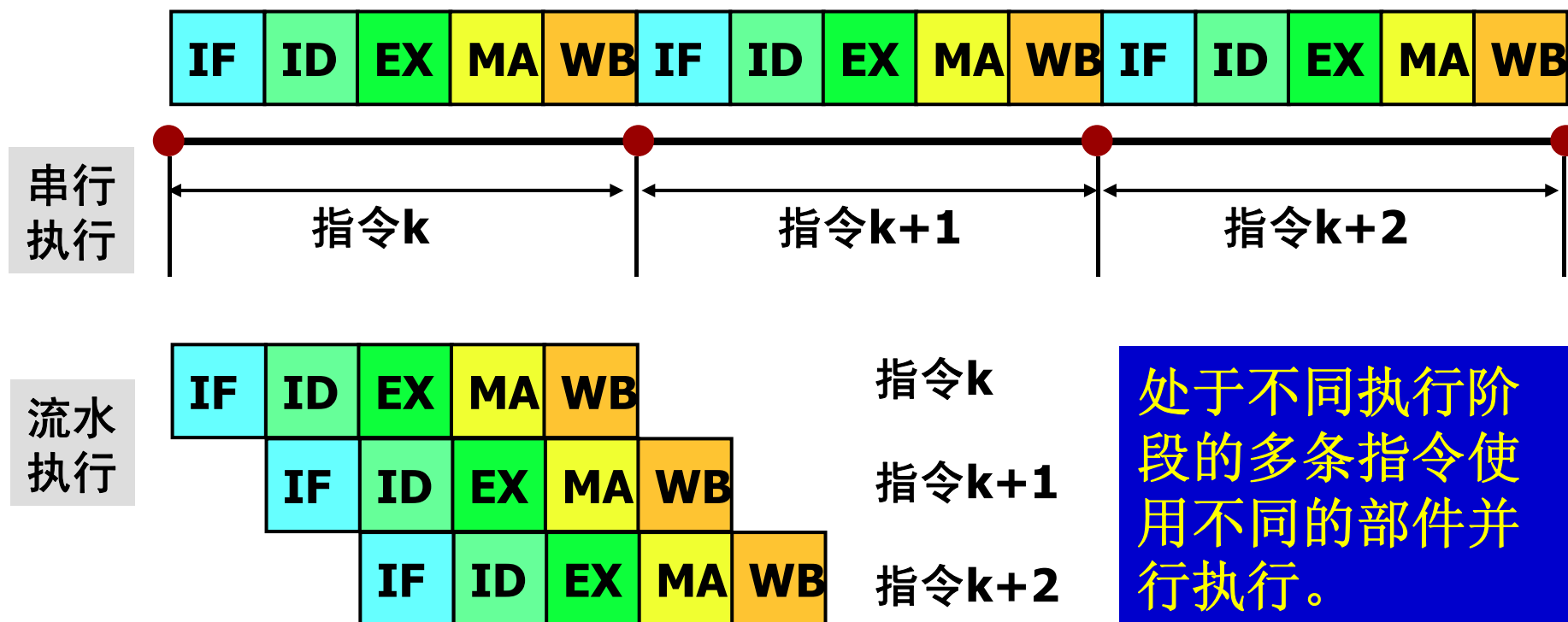
## 5.2.1 流水线的基本概念

### ■ 流水线技术：流水是重叠的引申。

将指令的执行过程分解为多个子过程，并让每个子过程分别由专用的部件完成，这些功能部件可以同时工作。让多条指令在时间上错开，依次通过各功能部件，这样，每个子过程就可以与其他的子过程并行进行，从而实现多条指令的并行执行，减少多条指令或一段程序的完成时间。



## 5.2.1 流水线的基本概念



假设每个子过程的处理时间均为 $\Delta t$ ，则：

串行执行完成时间= $5 \times \Delta t \times 3 = 15\Delta t$

流水执行完成时间= $7\Delta t$

# 流水线特点

- 是一种基于**时间重叠**的并行处理技术；不能加快一条指令的执行，能加快多条指令的执行；
- 由多个有联系的子过程组成。这些子过程称为流水线的“**级**”或“**段**”。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。
- **段的数目称为流水线的“深度”**；
- 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。设置寄存器会增加指令执行时间。
- **为了简化，在一般流水线中不画出流水锁存器。**

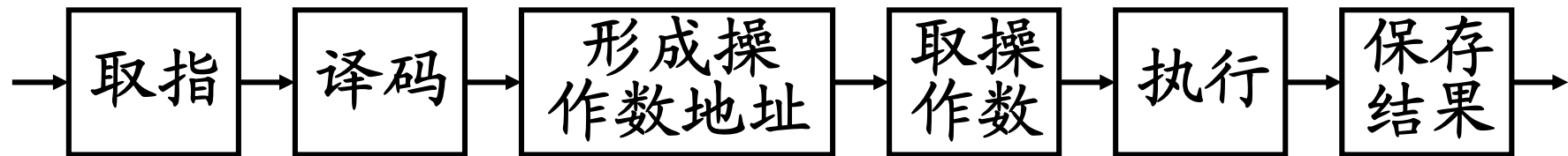
# 流水线特点

- 每个子过程分别由**专用的部件**完成。这些部件可以同时工作；
- 各流水段的**时间应尽量相等**。
- 流水线工作阶段可分为**建立、满载和排空三个阶段**：
  - 从第一个任务进入流水线到流水线所有部件都处于工作状态这个时期，称为**流水线建立阶段**；
  - 当所有部件都处于工作状态时，称为**流水线满载阶段**；
  - 从最后一条指令流入流水线到结果流出，称为**流水线的排空阶段**。
  - 建立和排空阶段所用的时间分别叫做“装入时间”和“排空时间”。
- **适合大量重复的时序过程**。只有连续不断地向流水线输入任务，才能发挥流水线的效力。

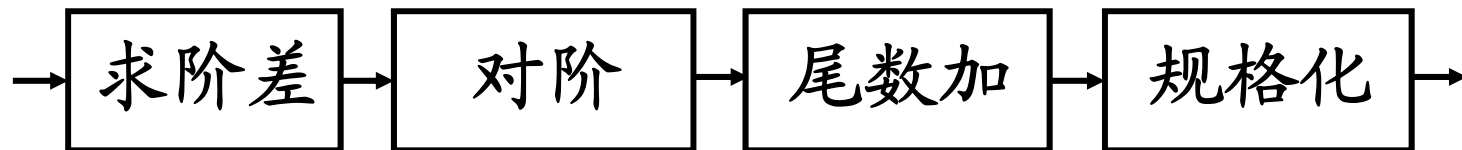


# 流水线特点

- 一般指令流水线有**4-12**个流水段。
- 等于及大于**8**个流水段的处理机称为**超流水线处理机**。

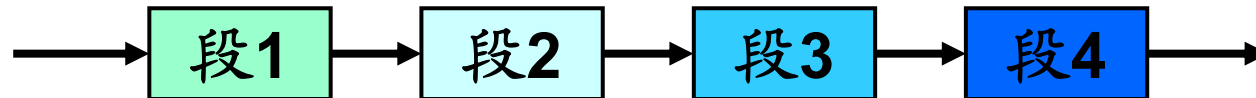


- 复杂指令的**执行阶段**也采用流水线，例如浮点加法器的流水线。



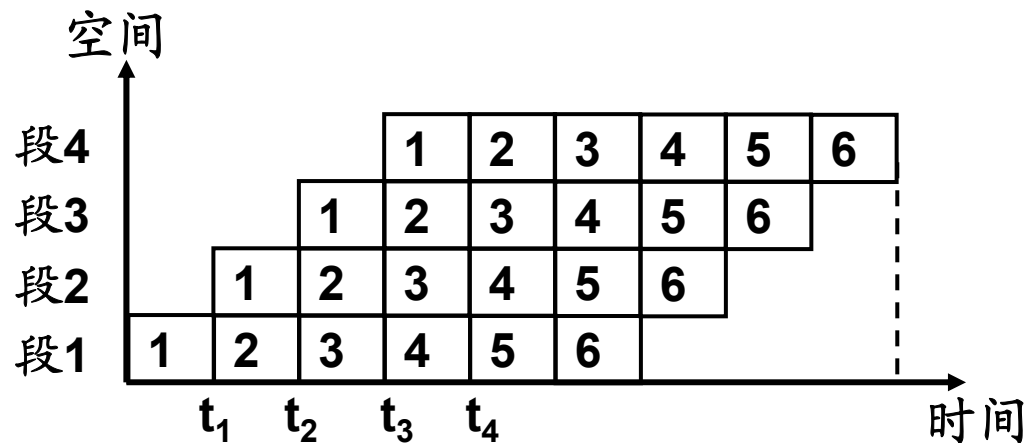
# 流水线表示方法

- **连线图：** 将各功能段用连接线连接在一起。



- **时空图：** 表示指令执行过程。

- 横坐标表示时间，纵坐标表示空间，即流水线的各个功能段。



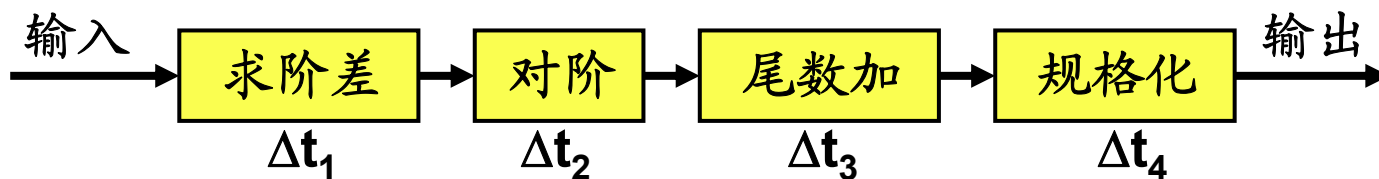
# 流水线分类

- 部件级/处理机级/处理机间级（宏流水线）
- 单功能/多功能
- 静态/动态
- 线性/非线性
- 标量/向量
- 同步/异步
- 顺序/乱序 等

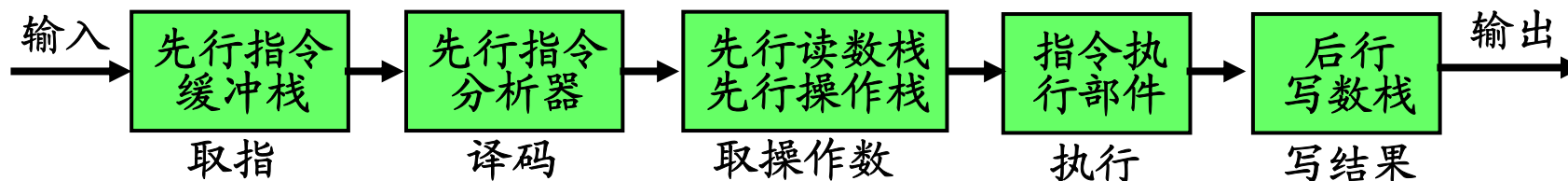
# 部件级 / 处理机级 / 处理机间级

- 根据向上和向下扩展的思想分类。
- **部件级**（又称为运算操作流水线）
  - 指构成部件内的各**子部件之间**的流水。
- **处理机级**（又称为指令流水线）
  - 指构成处理机的**各个部件之间**的流水。
- **系统级**（又称为宏流水）
  - 指构成计算机系统的**多个处理机之间**的流水。

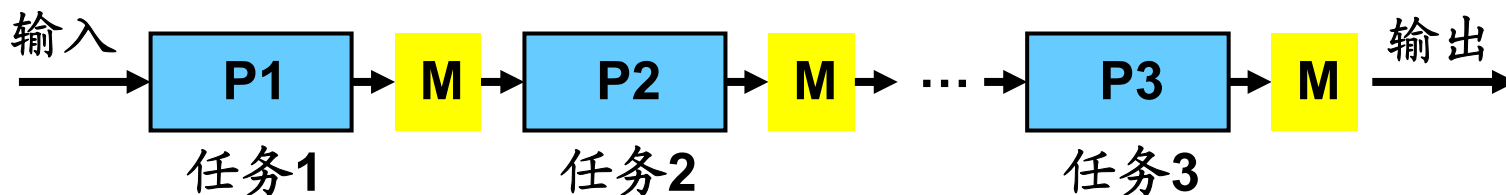
# 部件级 / 处理机级 / 处理机间级



部件级流水线（操作流水线）——浮点加法器流水线



处理机级流水线——先行控制方式中的指令流水线



处理机之间的流水线——宏流水线

# 单功能/多功能

## ■ 从流水线具有的功能分类。

### ■ 单功能流水线

- 指只能实现一种功能的流水线。
- 例如：只实现浮点加减的流水线
- 可以将多条单功能流水线组合起来，完成多功能的流水。

### ■ 多功能流水线

- 指同一流水线的各个段之间可以有多种不同的联接方式，以实现多种不同的运算或功能。

# 静态/动态

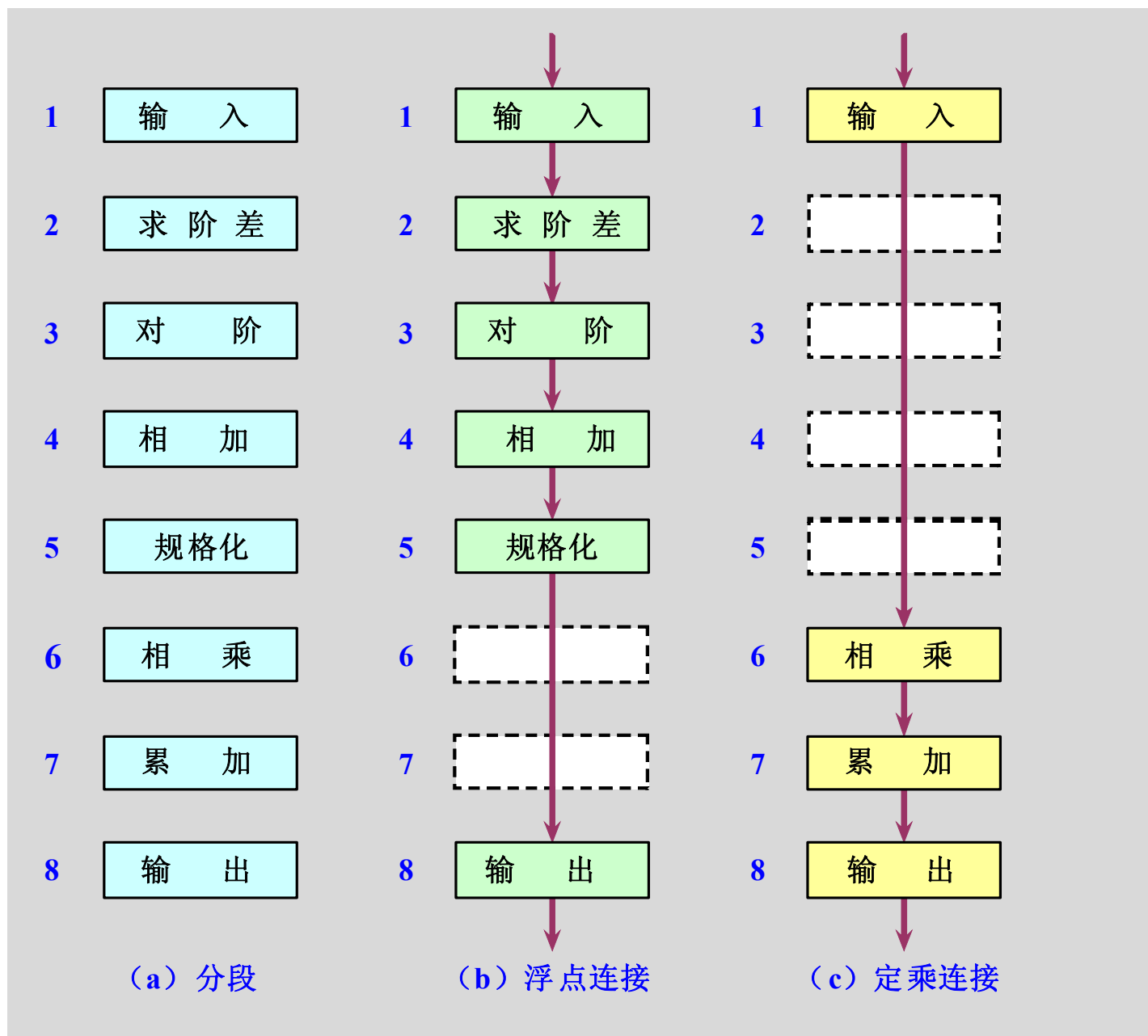
## ■ 从功能部件的联接关系上对多功能流水线分类。

### ■ 静态流水线

- 在某一时间段内，各功能段只能按一种功能的联接流水。只有等流水线全部流空后，才能切换成按另一种功能来联接流水。
- 适合于处理一串相同的运算操作。

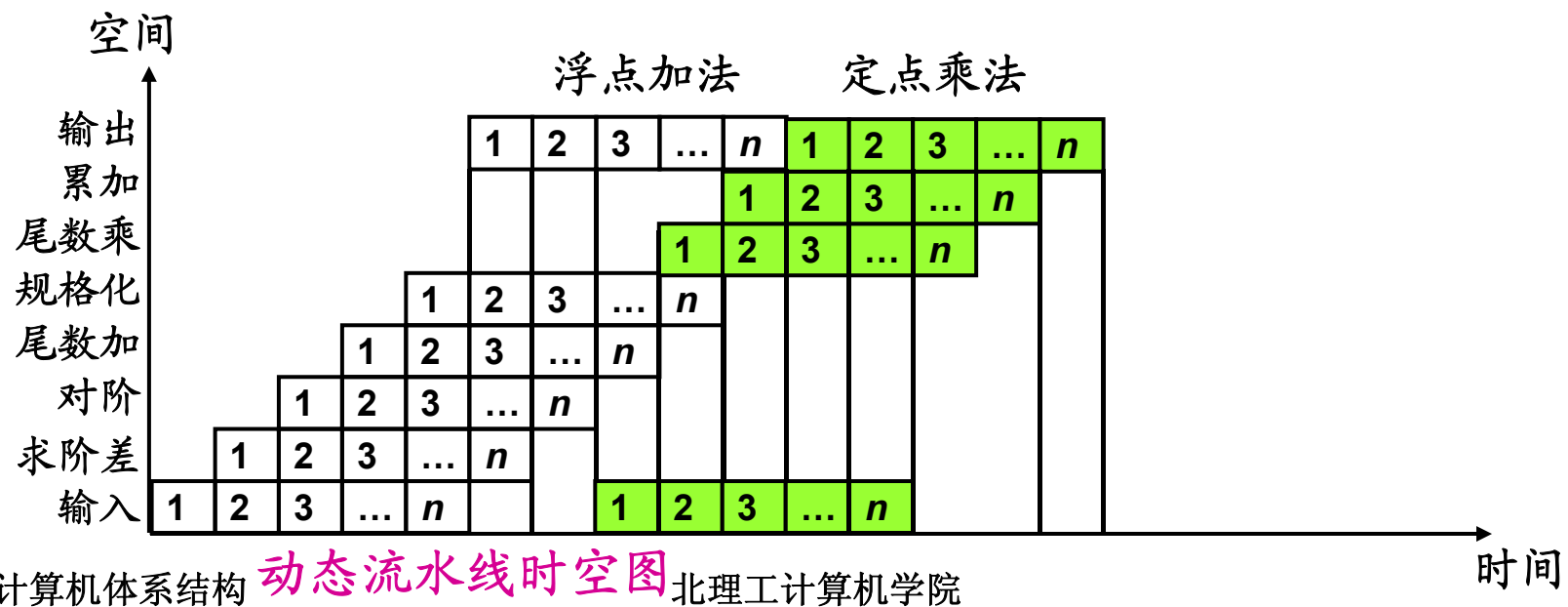
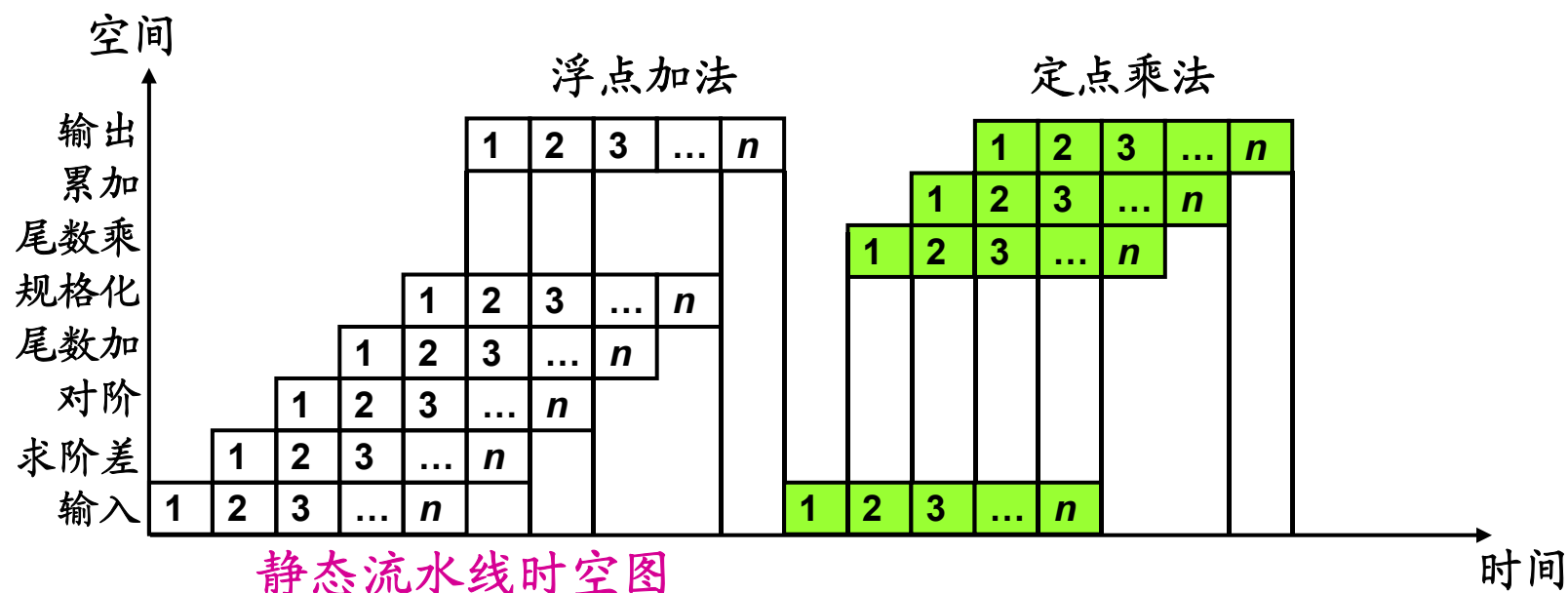
### ■ 动态流水线

- 各功能段在同一时间内可根据需要进行联接流水。当某些段正在实现某种运算时，另一些段却在实现另一种运算。
- 能提高流水的吞吐率和设备的利用率，但控制复杂，成本高。





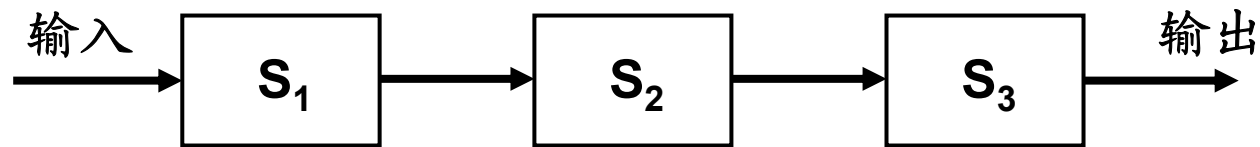
# 静态/动态



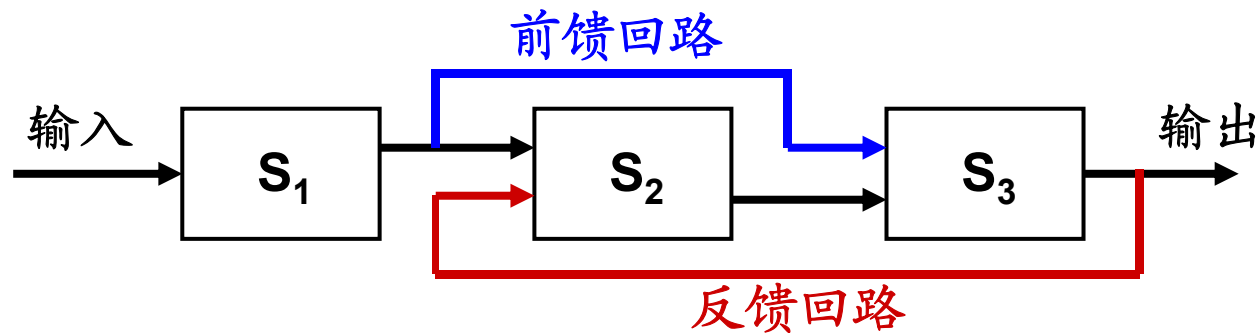
# 线性/非线性

- 按流水线各功能段之间是否有反馈回路分类。
- 线性流水线
  - 各功能段串行联接，没有反馈回路；
  - 各个段只经过一次。
- 非线性流水线
  - 各功能段不仅有串行联接，还有反馈回路；
  - 一个任务需要多次经过或越过某些段；
  - 其中一个重要的问题是流水线的调度：即何时向流水线输入新的任务，使此任务经过流水线各功能段时，不会与先前进入的任务争用功能段。

# 线性/非线性



一种简单的线性流水线



一种简单的非线性流水线

# 标量/向量

- 按机器具有的数据表示分类。

- 标量流水处理机

- 没有向量数据表示，仅对标量数据进行流水处理。

- 向量流水处理机

- 具有向量数据表示，能对向量、数组等进行流水处理；
  - 它是向量数据表示和流水技术的结合。

# 顺序/乱序

- 按照任务流出顺序与流入顺序是否相同分类。

- 顺序流水线

- 任务流出顺序与任务流入顺序相同的流水线。
- 顺序流水线又称为同步流水线。

- 乱序流水线

- 任务流出顺序与任务流入顺序不同的流水线。
- 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。

## 5.2.2 流水线处理机的主要性能

主要性能指标：3个

- 吞吐率
- 加速比
- 效率

# 吞吐率

- **定义：**流水线单位时间内能处理的指令条数或能输出的结果数。

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

$n$ ：任务数。

$T_k$ ：处理完成  $n$  个任务所用的时间。

# 吞吐率

## ■ ①最大吞吐率 $TP_{max}$

- 指流水线单位时间内能处理的最多指令条数或能输出的最多结果数。
- 显然，只有在流水线满负荷工作、达到稳定状态后才能达到。

## ■ ②实际吞吐率 $TP$

- 指流水线单位时间内实际处理的指令条数或实际输出的结果数。



# 线性流水线吞吐率 — 最大吞吐率

- 若流水线各段的时间相等，均为 $\Delta t_0$ ，则：

$$TP_{\max} = 1 / \Delta t_0$$

- 若流水线各段时间不等，第 $i$ 段时间为 $\Delta t_i$ ，则：

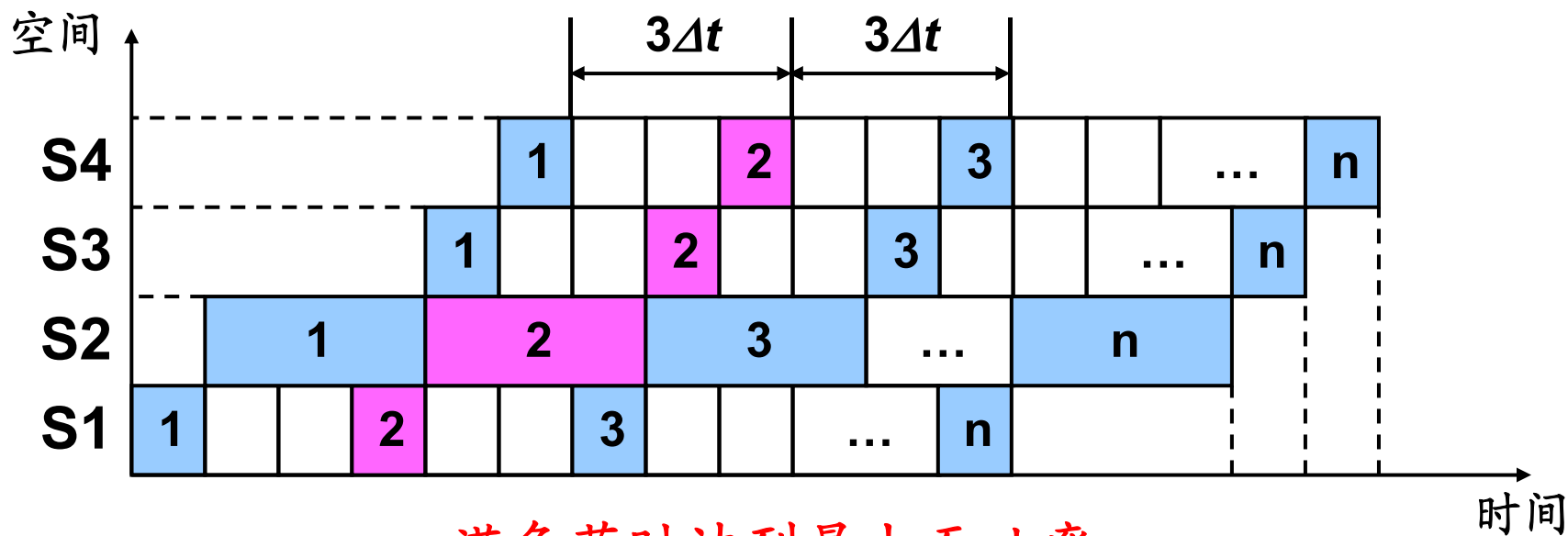
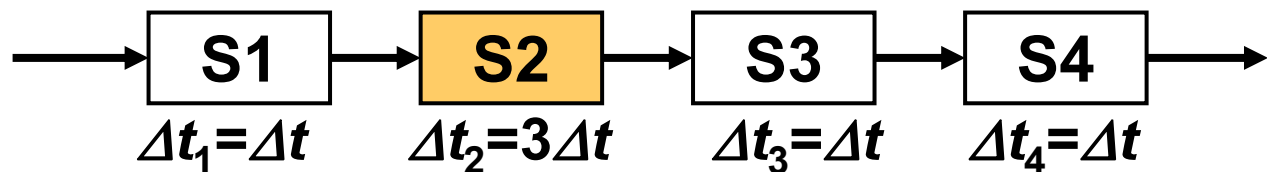
$$TP_{\max} = 1 / \max\{ \Delta t_i \}$$

它受限于流水线中最慢子过程所需要的时间。

将流水线中经过时间最长的子过程为瓶颈子过程。

为了提高流水线的最大吞吐率，必须找出瓶颈子过程，并设法消除。

# 线性流水线吞吐率 — 最大吞吐率



满负荷时达到最大吞吐率。

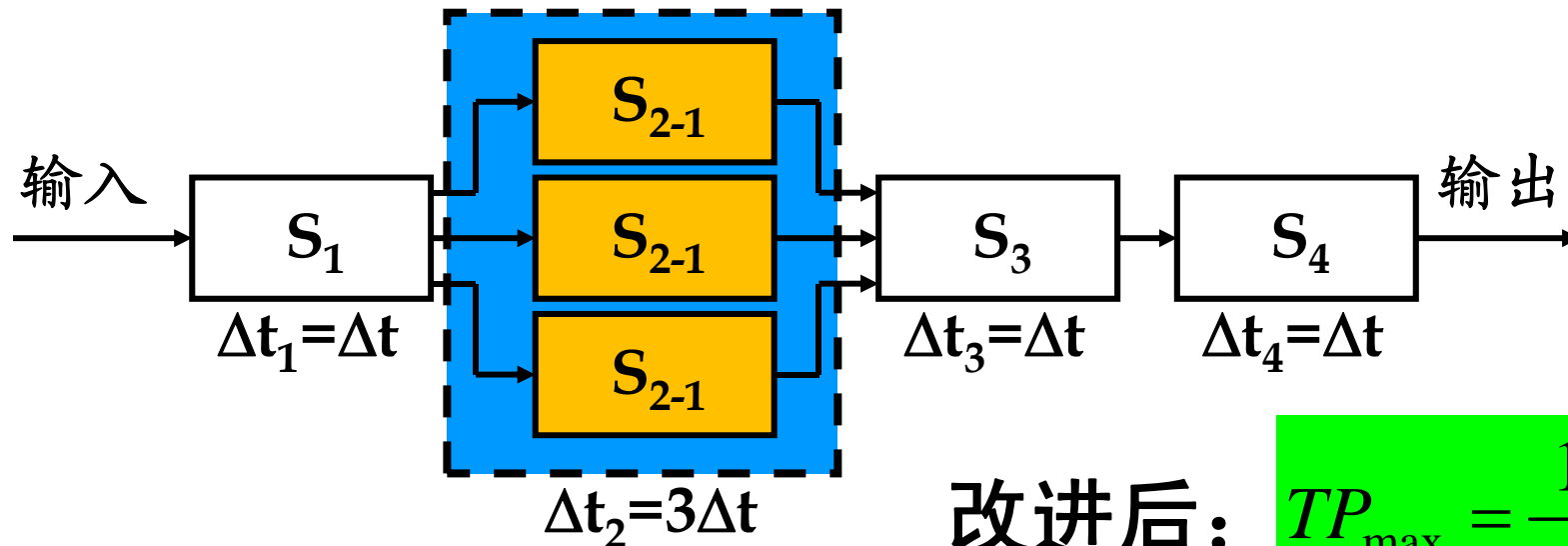
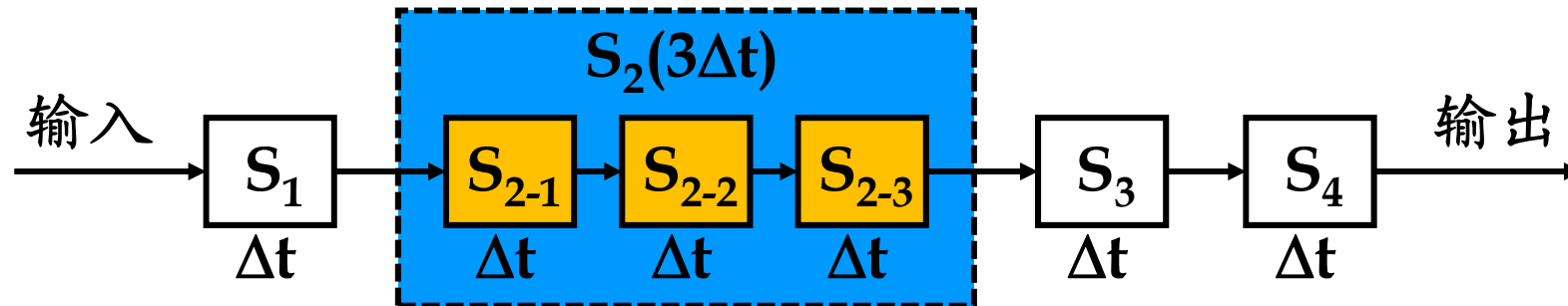
**S2**为瓶颈段。每隔  $3\Delta t$  流出一个结果。  $TP_{max} = 1 / 3\Delta t$

# 线性流水线吞吐率 — 最大吞吐率

## ■ 消除瓶颈子过程的方法有：

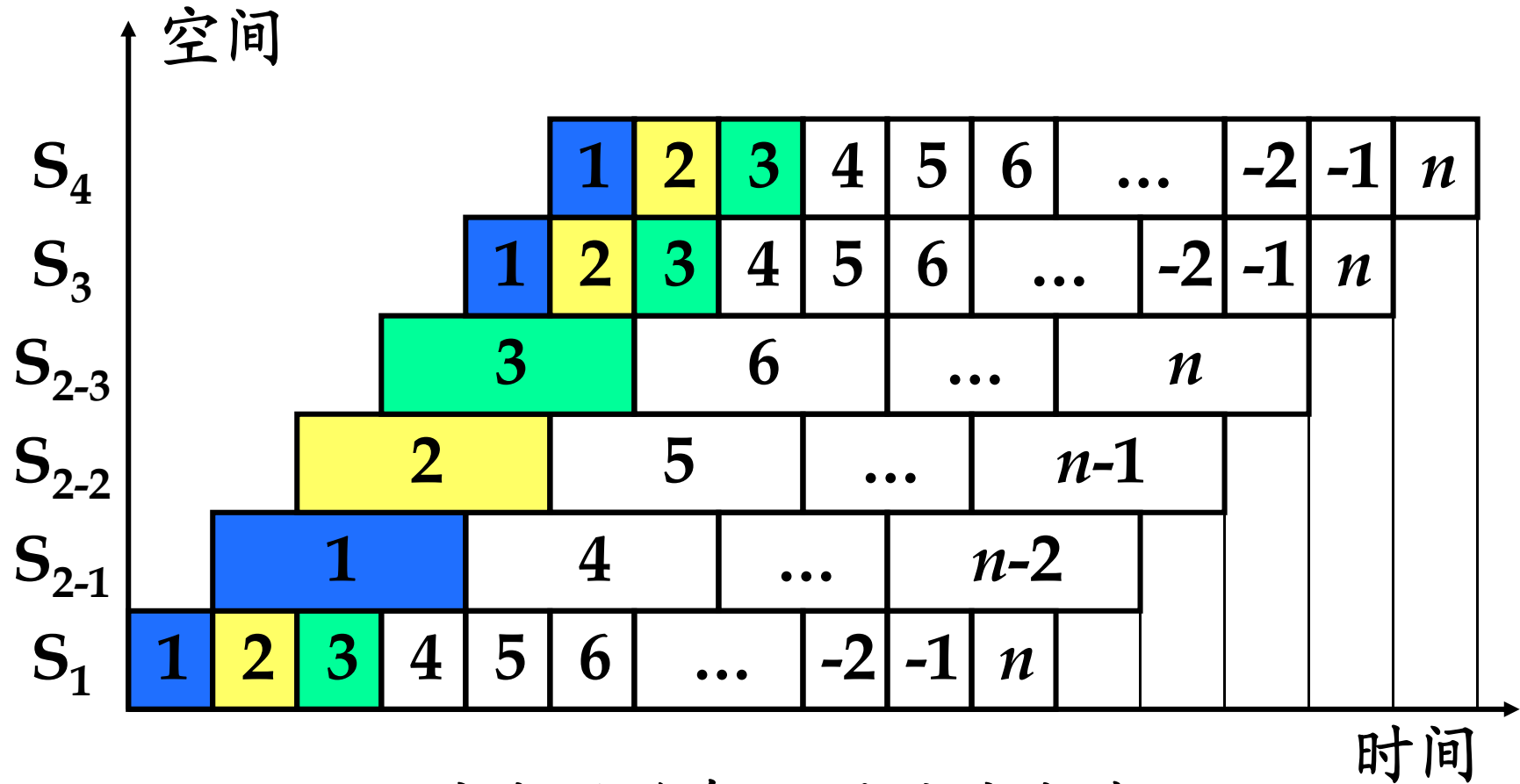
- **细分瓶颈过程：**使各功能段处理时间尽可能相等。
- **重复设置瓶颈功能段，并使它们交叉并行工作。**若瓶颈子过程不能再细分，可以采用此方法。但此方法的设备量增加，需要解决好并行子过程的任务分配和同步。

# 线性流水线吞吐率 – 最大吞吐率



改进后:  $TP_{\max} = \frac{1}{\Delta t}$

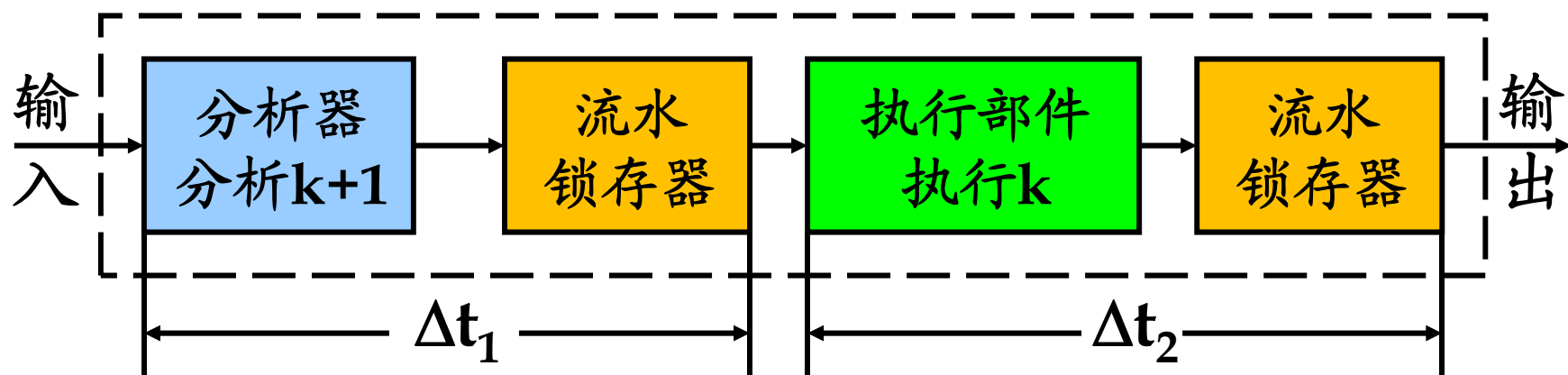
# 线性流水线吞吐率 - 最大吞吐率



## 流水段重复设置的流水线

# 最大吞吐率

- 各子过程所经过的时间会有不同。
- 为平滑各子过程速度差异，一般在子部件设置**高速接口锁存器**。
- 子过程越细分，锁存器数量越多，因此而增大的延迟就越长，从而影响最大吞吐率的提高。



# 线性流水线吞吐率 — 实际吞吐率

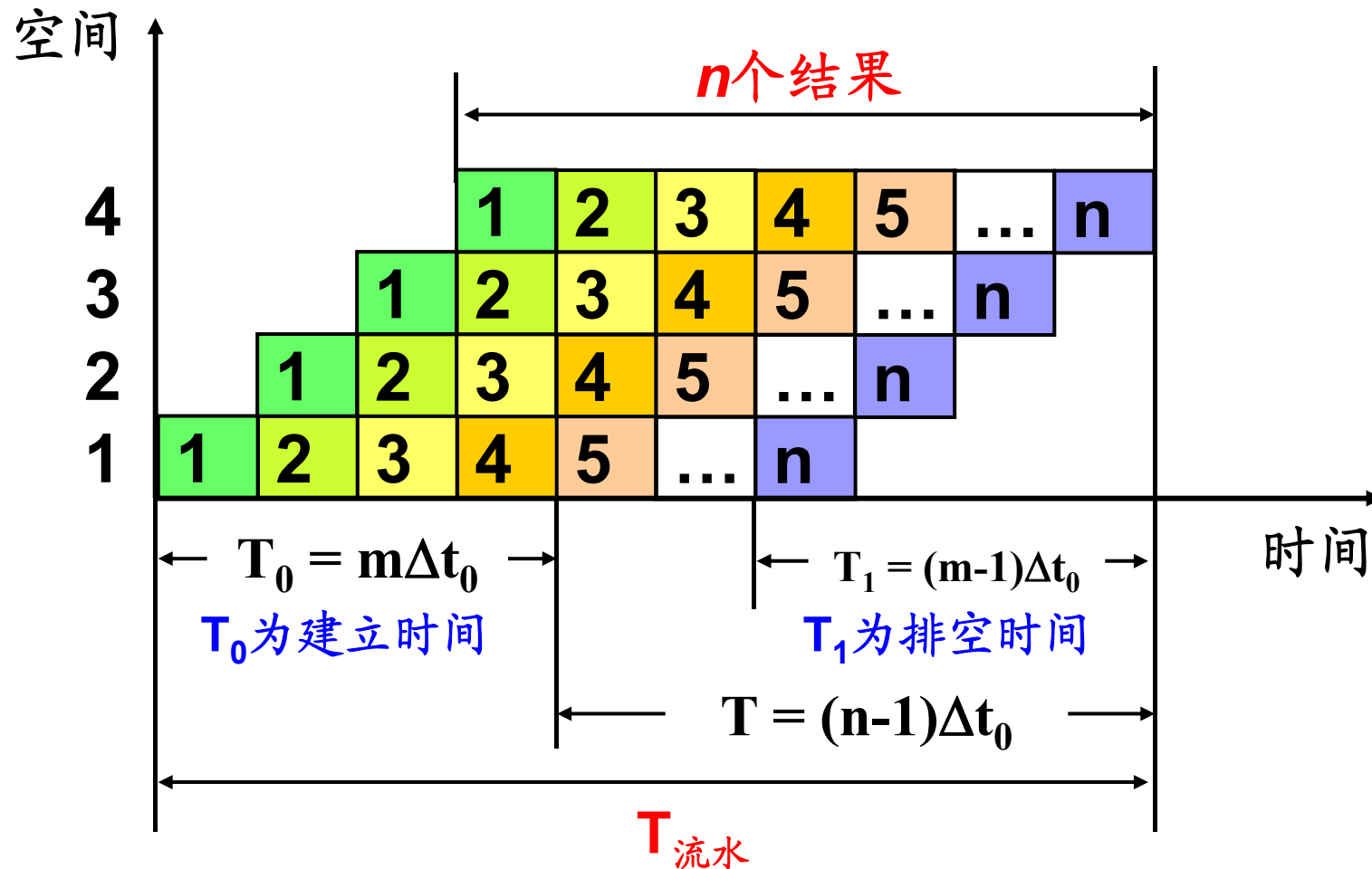
- 若流水线在  $T_{\text{流水}}$  时间内处理了  $n$  条指令，

$$TP = n / T_{\text{流水}}$$

- 通常，  $TP < TP_{\text{max}}$  ， 因为流水线开始时， 总有一段建立时间。
- **建立时间 $T_0$** ： 流水线从开始启动工作到流出第一个结果所需要的时间。
- **排空时间 $T_1$** ： 最后一个任务进入流水线到结果流出所需要的时间。

# 线性流水线吞吐率 — 实际吞吐率

各段的时间相等，均为 $\Delta t_0$





# 线性流水线吞吐率 — 实际吞吐率

- 若流水线各段的时间相等，均为 $\Delta t_0$ ，则：

$$TP = \frac{n}{m\Delta t_0 + (n-1)\Delta t_0} = \frac{1}{\Delta t_0(1 + \frac{m-1}{n})} = \frac{TP_{\max}}{1 + \frac{m-1}{n}}$$

- 若流水线各段的时间不相等，则：

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

# 线性流水线吞吐率 — 实际吞吐率

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

# 加速比

- **定义：**指流水线速度与等效的非流水线速度之比。

$$\text{加速比 } S_p = T_{\text{非流水}} / T_{\text{流水}}$$

对于线性流水线，若流水线为 $m$ 段，每段时间均为 $\Delta t_0$ ，则：

$$T_{\text{非流水}} = nm \Delta t_0, \quad T_{\text{流水}} = m \Delta t_0 + (n-1) \Delta t_0$$

$$S_p = \frac{mn}{m+n-1} = \frac{m}{1 + \frac{m-1}{n}}$$

$$S_{p \max} = \lim_{n \rightarrow \infty} \frac{m}{1 + \frac{m-1}{n}} = m$$

**思考：**流水线的段数愈多愈好？

# 加速比

- 对于线性流水线，若流水线为**m**段，每段时间**不相等**，则：

$$S_p = \frac{n \cdot \sum_{i=1}^m \Delta t_i}{\sum_{i=1}^m \Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \dots, \Delta t_m)}$$

# 效率

- **定义：**指流水线中的设备实际使用时间占整个运行时间之比，也称为流水线的设备时间利用率。
- 由于流水线有通过时间和排空时间，所以在连续完成  $n$  个任务的时间内，各段并不是满负荷地工作。

# 线性流水线效率

- 若各段时间相等，则各段的效率  $\eta_i$  相同：

$$\eta_1 = \eta_2 = \cdots = \eta_m = \frac{n\Delta t}{T_m} = \frac{n}{m+n-1}$$

分母 = 时—空图中  $m$  段和总时间  $T$  所围成的面积  
分子 = 时—空图中  $n$  个任务实际占用的总面积

- 整条流水线的效率为：

$$\eta = \frac{\eta_1 + \eta_2 + \cdots + \eta_m}{m} = \frac{m\eta_1}{m} = \frac{mn\Delta t}{mT_m}$$

或

$$\eta = \frac{n}{m+n-1}$$

与吞吐率的关系为：  $\eta = TP \bullet \Delta t_0$

- 最高效率为：

$$\eta_{\max} = \lim_{n \rightarrow \infty} \frac{n}{m+n-1} = 1$$

当  $n \gg m$  时，  $\eta \approx 1$ 。

# 线性流水线效率

- 若各段时间不相等，则各段的效率  $\eta_i$  不相同
- 整条流水线的效率 = ?

从时一空图上来看，效率实际上就是  $n$  个任务所占用的时空区面积与  $m$  个段所占用的总的时空区的面积之比。

$$\eta = \frac{n \text{ 个任务实际占用的时-空白}}{m \text{ 个段总的时-空区}}$$

- 若各段时间不相等

$$\eta = \frac{n \text{ 个任务实际占用的时-空白}}{m \text{ 个段总的时-空区}} = \frac{n \cdot \sum_{i=1}^m \Delta t_i}{m \cdot \left[ \sum_{i=1}^m \Delta t_i + (n-1) \Delta t_j \right]}$$

## 5.2.2 流水线处理机的主要性能

- **问题：流水线功能段的数量应为多少？**
- 增加流水线段数时，流水线的吞吐率和加速比都能提高。但随着段数的增多，总延迟时间将增加，流水线的价格也会增加。因此要综合考虑各方面的因素，根据性能价格比来选择流水线最佳段数。
- 目前，一般处理机中的流水线段数在**3到12**之间，极少有超过**15**段的流水线。
- 一般把**8**段或超过**8**段的流水线称为超流水线，采用**8**段以上流水线的处理机有时也称为超流水线处理机。



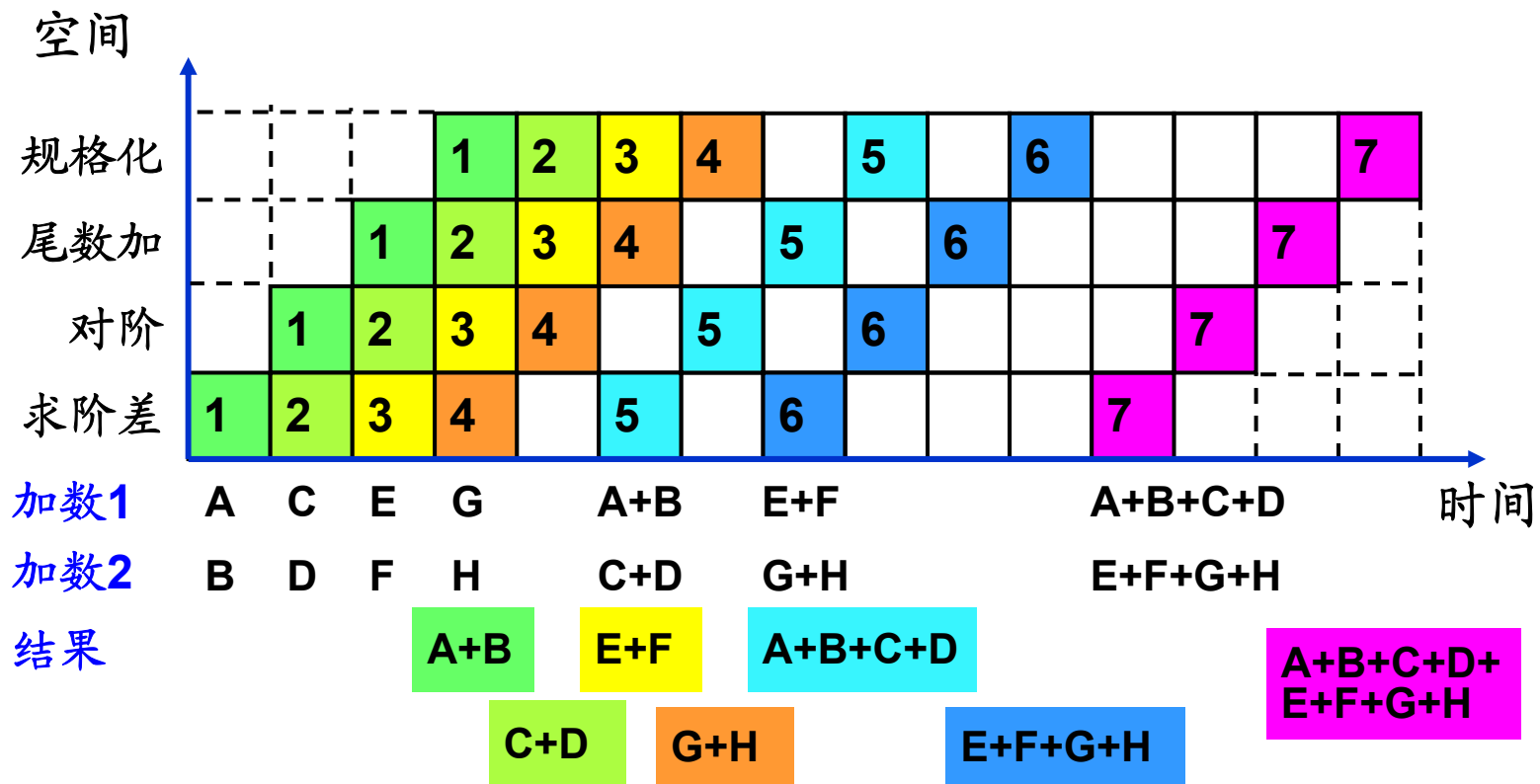
# 流水线性能举例

- **例5-2：**用一条4段浮点加法器流水线求8个浮点数的和： $Z=A+B+C+D+E+F+G+H$ 。计算其吞吐率、加速比和效率。4段浮点加法器分别为：求阶差、对阶、尾数加和规格化。各段处理时间相等，都为 $\Delta t$ 。

# 流水线性能举例

■ 例5-2 解：拆为多个加法，以便连续进行计算：

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



# 流水线性能举例

- **例5-2 解：**进行了**7**次浮点加法，共用了**15**个时钟周期。

$$\text{吞吐率 } TP = \frac{n}{T_{\text{流水}}} = \frac{7}{15 \Delta t}$$

$$\text{加速比 } S = \frac{T_{\text{非流水}}}{T_{\text{流水}}} = \frac{7 \times 4 \Delta t}{15 \Delta t} = 1.87$$

$$\text{效率 } E = \frac{\text{7个加法的时空区}}{\text{4个段总的时空区}} = \frac{7 \times 4 \Delta t}{4 \times 15 \Delta t} = 47\%$$

# 流水线性能举例

- 例1：在DLX的非流水实现和基本流水线中，5个功能单元的执行时间分别为：**10/8/10/10/7ns**。流水线额外开销为**1ns**，求相对于非流水指令实现而言，基本DLX流水线的加速比是多少？

- 解：

$$T_{\text{非流水}} = 10 + 8 + 10 + 10 + 7 = 45\text{ns}$$

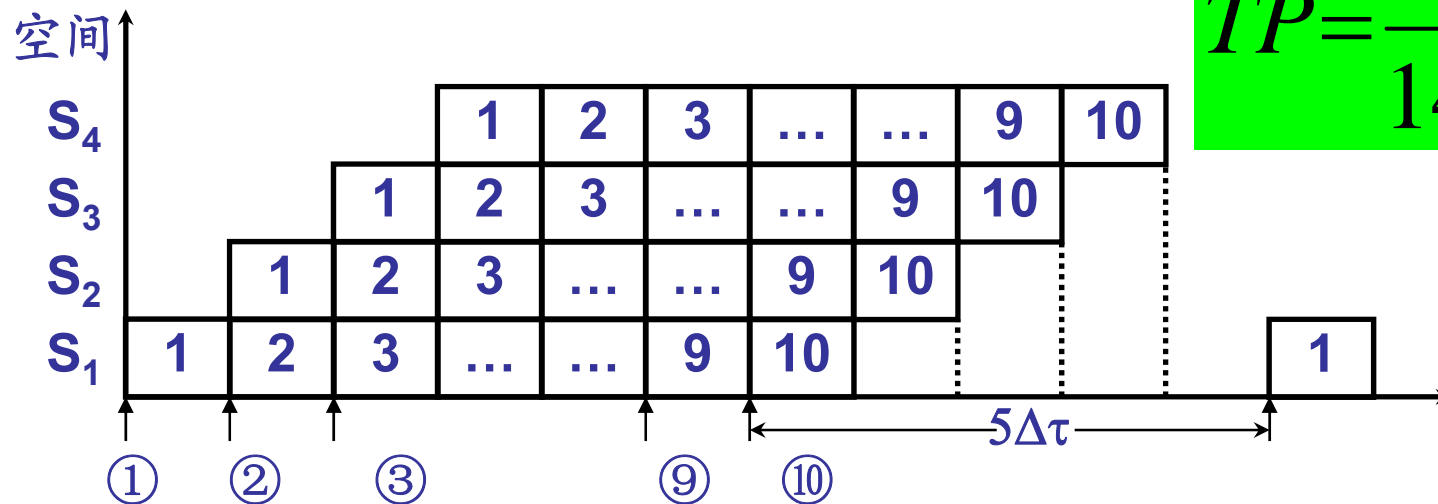
$$T_{\text{流水}} = 10\text{ns} + 1\text{ns} = 11\text{ns}$$

$$\text{加速比} S = 45/11 \approx 4.1$$

# 流水线性能举例

- 例2：流水线由4个功能部件组成，每个功能部件的延迟时间为 $\Delta t$ 。当输入10个数据后，间歇 $5\Delta t$ ，又输入10个数据，如此周期地工作。求此时流水线的吞吐率，并画出时空图。

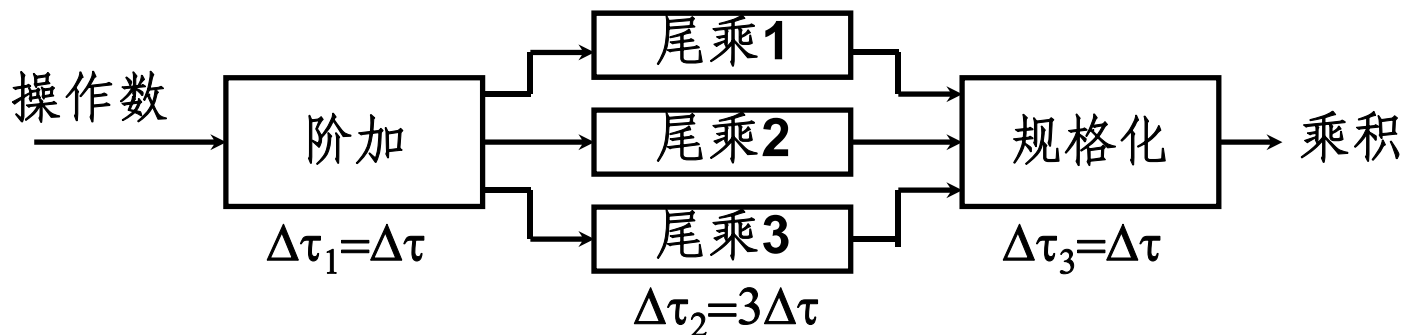
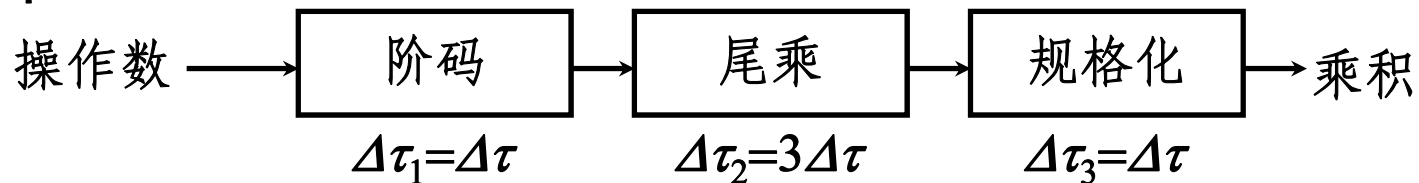
■ 解：



$$TP = \frac{10}{14\Delta t} = \frac{5}{7\Delta t}$$

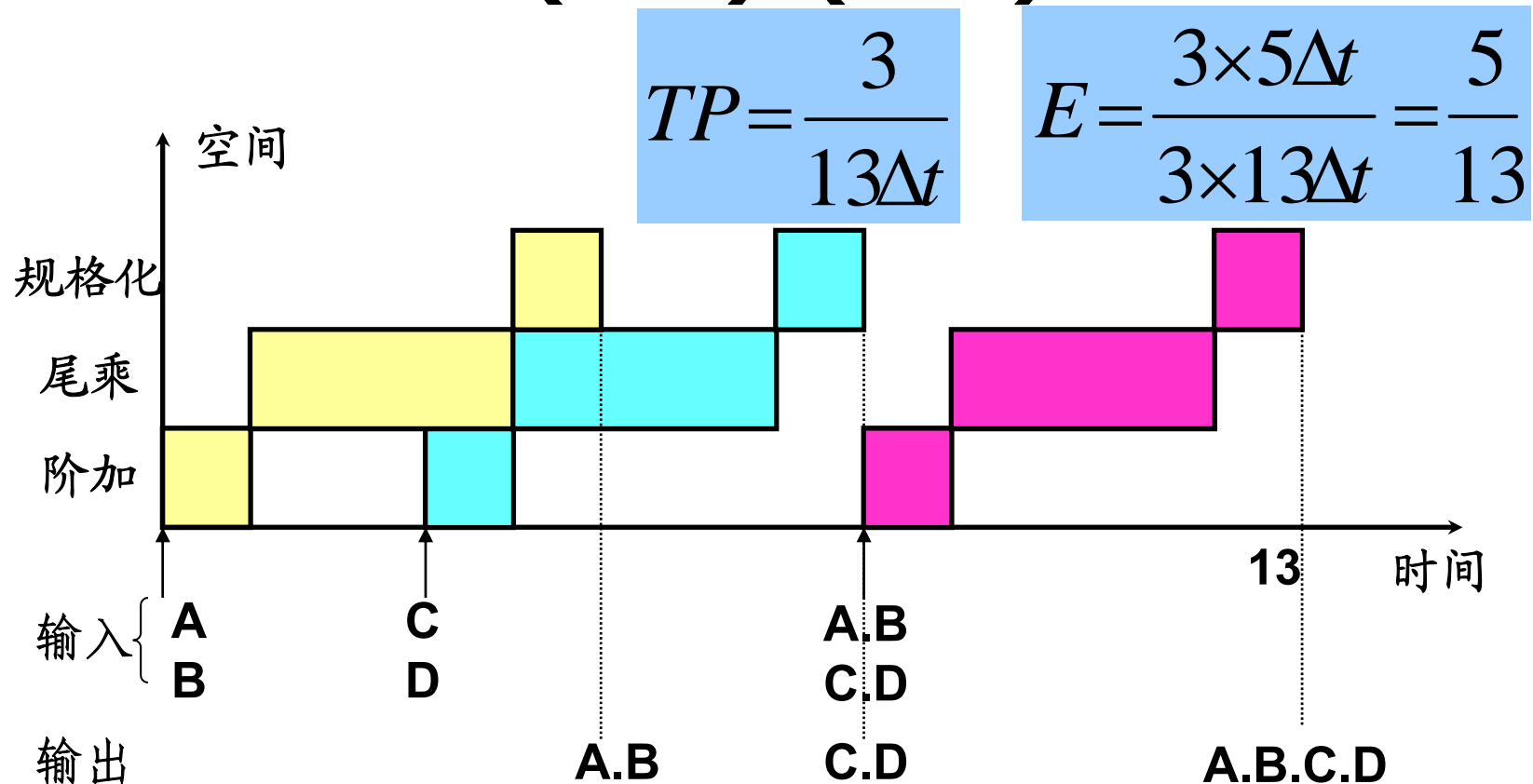
# 流水线性能举例

- 例3：有一个浮点乘流水线如图a所示，其乘积可直接返回输入端或暂存于相应缓冲寄存器中，画出实现 $A*B*C*D$ 的时空图以及输入端的变化，并求出该流水线的吞吐率和效率；当流水线改为图b形式实现同一计算时，求该流水线吞吐率和效率。



# 流水线性能举例

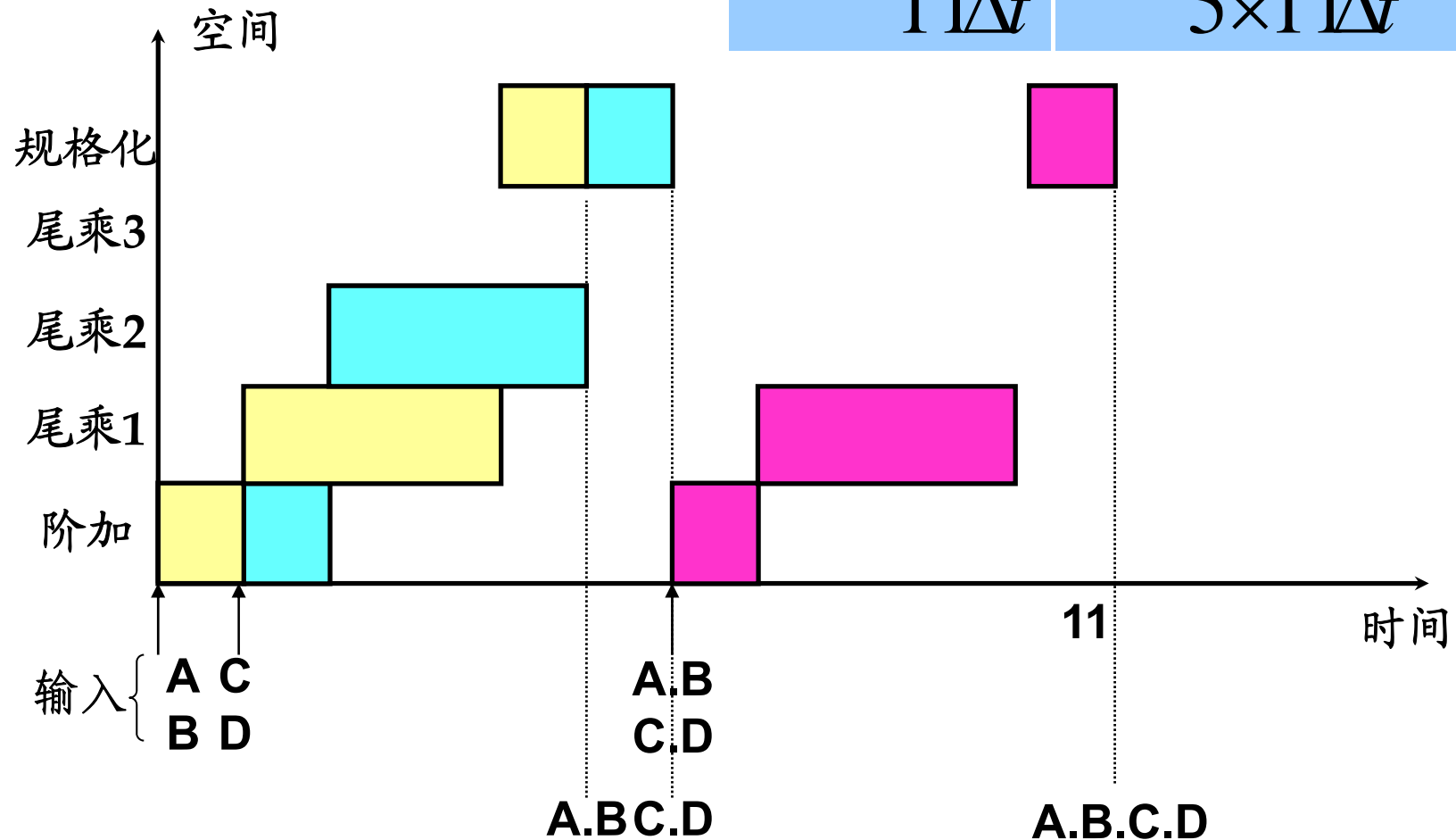
- 解：为减少运算过程操作数相关，将  $A*B*C*D$  改成  $(A*B)*(C*D)$ 。图a



# 流水线性能举例

■ 解：图b

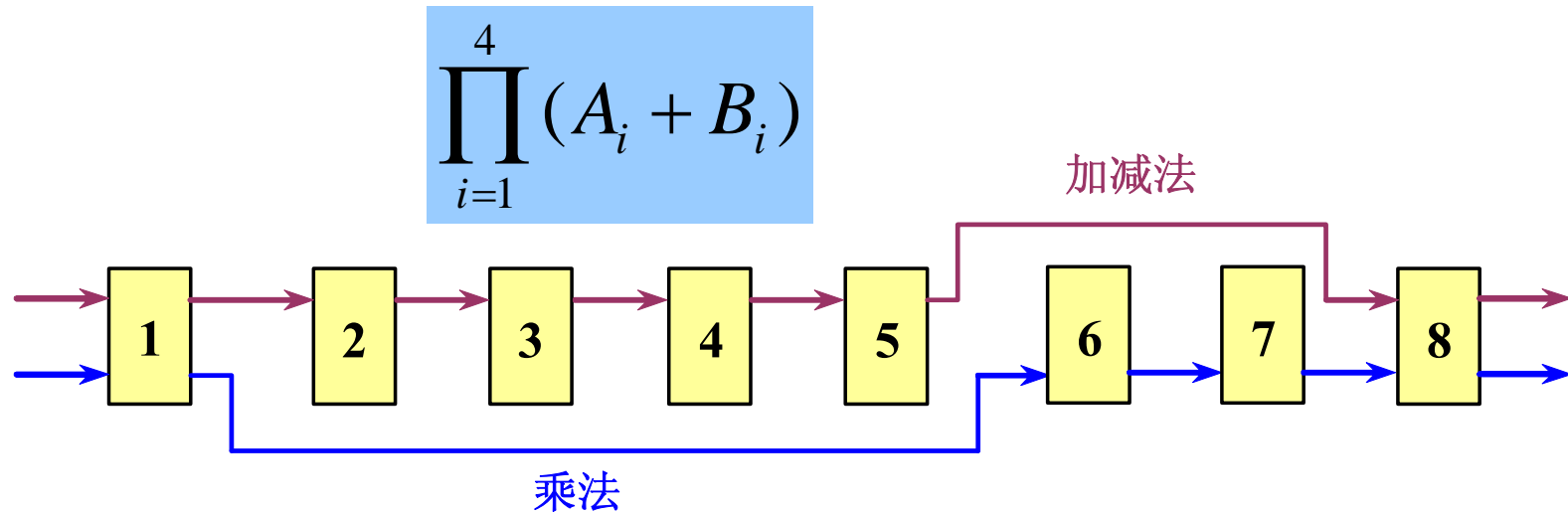
$$TP = \frac{3}{11\Delta t} \quad E = \frac{3 \times 5\Delta t}{5 \times 11\Delta t} = \frac{3}{11}$$





# 流水线性能举例

- 例4：设在下图所示的静态流水线上计算：



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

# 流水线性能举例

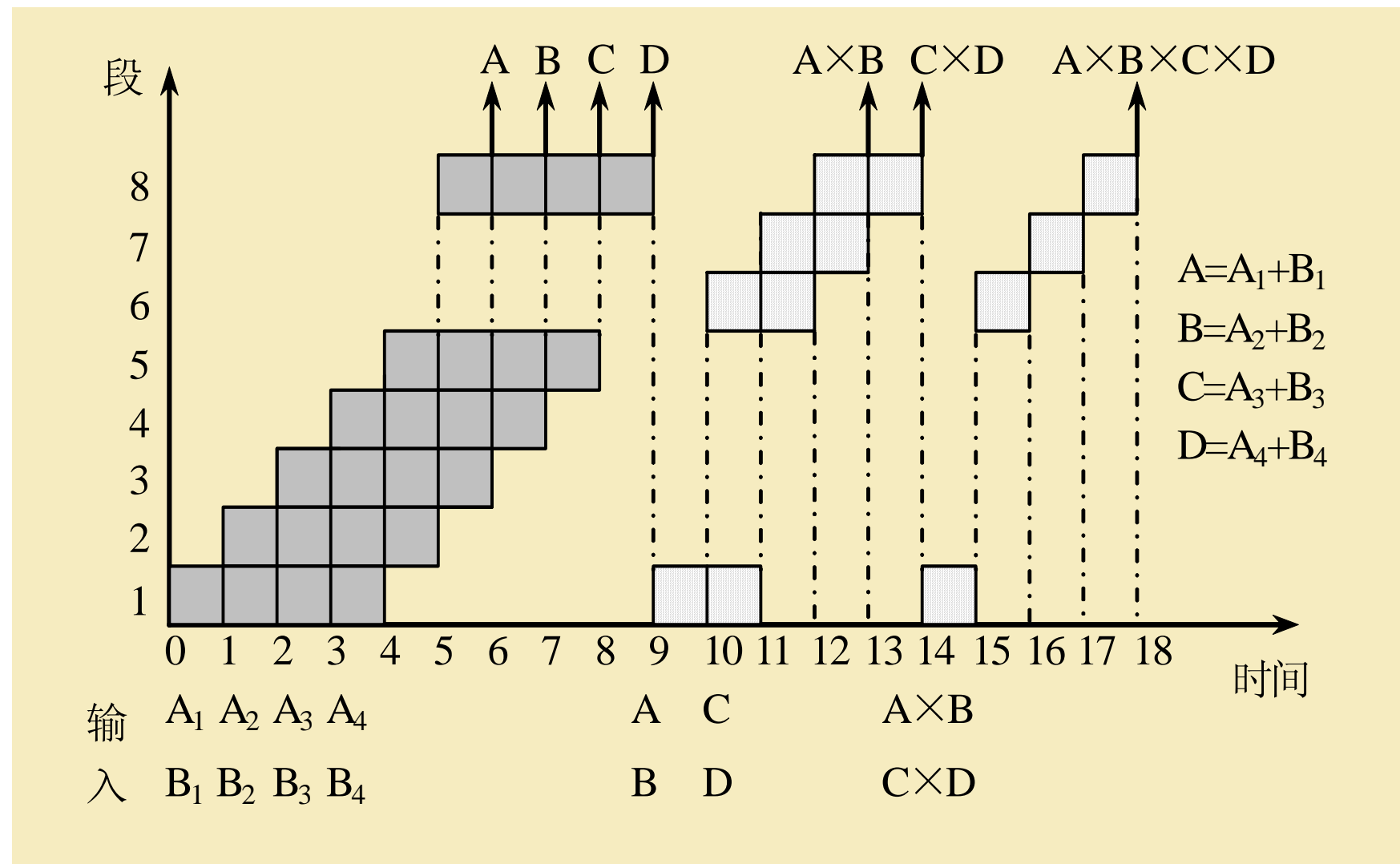
## ■ 例4：解

### (1) 选择适合于流水线工作的算法

- 先计算 $A1+B1$ 、 $A2+B2$ 、 $A3+B3$ 和 $A4+B4$ ;
- 再计算 $(A1+B1) \times (A2+B2)$ 和 $(A3+B3) \times (A4+B4)$ ;
- 然后求总的乘积结果。

### (2) 画出时空图

# 流水线性能举例



# 流水线性能举例

## ■ 例4：解

### （3）计算性能

在**18**个 $\Delta t$ 时间中，给出了**7**个结果。吞吐率为：

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

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

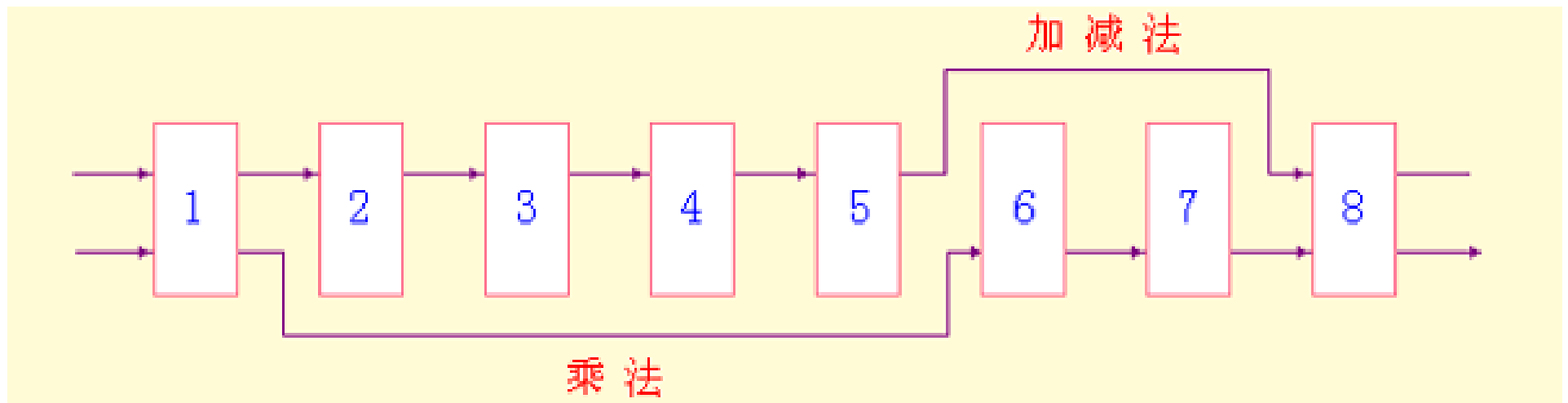
$$S = \frac{36\Delta t}{18\Delta t} = 2$$

流水线的效率：

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

# 流水线性能举例

- 例5：在静态流水线上计算  $\sum A_i B_i$ ， $i=1-4$ 。  
求：吞吐率，加速比，效率。



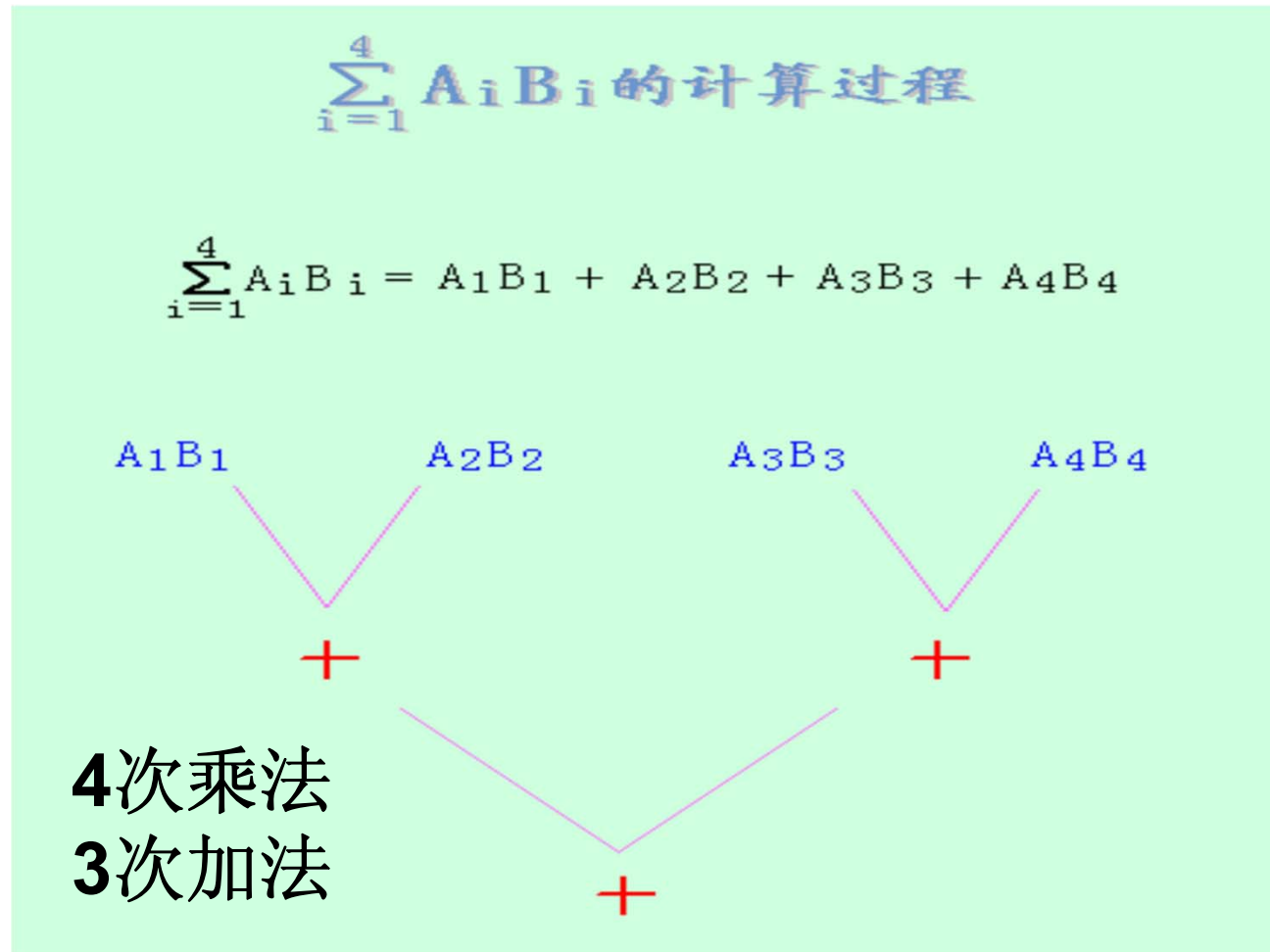
# 流水线性能举例

## ■ 例5：解

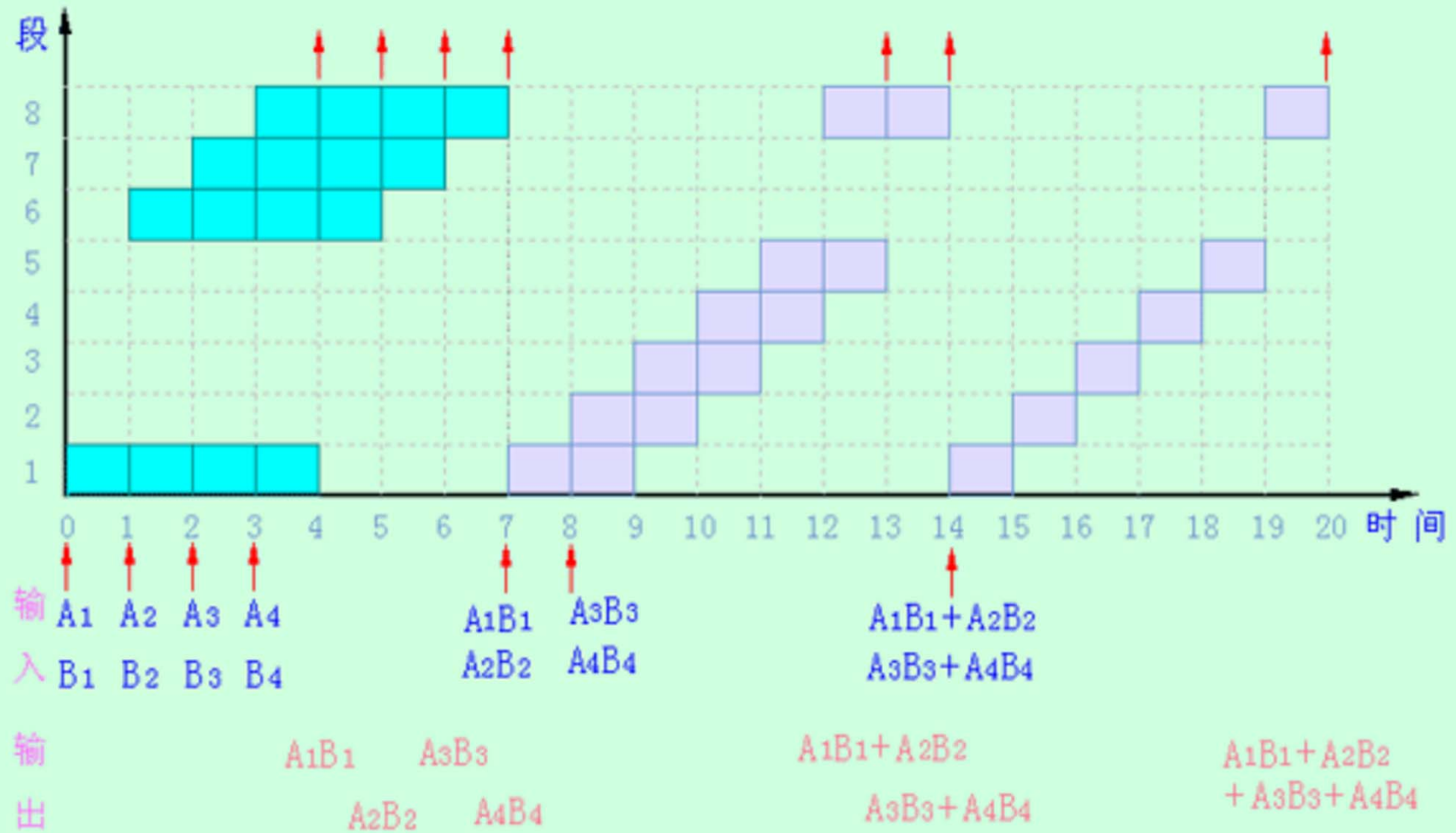
- (1) 选择适合于流水线工作的算法
- (2) 画出时空图
- (3) 计算性能

# 流水线性能举例

## ■ 例5: (1) 选择适合于流水线工作的算法

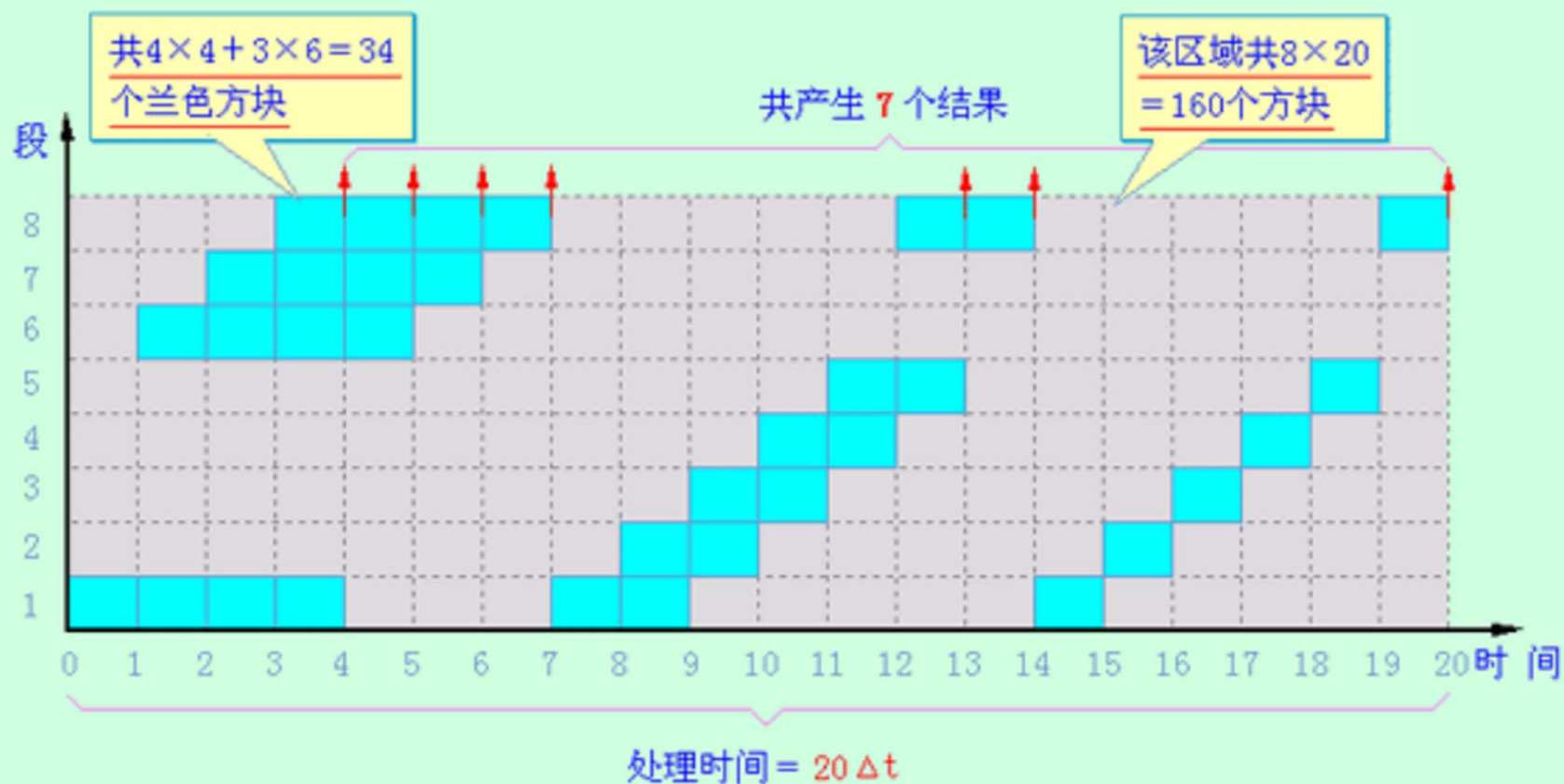


# 时空图





## 性能计算



吞吐率  $TP = 7/20 \cdot \Delta t_0$

加速比  $S = 34 \cdot \Delta t_0 / 20 \cdot \Delta t_0 = 1.7$

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

讨论：为何两题的流水线效率较低？

# 流水线性能举例

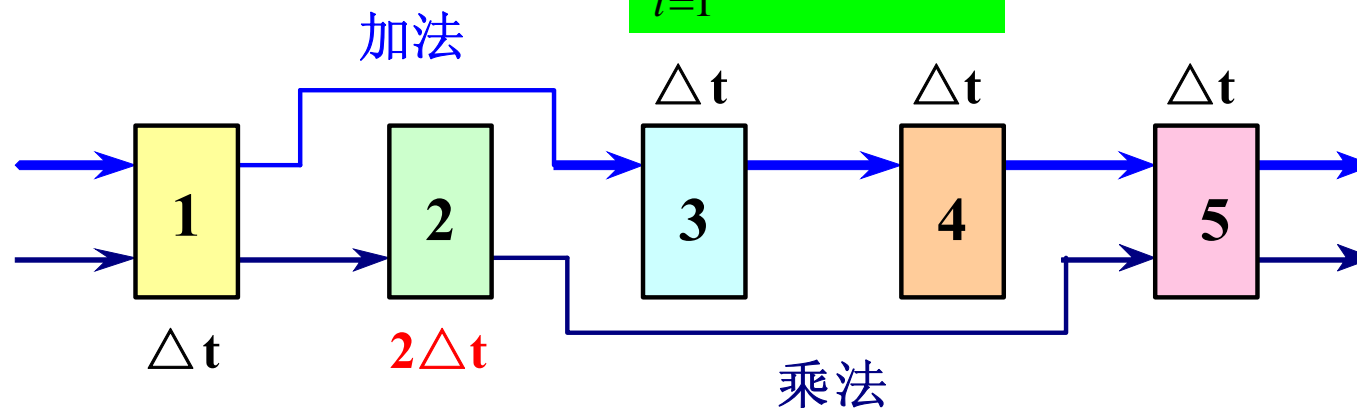
## ■ 讨论：主要原因

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

# 流水线性能举例

- 例6：有一条**动态多功能流水线**由5段组成，加法用1、3、4、5段，乘法用1、2、5段，第2段的时间为 $2\Delta t$ ，其余各段时间均为 $\Delta t$ ，而且流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中。若在该流水线进行以下计算，试计算其吞吐率、加速比和效率。

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



# 流水线性能举例

## ■ 例6：解

(1) 选择适合于流水线工作的

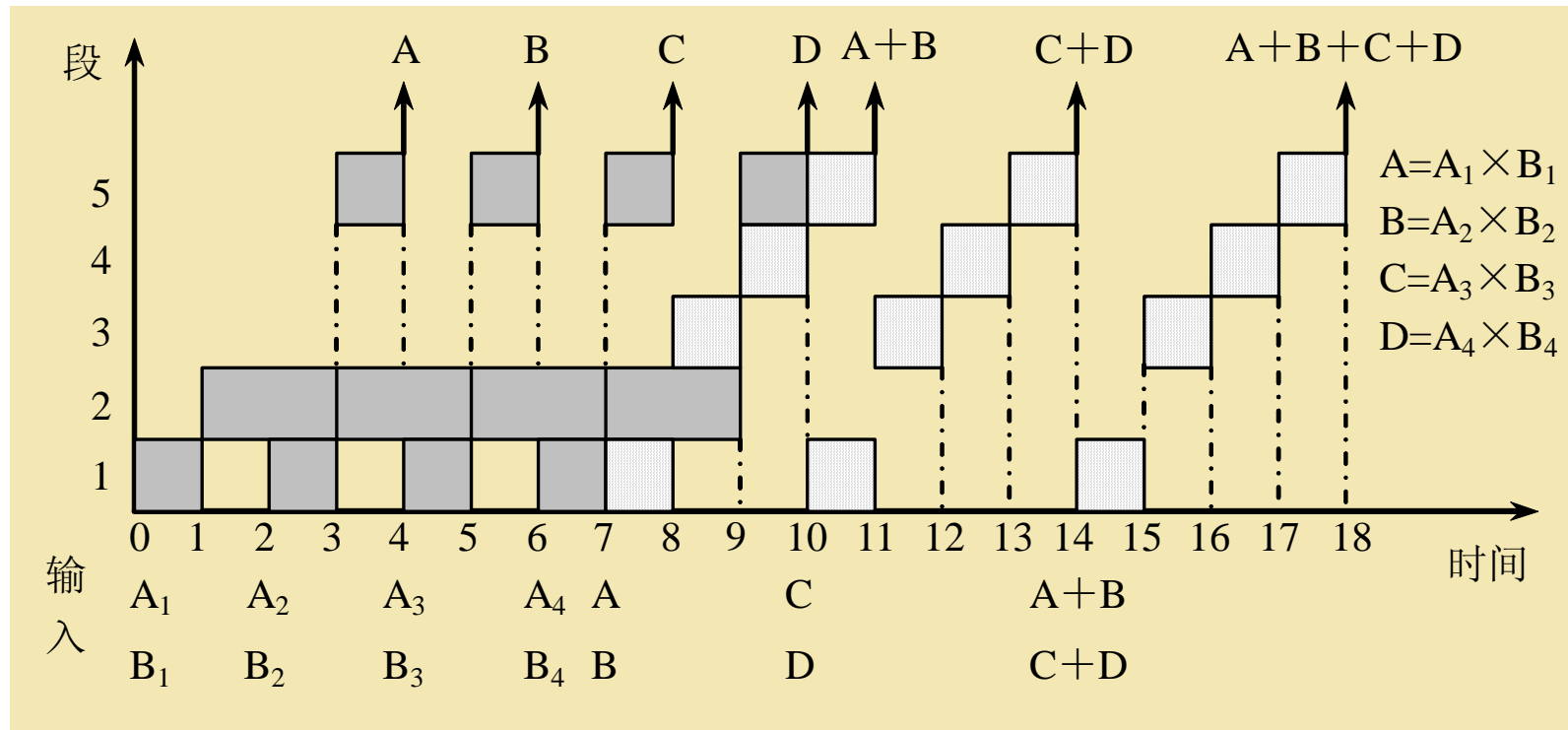
- 应先计算 $A1 \times B1$ 、 $A2 \times B2$ 、 $A3 \times B3$ 和 $A4 \times B4$ ;
- 再计算 $(A1 \times B1) + (A2 \times B2)$  和  $(A3 \times B3) + (A4 \times B4)$ ;
- 然后求总的累加结果。

(2) 画出时空图

(3) 计算性能

# 流水线性能举例

## ■ 例6：解



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

$$S = \frac{28\Delta t}{18\Delta t} \approx 1.56$$

$$E = \frac{4 \times 4 + 3 \times 4}{5 \times 18} \approx 0.31$$

## 5.2.3 流水线调度

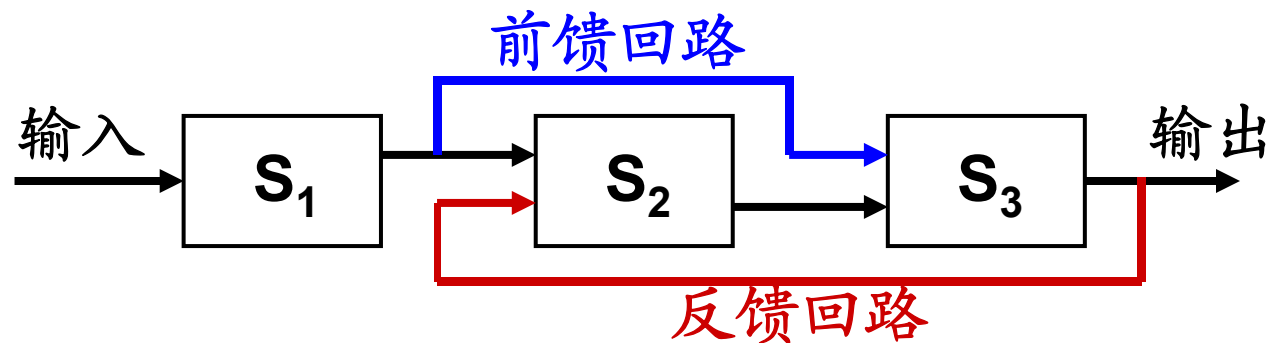
- 由于功能部件一般只有一套，所以难免会发生**资源冲突**。
- **资源冲突**
  - 指几个任务争用一个功能段的现象。
- 因此，如何调度任务，使之合理、高效地利用流水线，使流水线的吞吐率和效率不致下降，是非常关键的。

# 1. 线性流水线的调度

- 线性流水线无反馈，每个任务只通过每个段一次，不会发生冲突，因此只需每隔 $\Delta t$ （或数个 $\Delta t$ ）输入一个任务即可。
- 注意：需解决好其他相关问题！

## 2. 非线性流水线的调度

- 非线性流水线段间有反馈回路，一个任务在流水执行的全过程中，可能会多次通过同一段，或越过某些段。



一种简单的非线性流水线



## 2. 非线性流水线的调度

- 如果每拍向流水线输入任务，将会发生资源冲突。
  - 注意：可能还有其他相关问题！
- 问题：那么究竟间隔几拍向流水线输入任务，才能既不发生功能段使用冲突，又能使流水线有较高的吞吐率呢？
- 这就是流水线调度要解决的问题！

## 2. 非线性流水线的调度

- 在一般情况下，间隔的拍数当然应该越小越好，而且往往不是一个常数。

- 非线性流水线的调度的任务：

找出一个最小的循环周期，按照这个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。

## 2. 非线性流水线的调度

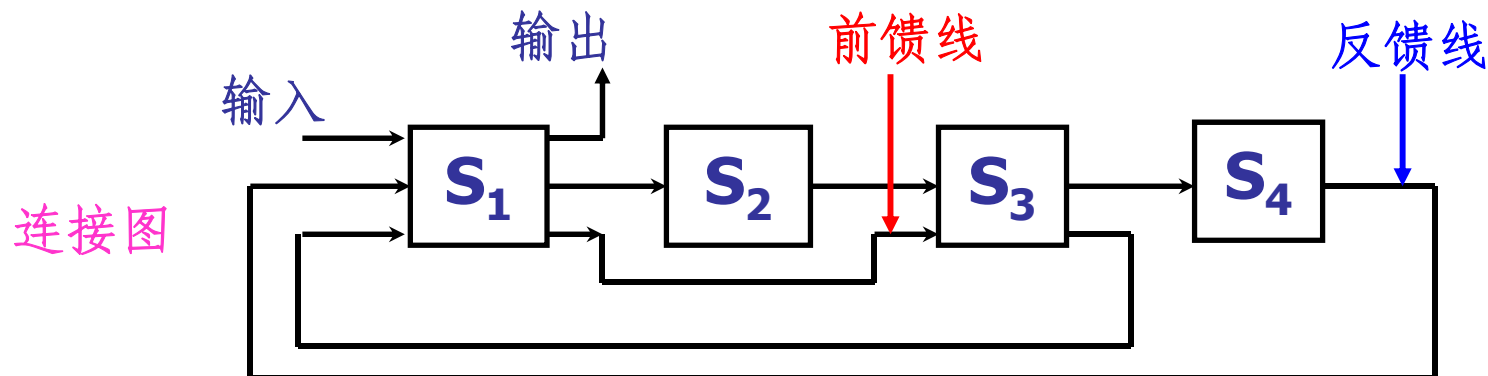
### ■ 二维预约表法(Reservation Table)

- E. S. Davidson 于1971年提出。
- 假设有一个由 **K** 段组成的单功能非线性流水线，每个任务通过流水线需要 **N** 拍。
- 利用类似时—空图的方法，可以得到一个任务使用流水线各段的时间关系表（二维预约表）。
- 如果任务在第 **n** 拍用到第 **K** 段，就在相应第 **n** 列和第 **K** 行的交叉点处用 **✓** 表示。

## 2. 非线性流水线的调度

- 现有一个由4个功能段组成非线性流水线，每个任务通过流水线需要7拍。该流水线的二维预约表如下：

	1	2	3	4	5	6	7
S1	√			√			√
S2		√			√		
S3		√			√		
S4			√				



## 2. 非线性流水线的调度

### ■ 流水线的启动距离:

- 也称为等待时间
- 向流水线连续输入两个任务之间的时间间隔

### ■ 流水线的冲突:

- 几个任务争用同一个流水段

## 2. 非线性流水线的调度

	1	2	3	4	5	6	7	8	9	10	11
S1	X1			X1X2			X1X3 X2	X1		X2X3	X1
S2		X1			X1X2			X2X3	X1		X3
S3		X1			X2	X1		X3	X1X2		
S4			X1			X2			X3	X1	

← 启动周期
← 重复启动周期

启动距离=3时，发生冲突

## 2. 非线性流水线的调度

	1	2	3	4	5	6	7	8	9	10	11
S1	X1		X2	X1	X3	X2	X1	X1X3	X2		X1X3 X2
S2		X1		X2	X1	X3	X2		X1X3		
S3		X1		X2		X1X3		X2	X1	X3	
S4			X1		X2		X3			X1	

← 启动周期
← 重复启动周期

启动距离=2时，发生冲突

## 2. 非线性流水线的调度

- 根据二维预约表可以确定非线性流水线的调度方案。
- 5个步骤：
  - ① 构建延迟禁止向量  $F$
  - ② 构建初始冲突向量  $C$
  - ③ 计算流水线的新的冲突向量
  - ④ 构造用冲突向量表示的流水线状态图
  - ⑤ 确定调度方案



## 2. 非线性流水线的调度

- **例1：**现有一个由 5 个功能段组成的单功能非线性流水线，每个任务通过流水线需要 9 拍。该流水线的二维预约表如下：

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

## 2. 非线性流水线的调度

- ① 根据预约表可以得出一个任务使用各功能段所需间隔的拍数，构建延迟禁止向量  $F$ 。
  - 间隔这些拍数就会产生争用或冲突。引起流水线冲突的启动距离为禁止启动距离。
    - ◆ 例如：1段为8，2段有1、5、6三种
  - 将流水线中所有各功能段对一个任务流过时会产生争用的节拍间隔数汇集在一起，构成延迟禁止向量  $F$ 。
  - 上述例子的  $F=\{1, 5, 6, 8\}$ ，这些间隔拍数应当禁止使用。

## 2. 非线性流水线的调度

- ② 将禁止向量(或禁止表)转换成 $m$ 位的、用二进制表示的初始冲突向量  $C$ 。

- Collision vector

- 冲突向量  $C$  ( $C_m C_{m-1} \dots C_i \dots C_1$ ) 的第  $i$  位表示与“当时”相隔  $i$  拍给流水线送入后续任务，是否会发生功能段冲突

- ◆ 若  $C_i = 1$  间隔  $i$  拍会产生冲突

- ◆ 若  $C_i = 0$  间隔  $i$  拍不会产生冲突

- $m = ?$

$m = \text{禁止向量 } F \text{ 中的最大值}$

## 2. 非线性流水线的调度

- 根据延迟禁止表  $F = \{1, 5, 6, 8\}$ ，可以得到任务刚进入流水线时的初始冲突向量：

$$C = (10110001)$$

- 其中  $C_2$ 、 $C_3$ 、 $C_4$ 、 $C_7$  为 0
- 所以，第 2 个任务可以距第 1 个任务 2、3、4 或 7 拍流入流水线，而不会与第 1 个任务冲突。

## 2. 非线性流水线的调度

### ■ ③ 计算流水线的新的冲突向量

- 当第 2 个任务进入流水线后，应当产生新的流水冲突向量，以便决定第 3 个任务相隔多少拍流入流水线，才不会与前面的第 1 个和第 2 个任务争用功能段。
- 随着流水线中的任务每拍向前推进一个功能段，原先禁止后续任务进入流水线的间隔拍数应相应地减1。

## 2. 非线性流水线的调度

### ■ ③ 计算流水线的新的冲突向量（续）

- 这意味着可以将冲突向量放在一个右移位器中，  
每拍右移1位，让左面移出的空位补“0”
- 用移位器中的值与初始冲突向量作“按位或”运算，得到一个新的冲突向量
- 随着任务在流水线中向前推进，会不断形成任务当时的冲突向量
- 如此重复，这样的操作共进行  $n$  次

## 2. 非线性流水线的调度

例1:

■ 初始冲突向量  $C1 = (10110001)$

如果选择第 2 个任务间隔 2 拍时流入流水线，  
则：

■ 第 1 个任务的初始冲突向量右移 2 位，成为  
 $C1 = (00101100)$

■ 第 2 个任务刚流入流水线，其初始冲突向量  
 $C2 = (10110001)$

## 2. 非线性流水线的调度

例1（续）：

- 则流水线**新的冲突向量C**就应当是第 1 个任务**当前**的冲突向量与第 2 个任务**初始**的冲突向量的**按位“或”**

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & (C2) \\ \vee & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & (C1) \\ \hline & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & (C) \end{array}$$

若让第 3 个任务流入流水线后，即不与第 1 个任务发生功能段冲突，也不与第 2 个任务发生功能段冲突，则应与第 2 个任务间隔 2 拍或 7 拍。



## 2. 非线性流水线的调度

如果选择第 3 个任务间隔 2 拍时流入流水线，则：

- 第 1 个任务的冲突向量 **(00101100)** 继续右移 2 位，成为 **C1 = (00001011)**
- 第 2 个任务的冲突向量 **(10110001)** 右移 2 位，成为 **C2 = (00101100)**
- 第 3 个任务刚流入流水线，其初始冲突向量 **C3 = (10110001)**

## 2. 非线性流水线的调度

- 流水线新的冲突向量C就应当是第 1 个任务当前的冲突向量与第 2 个任务的当前冲突向量和第 3 个任务初始的冲突向量的按位“或”

	1	0	1	1	0	0	0	1	(C3)
	0	0	1	0	1	1	0	0	(C2)
∨	0	0	0	0	1	0	1	1	(C1)
<hr/>									
	1	0	1	1	1	1	1	1	(C)

## 2. 非线性流水线的调度

- 也可以这样计算流水线新的冲突向量C：将流水线的当前的冲突向量  $C = (1011101)$  右移两位，成为  $C = (0010111)$ ，然后与第3个任务初始的冲突向量  $C3 = (10110001)$  的按位“或”

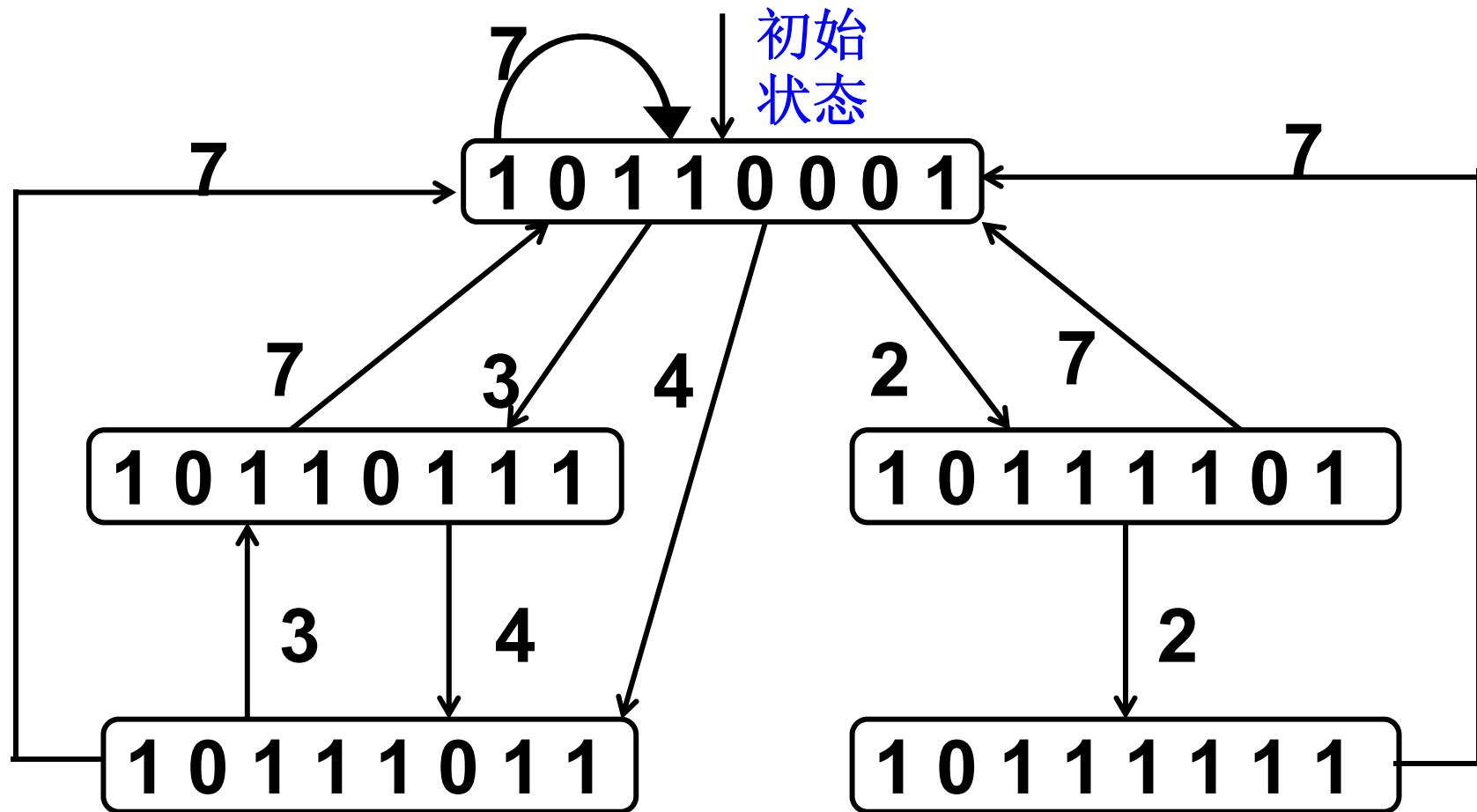
$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & (C3) \\ \vee & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & (C) \\ \hline & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & (C) \end{array}$$

## 2. 非线性流水线的调度

- 按照这样的思路，选择各种可能的拍数输入信任任务，又可以产生新的冲突向量，一直到不再产生不同的冲突向量为止，这样就考虑了所有的可能性。

## 2. 非线性流水线的调度

### ■ ④ 构造用冲突向量表示的流水线状态图



## 2. 非线性流水线的调度

### ■ ⑤ 确定调度方案

- 从状态图中的初始状态出发，找到一个间隔拍数呈周期性重复的方案，并按此方案进行调度，就不会发生功能段冲突
- 显然，可以找到多个调度方案

## 2. 非线性流水线的调度

- 问题：但哪一种调度方案最佳呢？
- 所谓最佳是指：
  - 流水线吞吐率最高
  - 控制简单
- 在这些调度方案中，找出平均延迟最短的调度方案，就是流水线的最佳调度方案。

## 2. 非线性流水线的调度

- 根据上述状态图，可以找出所有的调度方案，并计算出每种方案的平均间隔拍数

调度方案	平均间隔拍数	调度方案	平均间隔拍数
(2, 2, 7)	3.67	(3, 7)	5.00
(2, 7)	4.50	(4, 3, 7)	4.67
(3, 4)	3.50	(4, 7)	5.50
(4, 3)	3.50	(7)	7.00
(3, 4, 7)	4.67	...	...



## 2. 非线性流水线的调度

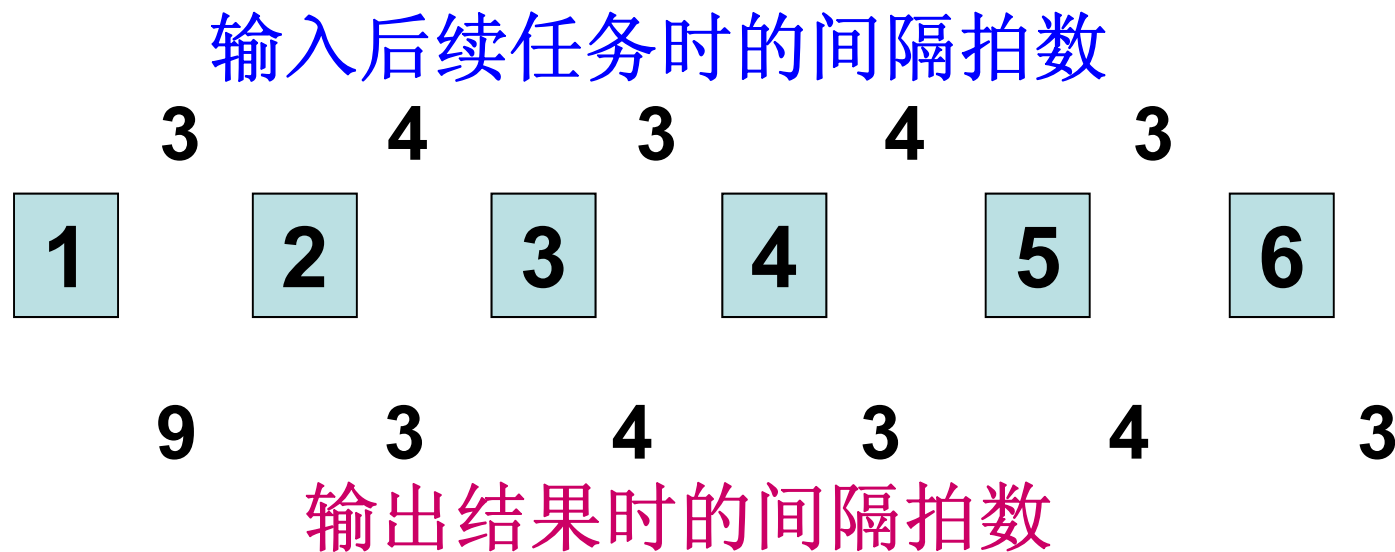
- 可以看出，**(3, 4) / (4, 3)** 调度方案平均间隔拍数最小，吞吐率最高，是最佳调度方案
- 为了简化控制，也可以采用**等间隔调度**，但常常会使吞吐率和效率下降
- 本例中只有一个等间隔调度方案：每隔**7**拍输入一个任务

## 2. 非线性流水线的调度

- **(3, 4) / (4, 3) 哪种调度方案更好?**
- **需要具体分析!**
- **例：连续输入6个任务，分别计算采用 (3, 4) / (4, 3) 调度方案时的吞吐率和效率**

## 2. 非线性流水线的调度

### ■ 采用 (3, 4) 时



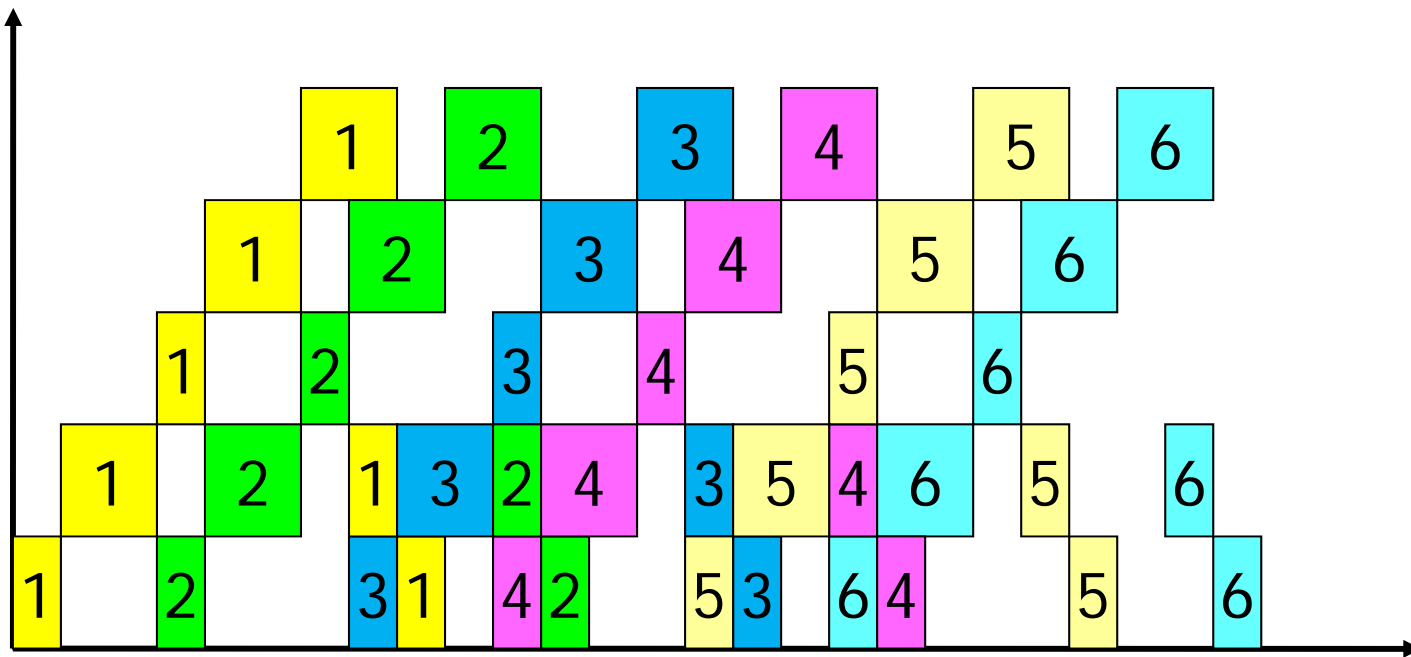
$$TP_{\max} = 1/3.5\Delta t \quad Sp = ?$$

$$TP = 6/(9+3+4+3+4+3) = 6/26\Delta t$$

$$\text{效率} = 6 \times 10 \Delta t / 5 \times 26 \Delta t = 46\%$$

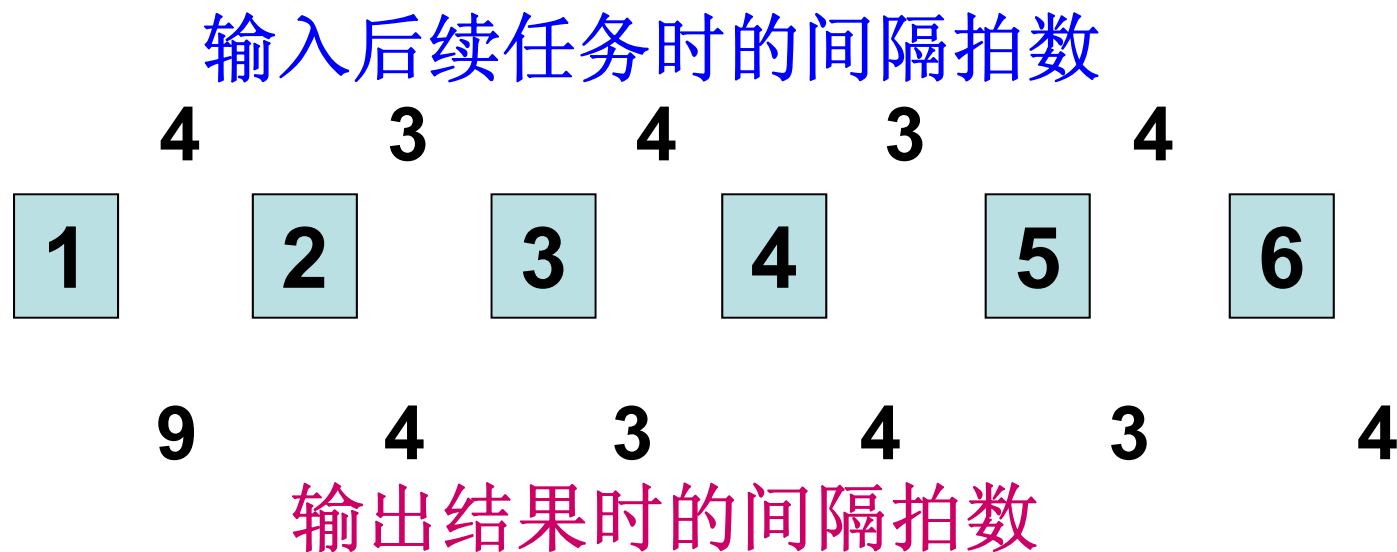
## 2. 非线性流水线的调度

### ■ 采用 (3, 4) 时



## 2. 非线性流水线的调度

### ■ 采用 (4, 3) 时



$$TP_{\max} = 1/3.5\Delta t$$

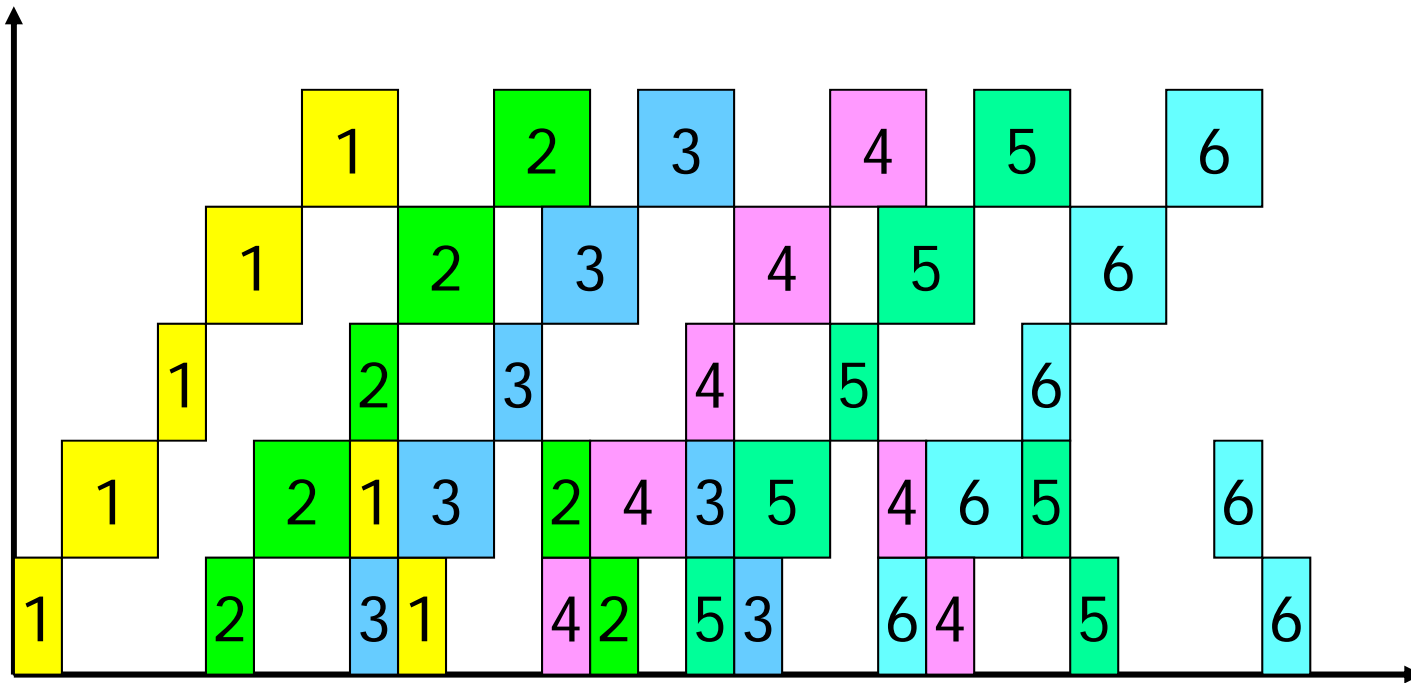
$$Sp = ?$$

$$TP = 6 / (9 + 4 + 3 + 4 + 3 + 4) = 6 / 27 \Delta t$$

$$\text{效率} = 6 \times 10 \Delta t / 5 \times 27 \Delta t = 44\%$$

## 2. 非线性流水线的调度

### ■ 采用 (4, 3) 时



## 2. 非线性流水线的调度

- 例2：一条有 4 个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

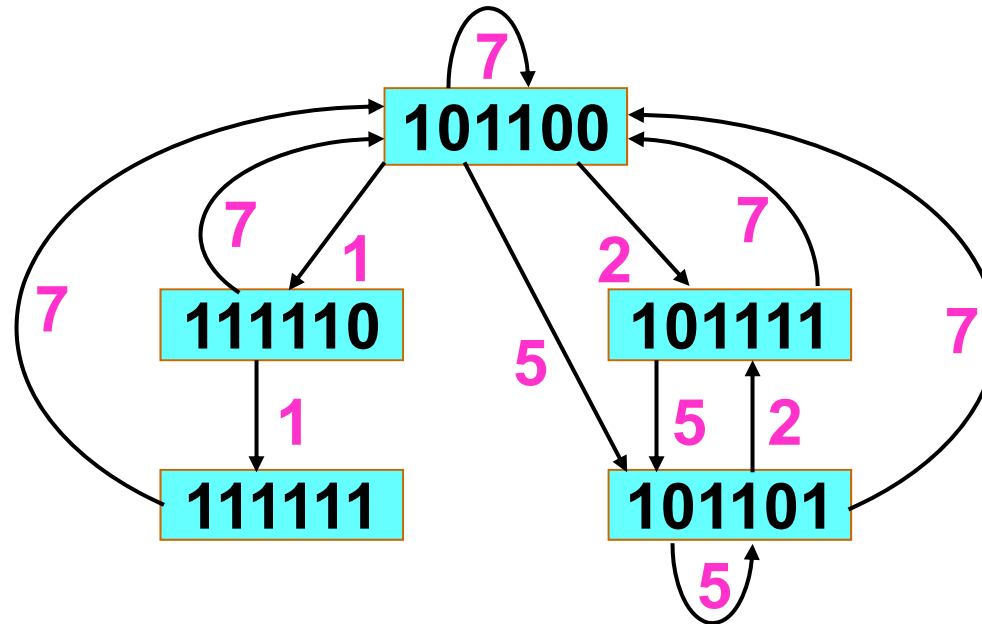
	1	2	3	4	5	6	7
S1	√			√			√
S2		√			√		
S3		√				√	
S4			√				

- (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) 求流水线的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。

## 2. 非线性流水线的调度

例2 解:

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





## 2. 非线性流水线的调度

例2 解:

- 可能的调度方案:
- 平均延迟最小的启动循环被称为**最小启动循环**。
- 最小启动循环为 **(1, 1, 7)**。
- 最小平均启动距离为 **3**。
- 启动距离**最小的恒定循环**是 **(5)**。

调度方案	平均延迟
(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
(7)	7
(5, 2, 7)	4.7

## 2. 非线性流水线的调度

例2 解:

最小启动循环 (1, 1, 7) 的流水线预约表

	1	2	3	4	5	6	7	8	9	10	11	12
S1	✓	▲	▬	✓	▲	▬	✓	▲	▬	▼		
S2		✓	▲	▬	✓	▲	▬				▼	
S3		✓	▲	▬		✓	▲	▬			▼	
S4			✓	▲	▬							▼

### 3. 多功能非线性流水线的调度

- 可以利用上述单功能流水线的调度的基本思想和方法，解决多功能流水线的调度。
- 方法：
  - 将每种功能的预约表都叠加在一起，然后构造交叉冲突向量，以反映多功能动态流水线的各个后继任务流入流水线时应禁止使用的拍数间隔。

## 5.2.4 流水机器的相关处理和控制机构

### ■ 有关流水线性能的若干问题:

- 流水线并不能减少（而且一般是增加）单条指令的执行时间，但能够提高吞吐率；
- 增加流水线的深度可以提高流水线性能；
- 流水线深度受限于流水线的延迟和额外开销；
- 需要用高速锁存器作为流水线寄存器；

#### ◆ Earle锁存器

- 流水线只有连续流动不断流，才能获得较高效率；

但是，指令之间存在的相关，限制了流水线的性能。

# 流水线中的相关

- **流水线中的相关**是指相邻或相近的两条指令因存在某种关联，它使得指令序列中下一条指令无法按照设计的时钟周期开始执行。

相关	局部性相关	<ul style="list-style-type: none"><li>● 指令相关，主存数相关，通用寄存器组数据相关/基址(编址)相关等；</li><li>● 只影响数条指令，不影响指缓中预取的指令。</li><li>● 同步流动：<b>WR</b> 引起；</li><li>● 异步流动：<b>WW</b>、<b>RW</b> 引起；</li></ul>
	全局性相关	<ul style="list-style-type: none"><li>● 转移、中断等引起；</li><li>● 影响后续指令；</li><li>● 影响指缓中预取的指令；</li><li>● 使流水线断流，使吞吐率和效率下降等。</li></ul>

# 1. 局部相关处理

- **原因：** 由于机器同时解释执行多条指令时，这些指令对同一存贮单元要求“**先写后读**”而造成的。
- **解决方法：2种**
  - 推后读。安排流动顺序。
    - ◆ 同步流动
    - ◆ 异步流动
  - 设置相关专用通路

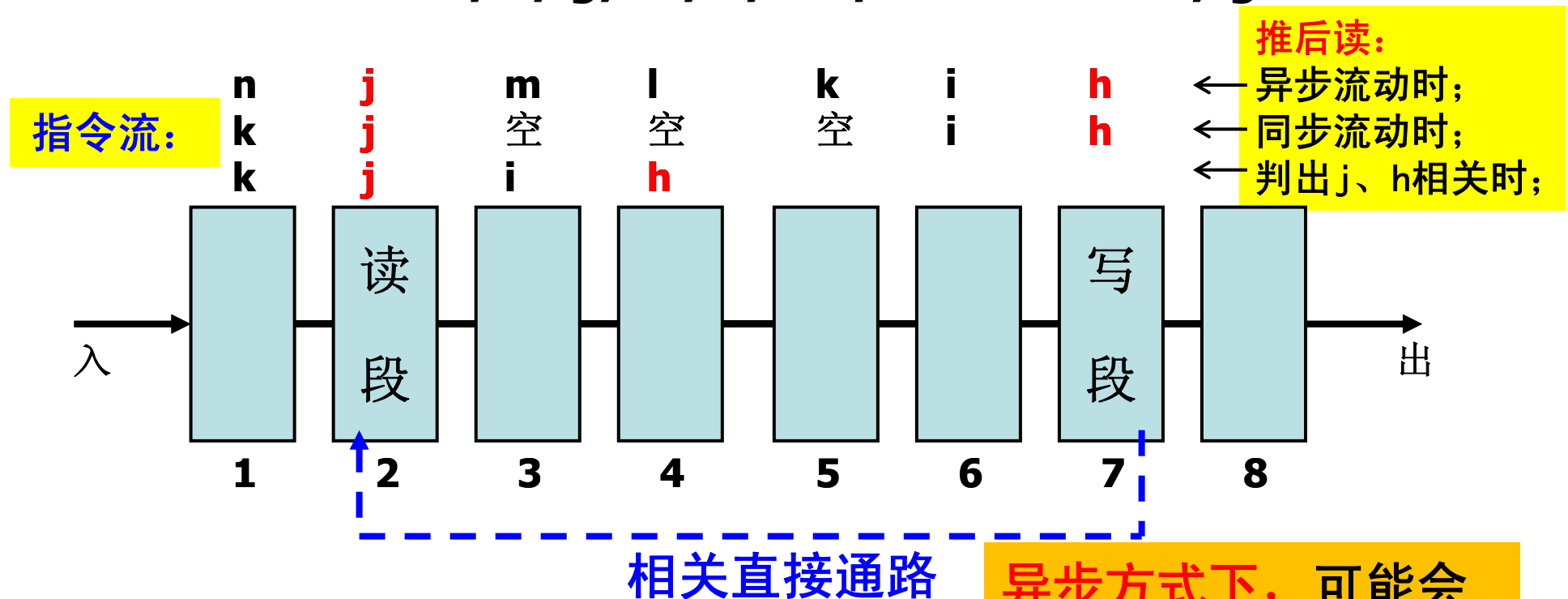
# 1. 局部相关处理

## ■ 任务在流水线中流动顺序的安排和控制

- **同步流动方式**（顺序流动方式）：任务流出流水线的顺序保持与流入流水线的顺序一致
  - ◆ 控制简单，但相关后吞吐率和效率下降
- **异步流动方式**：任务流出流水线的顺序与流入流水线的顺序不一致

# 1. 局部性相关处理

- 8段流水线，第2段为读段，第7段为写段
- 指令流：h, i, j, k, l, m, n。其中h, j 相关。



顺序流动和异步流动

异步方式下，可能会出现“写—写”相关和“先读后写”相关。



# 1. 局部相关处理

## ■ 设置相关专用通路

- 硬件的方法，可以免去对相关单元的写入和重新读出两个子程。
- **缺点：** 由于流水线同时解释执行多条指令，需要在各个功能部件之间为每种局部性相关都设置单独的相关专用通路，硬件耗费大，控制复杂。

## 2. 全局性相关处理

- **原因：** 由分支转移指令引起的相关
  - 影响后续指令
  - 影响指缓中的指令
- **对流水线的影响是全局的，包括：**
  - 指缓中的指令可能要全部作废
  - 流水线断流，使吞吐率和效率下降等

## 2. 全局性相关处理

### ■ 解决方法

- 为了使流水线的吞吐率和效率不致下降太多，**关键是能正确判断相关**，即能正确判断转移分支
- **无条件转移指令的转移分支是已知的**
- **条件转移指令的转移分支是不确定的**，需要等指令流出流水线以后才能确定。为判断条件转移分支，可以采取下述方法：

## 2. 全局性相关

### ■ 条件转移分支判断方法

- 猜测法
- 加快和提前形成条件码
- 采用延迟转移
- 加快短循环程序的处理

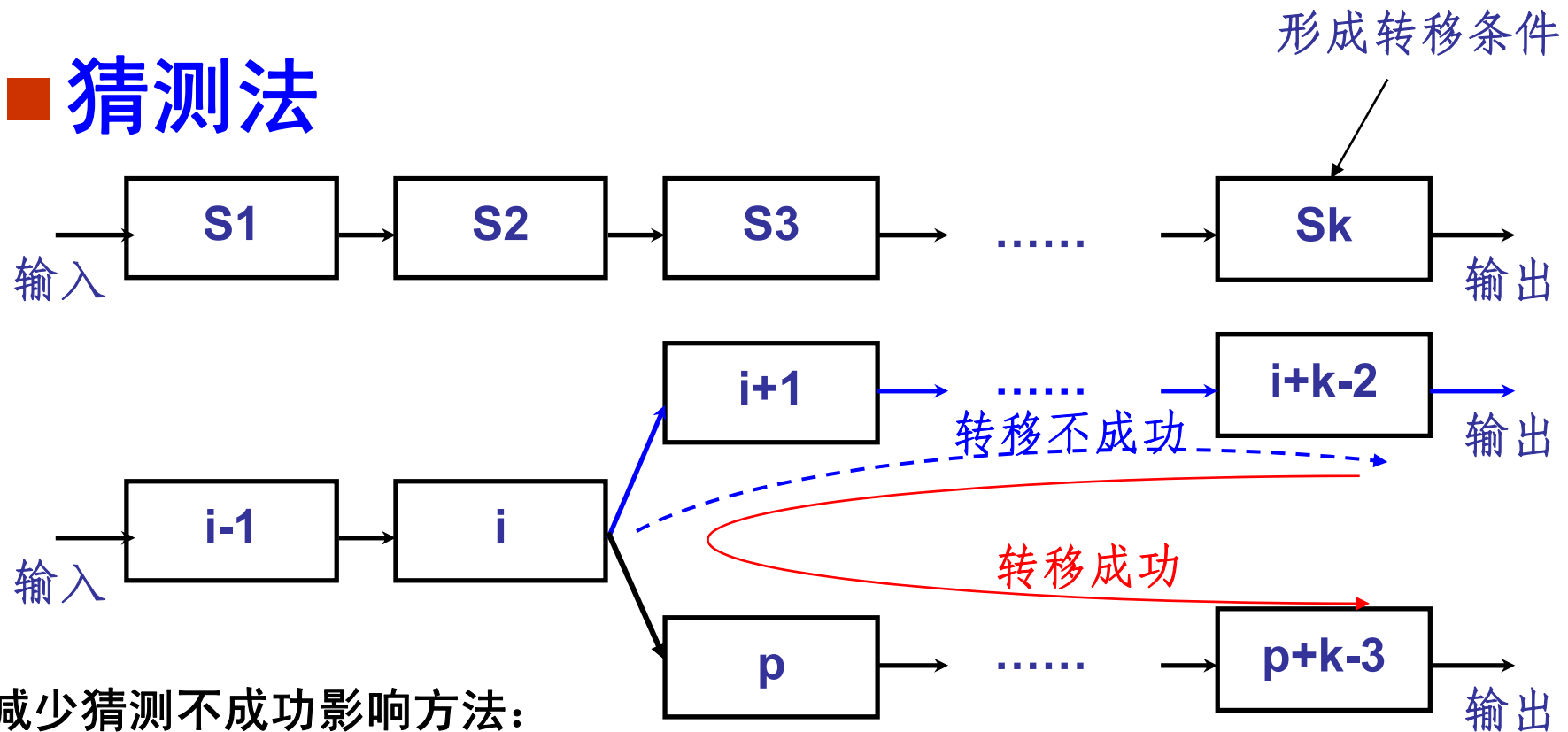
## 2. 全局性相关

### ■ 猜测法

- 选取一个分支继续执行
- 选取哪个分支？
  - ◆ 若知道分支的概率，则选取概率高的分支
  - ◆ 若不知道概率，或两个分支的概率相近时，宜选取不成功的分支

## 2. 全局性相关

### ■ 猜测法



减少猜测不成功影响方法：

- 1) 猜测时只译码、取操作数，在条件未形成之前不执行。
- 2) 执行，但不送结果。
- 3) 执行，送结果，使用后援寄存器。
- 4) 为了尽可能快地回到原分支点，可以预取转移成功分支的头几条指令。

## 2. 全局性相关

### ■ 加快和提前形成条件码

- 加快单条指令条件码形成
- 加快一段程序的条件码形成

### ■ 采用延迟转移

- 用软件方法进行静态指令调度，将转移指令与其前面不相关的一条或多条指令交换位置

## 2. 全局性相关

### ■ 加快短循环程序的处理

- 将短循环程序全部装入指缓
- 对循环程序出口分支的处理，恒猜选环分支，因为循环分支的概率高。一般可以使循环时的流水加快**1/3到3/4**



# 3. 流水机器的中断处理

- 中断会使流水线断流
- 中断的特点：
  - 不经常发生
  - 不可预测
  - 一旦进入中断，可能持续很长时间
- 流水机器中断的处理主要是断点现场的保护与恢复

### 3. 流水机的中断处理

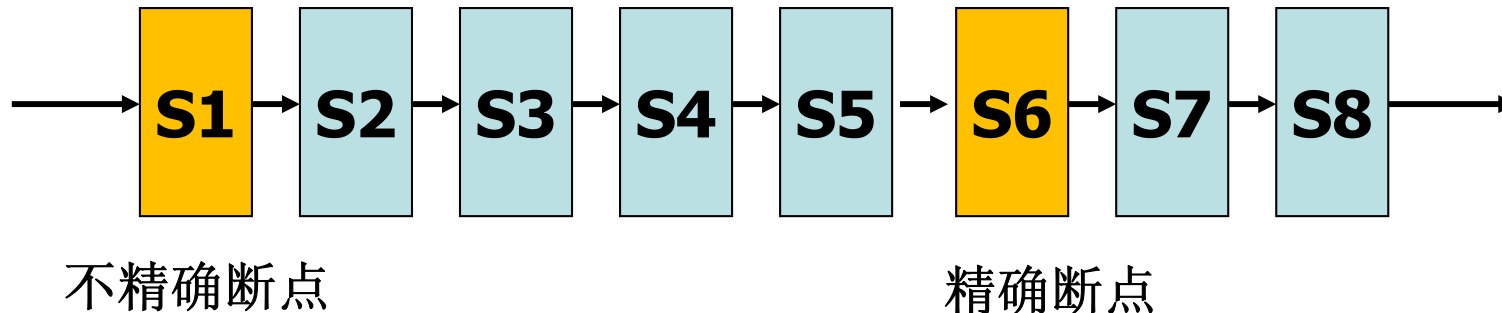
#### ■ 流水线中断有其自己的特点:

- 当在第  $i$  条指令发生中断时，由于流水线同时处理多条指令，中断断点的现场可能已经不是第  $i+1$  条指令尚未开始的地方，而是第  $i+1$ ， $i+2$ ，...等已经被部分解释执行了的地方。
- 对于异步流水线，这些指令有些可能已经流到了指令  $i$  的前面去了

### 3. 流水机的中断处理

■ 可以采取以下解决方法：

- 不精确断点法
- 精确断点法



流水线处理机的中断处理

# 3. 流水机的中断处理

## ■ 不精确断点法

- 当在第 $i$ 条指令发生中断时，不再允许尚未进入流水线的后续指令再进入，但允许已经进入流水线的指令全部执行完毕，然后再转入中断处理程序，这样，断点就不一定是第 $i$ 条，有可能是第 $i+1$ ， $i+2$ ， $\dots$ ， $i+m$ ，即断点是不精确的
- 优点：中断处理简单
- 缺点：调试困难

# 3. 流水机的中断处理

## ■ 精确断点法

- 多数流水机器采用此方法
- 不论指令*i*在流水线的哪一段发生中断，只保护第*i*条指令的现场。*i*之后流入流水线的指令的现场都能恢复
- 此方法需要设置大量的后援寄存器，但这些后援寄存器也是“指令复执”所需要的，不必另外设置

# 学习内容

- 5.1 重叠方式
- 5.2 流水方式
- 5.3 向量的流水处理与向量处理机
- 5.4 指令级高度并行的超级处理机
- 5.5 ARM流水线处理器举例

## 5.3 向量的流水处理与向量处理机

### ■ 向量的特点

- 元素之间无相关 → 可连续流动
- 对元素一般执行相同类型的操作 → 单一功能，很少切换

### ■ 所以向量很适合于用流水处理方式

## 5.3 向量的流水处理与向量处理机

- 一般将向量数据表示与流水处理方式结合在一起，构成**向量流水处理机**，也称其为**向量处理机**
- 向量处理机是解决数值计算问题的一种高性能计算机结构。
- 向量处理机一般都采用流水线结构，有多条流水线并行工作。
- 向量处理机通常属大型或巨型机，也可以用微机加一台向量协处理器组成。



## 5.3 向量的流水处理与向量处理机

- 一般向量计算机中包括有一台高性能标量处理机。
- 必须把要解决的问题转化为向量运算，向量处理机才能充分发挥作用。

# 5.3.1 向量的流水处理

## ■ 向量的处理方式

- 只有选择合适的向量的处理方式，才能充分发挥流水线的效能
- 向量的流水处理所要研究的一个问题就是向量的处理方式

# 5.3.1 向量的流水处理

## ■ 向量的处理方式

- 但向量的处理方式与计算机的系统结构紧密相连，并互相影响
- 不同的处理方式对流水处理机的系统结构、组成提出不同的要求，而系统结构、组成不同的向量处理机也会要求采用不同的向量的处理方式
- 要根据向量运算的特点和向量处理机的类型选择向量的处理方式。

## 5.3.1 向量的流水处理

### ■ 向量的处理方式主要有三种：

- 水平（横向）处理方式
- 垂直（纵向）处理方式
- 分组纵横处理方式

### ■ 下面以 $D=A*(B+C)$ 为例说明向量的处理方式

## 5.3.1 向量的流水处理

### ■ 水平（横向）处理方式

- 即逐个求D向量元素的方法

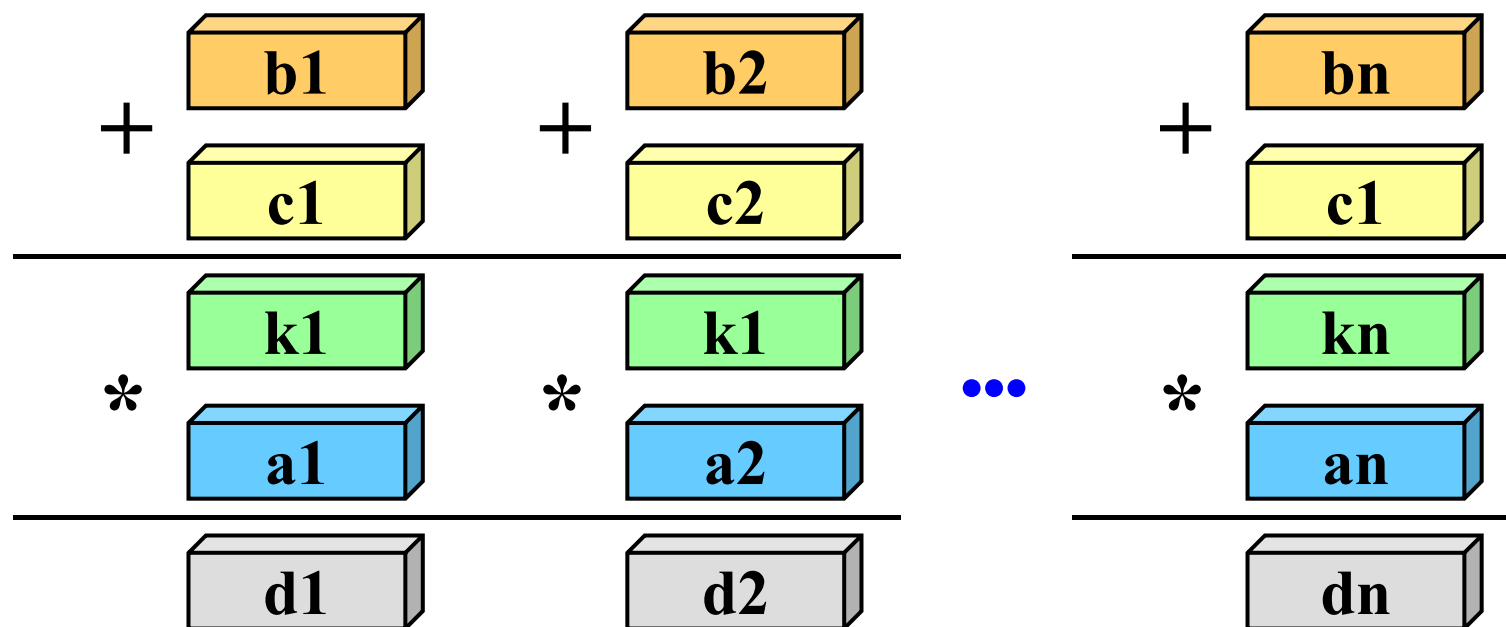
- ◆ 访存，取得三个操作数 $a_i$ ,  $b_i$ ,  $c_i$

- ◆ 先计算  $b_i + c_i \rightarrow k_i$

- ◆ 再计算  $k_i * a_i \rightarrow d_i$

- 标量机上通常采用此方式，使用循环程序实现

## 5.3.1 向量的流水处理



标量机上通常采用此方式，使用循环程序实现。

## 5.3.1 向量的流水处理

### ■ 水平（横向）处理方式

#### ● 2个主要问题:

- ◆ 每个D向量元素至少需两个操作（切换），使流水线难以连续流动
- ◆ 都需要用到k，两条指令之间存在相关（先写后读）→ 不适合流水处理

## 5.3.1 向量的流水处理

### ■ 垂直（纵向）处理方式

- 以向量为单位进行计算

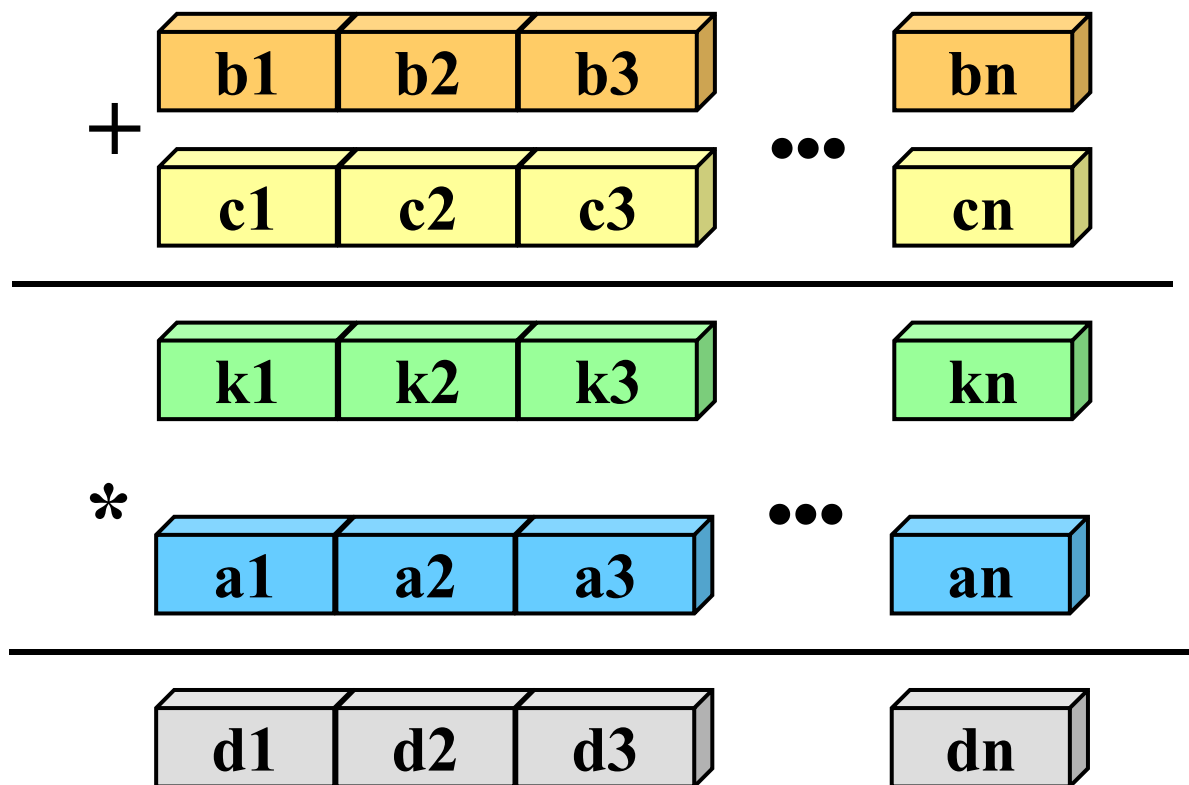
- ◆ 先计算所有的元素  $b_i + c_i \rightarrow k_i$  ( $1 \rightarrow N$ )

- ◆ 再计算所有的元素  $k_i * a_i \rightarrow d_i$  ( $1 \rightarrow N$ )

- 这样元素之间无相关，切换总共只需要一次



## 5.3.1 向量的流水处理



- 只要能为流水线提供连续输入，即可获得高吞吐率。
- 需要存储器具有较快的速度。

## 5.3.1 向量的流水处理

### ■ 垂直（纵向）处理方式

#### ● 问题：

- ◆ 是否能为流水线连续提供输入？
- ◆ 如果主存速度不足，就无法连续提供输入

# 5.3.1 向量的流水处理

## ■ 垂直（纵向）处理方式

### ● 解决方法：

- ◆ 采用多体交叉存贮。这是一种面向存贮器—存贮器型结构的流水线处理机。由于很多通道也要使用主存，要保证连续提供数据很难，因此将主存直接连在流水线输入、输出端的做法并不是最好
- ◆ 设置向量寄存器组。这是一种面向寄存器—寄存器型结构的流水线处理机。将流水线的输入、输出端直接连到大容量向量寄存器组，向量寄存器组与主存之间成组传送

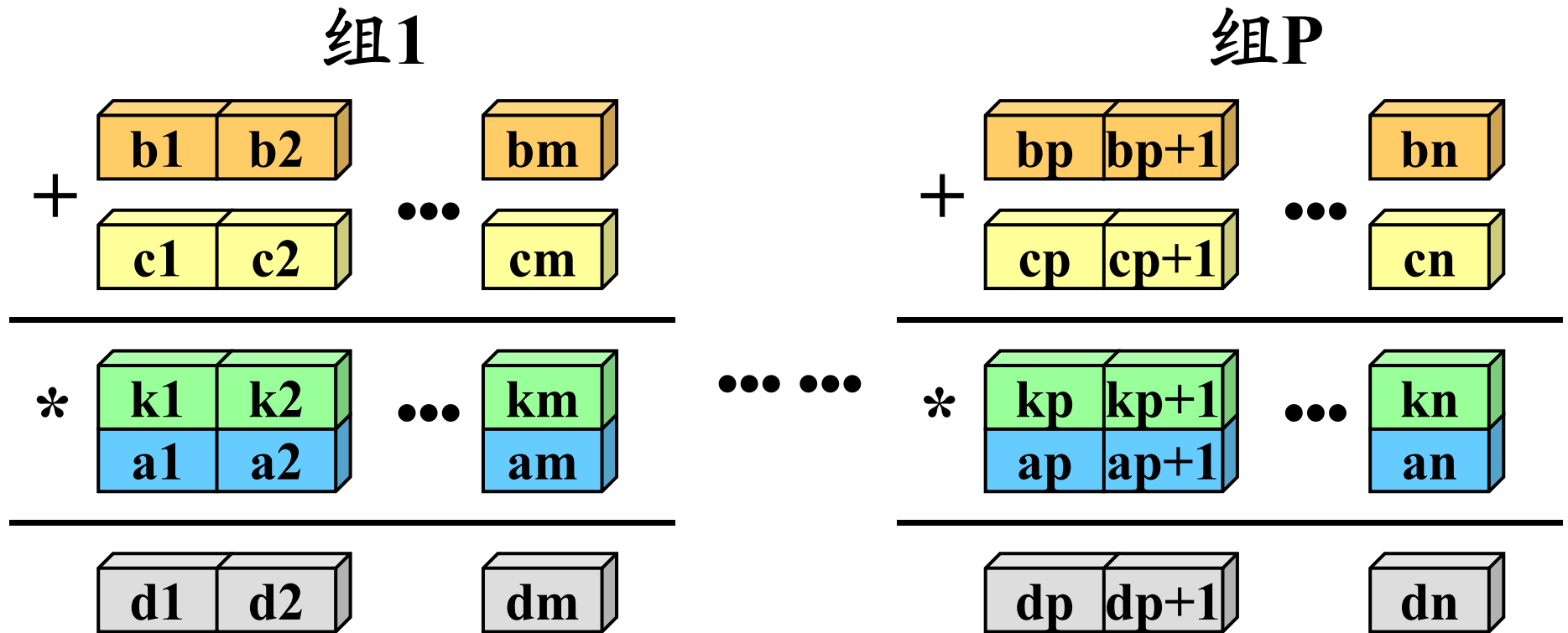
## 5.3.1 向量的流水处理

### ■ 分组（纵横）处理方式

- 如果寄存器装得下整个向量，则采用垂直处理方式；
- 如果向量太长，使得寄存器装不下整个向量，则将向量分组，使每组都能装入寄存器
  - ◆ 寄存器组内采用垂直处理方式
  - ◆ 寄存器组间采用水平处理方式

### ■ 这种处理方式称为**分组处理方式**

## 5.3.1 向量的流水处理



## 5.3.2 向量流水处理机

- 将向量数据表示与流水处理方式结合在一起，构成**向量流水处理机**，也称其为向量处理机。
- 向量处理机在**70年代**出现，经过**80年代**和**90年代**的发展，成为超级计算机的基础。



**CDC STAR -100**

世界上第一台使用向量处理器的计算机

计算机体系结构



**Cray-1超级计算机**



北理工计算机学院

**1976年，CRAY公司推出CRAY-1向量机，开始了向量机的蓬勃发展，其峰值速度为0.1 Gflops。**



计算机体系结构



北理工计算机学院

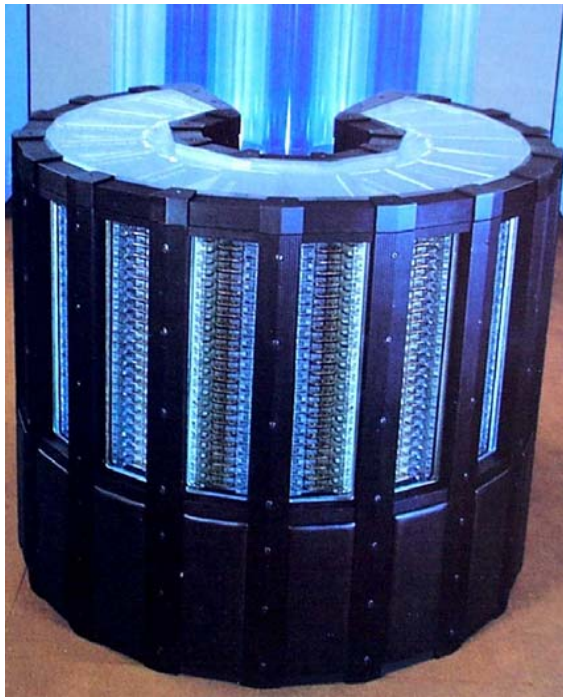
***Cray 1***



**1985年, CRAY-2, 1G flops**

**1990年, SX-3, 22G flops**

**1991年, Cray-YMP-C90, 16Gflops**



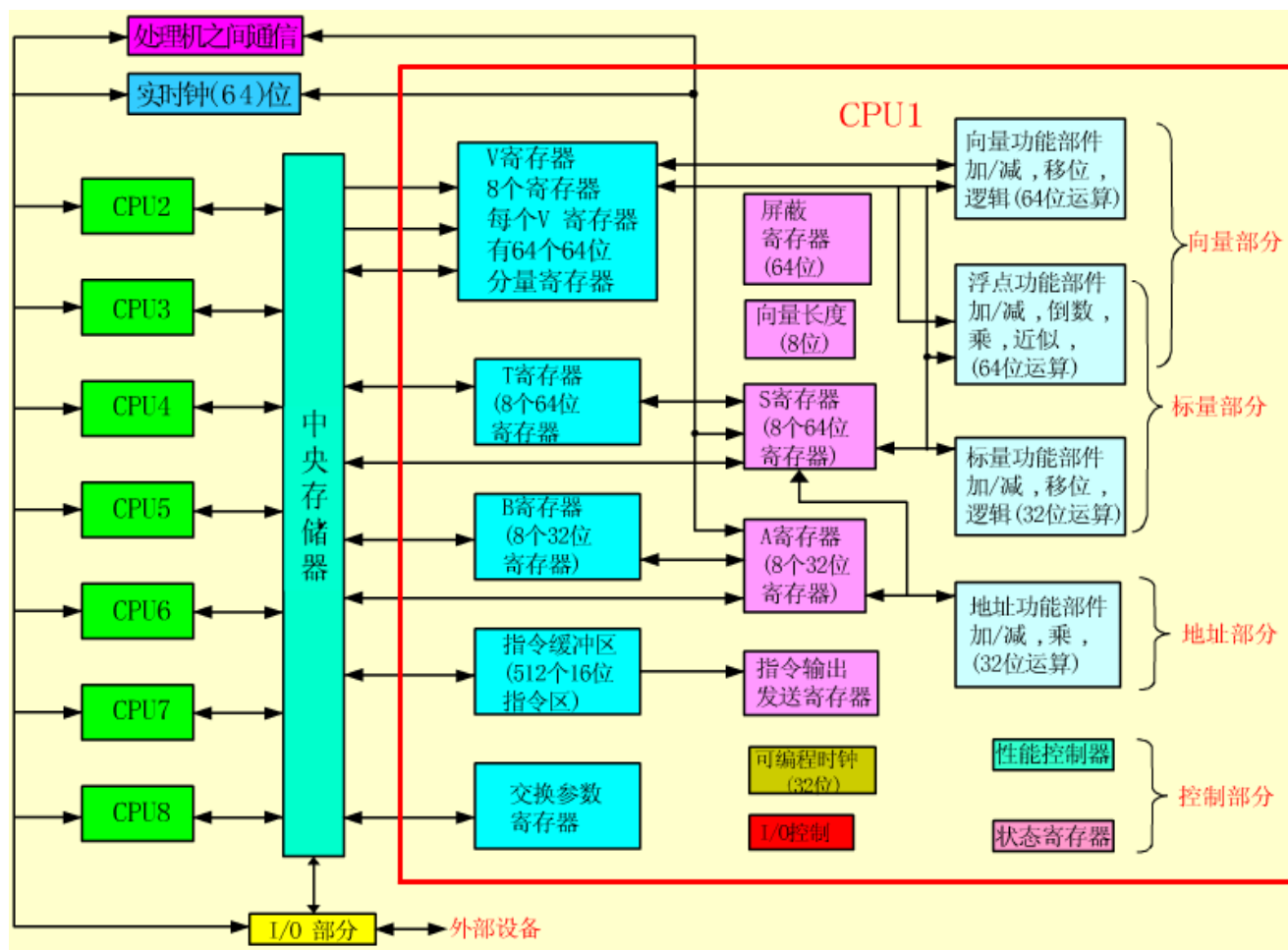
***Cray 2***



***Cray XMP/4***



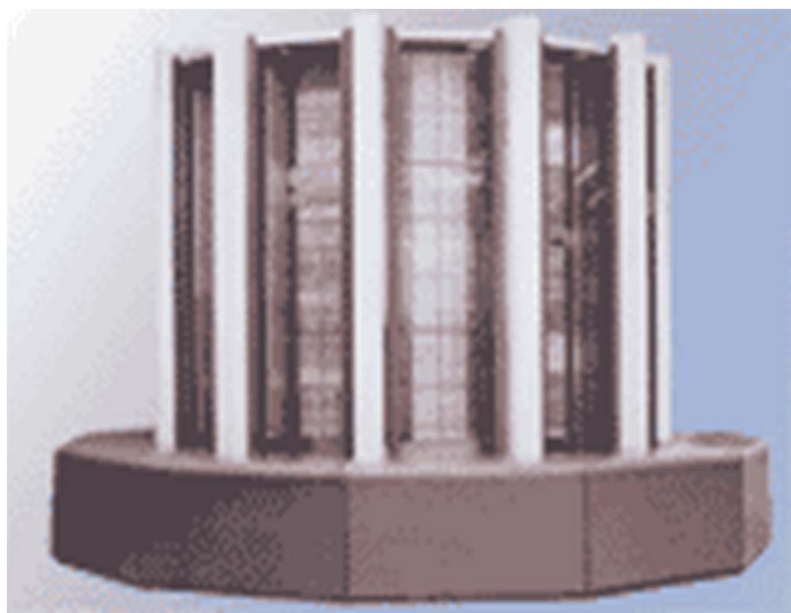
# CRAY Y-MP816系统结构



- 1991年
- 多向量处理器
- 时间并行+空间并行
- 256交叉存储
- 16MB—1GB
- 大量使用寄存器
- 64位浮点/定点

**1983年12月，银河-I巨型计算机由国防科技大学计算机研究所研制成功。**

**1992年11月，银河-II并行巨型计算机由国防科技大学计算机研究所研制成功。**



银河1



银河2

## 5.3.2 向量流水处理机

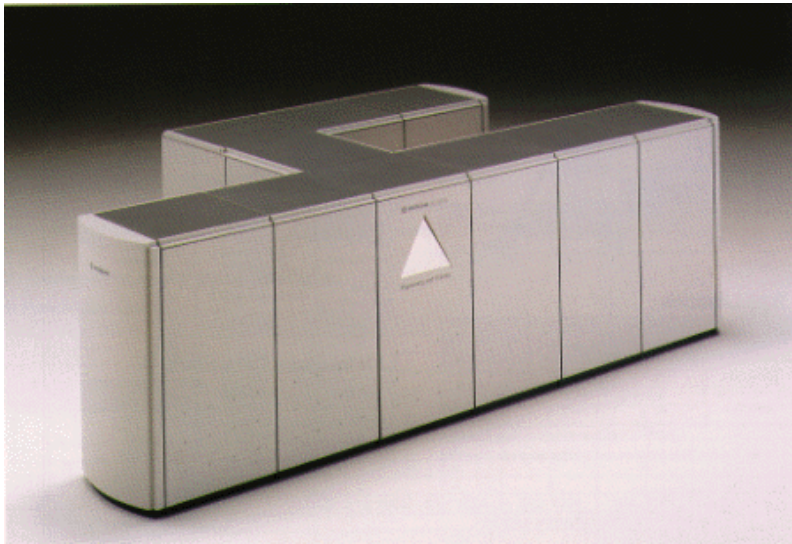
### ■ 向量处理机特点:

- 具有向量处理指令，如向量运算、向量传送、向量压缩与恢复等，可以很有效的对向量进行处理；
- 一般都采用流水线结构，有多条流水线并行工作；
- 在执行向量操作时，一条指令可以同时对向量的多个元素进行运算。**向量处理机是单指令流多数据流（SIMD）处理机。**

### ■ 向量处理机是解决数值计算问题的一种高性能计算机结构，**是向量并行计算、以流水线结构为主的并行处理计算机。**

## 5.3.2 向量流水处理机

- 向量处理机通常属大型或巨型机，也可以用微机加一台向量协处理器组成。
- 一般向量计算机中包括有一台高性能标量处理机。



**Turing Hitachi S3600**  
包括一台标量处理机和一台向量处理机



个人超级计算机

# 向量处理机的结构

- 向量处理机的**最关键问题**是存储器系统能够满足运算部件带宽的要求。
- 主要采用两种方法：
  - 1. 存储器—存储器结构
  - 2. 寄存器—寄存器结构

# 向量处理机的结构

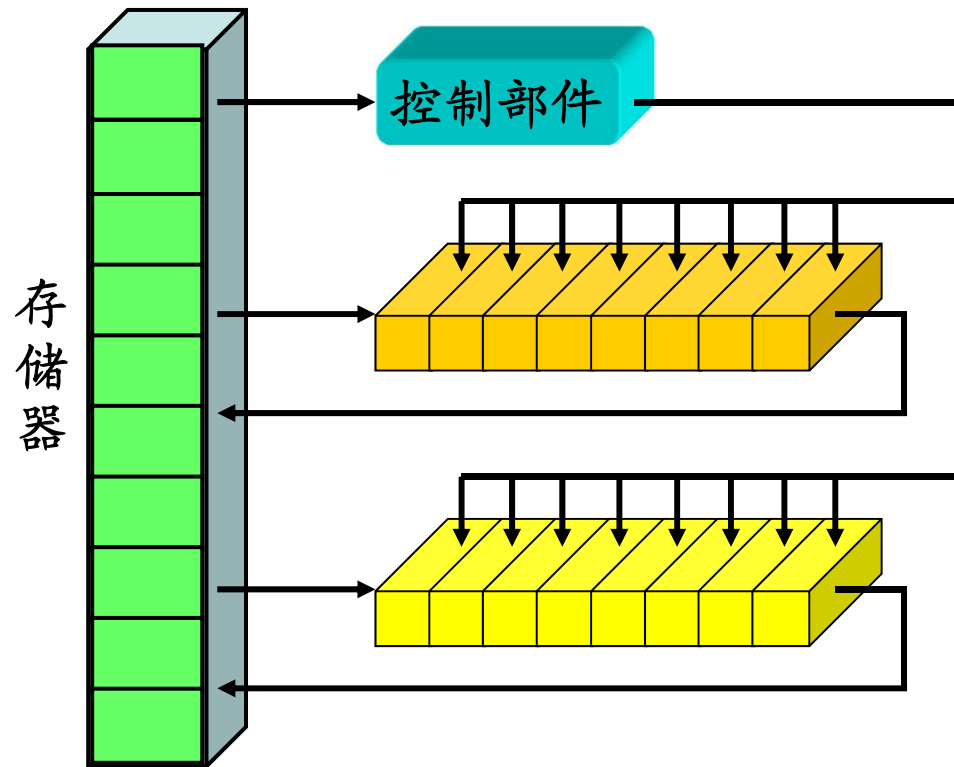
## ■ 1. 存储器—存储器结构

- 多个独立的存储器模块并行工作。
- 处理机结构简单，对存储系统的访问速度要求很高。

## ■ 2. 寄存器—寄存器结构

- 运算通过向量寄存器进行。
- 需要大量高速寄存器，对存储系统访问速度的要求降低。

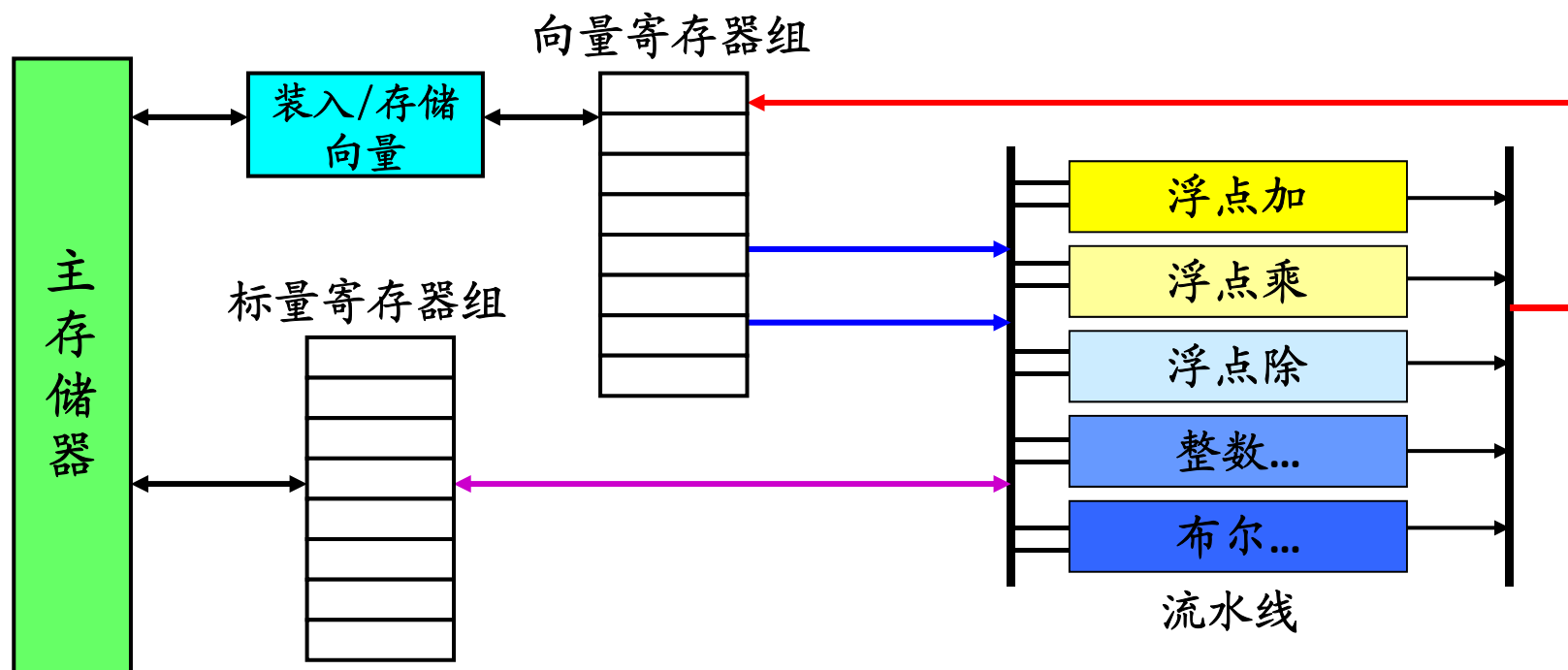
# 向量处理机的结构



- 为减少等待时间，存储器通常采用多体交叉器。多个独立的存储器模块并行工作。
- 向量元素存储在不同的存储器模块中，可以同时访问多个元素。

存储器 - 存储器结构的向量处理机

# 向量处理机的结构



寄存器 - 寄存器结构的向量处理机



# 向量处理机的指令系统

- 一般包括向量型和标量型两类指令
- 向量型运算指令一般有以下几种：
  - 向量V1运算得向量V2
  - 向量V1运算得标量S
  - 向量V1与向量V2运算得向量V3
  - 向量V1与标量S运算得向量V2

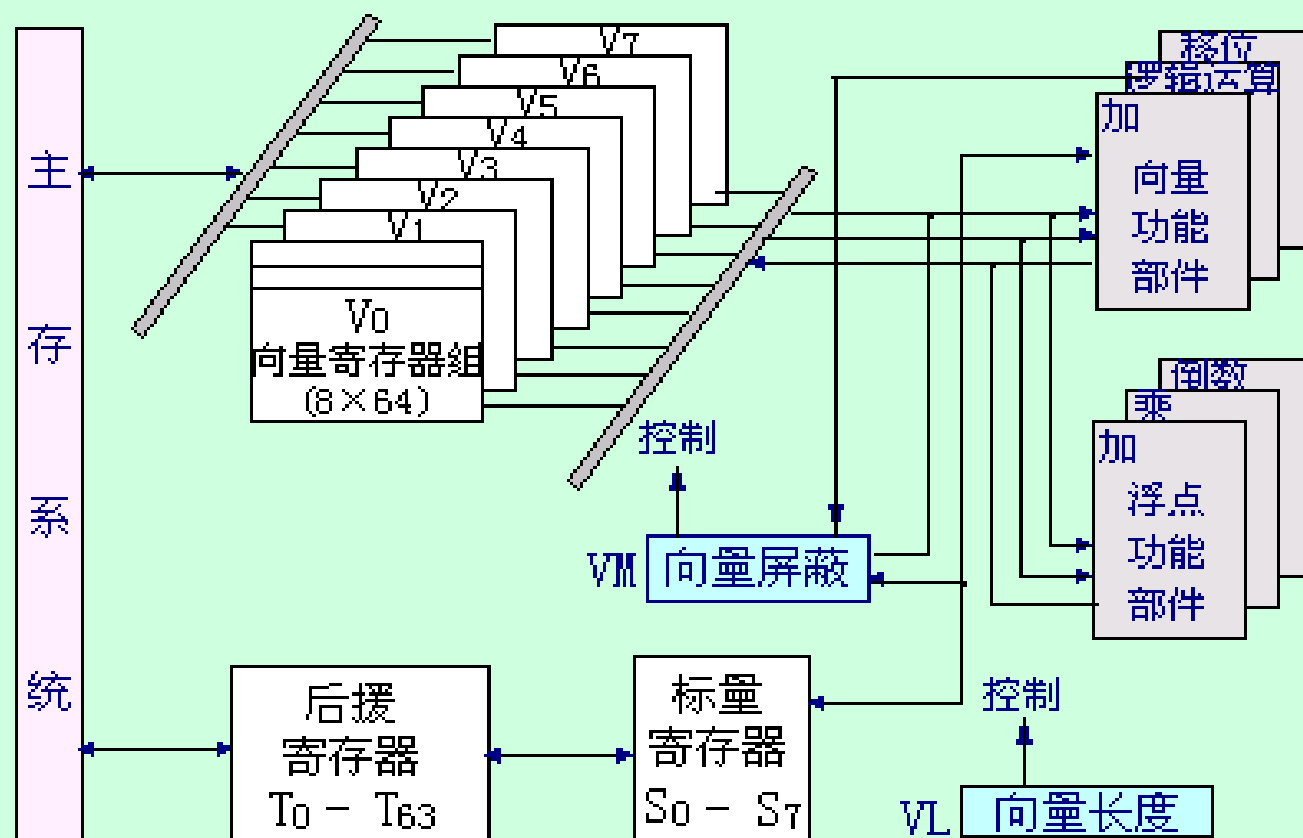
# 向量处理机的指令系统

## ■ 向量指令格式一般包括：

- 操作码
- 源或目的操作数地址
- 地址偏移量
- 地址增量
- 向量长度等

# CRAY-1

CRAY-1的基本结构



# CRAY-1结构

- 共有**12**条可**并行工作**的**单功能**流水线，可分别流水地进行地址、向量、标量的各种运算。
- **6**个单功能流水部件：**进行向量运算**
  - 整数加（**3**拍）
  - 逻辑运算（**2**拍）
  - 移位（**4**拍）
  - 浮点加（**6**拍）
  - 浮点乘（**7**拍）
  - 浮点迭代求倒数（**14**拍）

# CRAY-1结构

## ■ 向量寄存组V

- 由**512**个**64**位的寄存器组成，分成**8**块。
- 编号：**V0~V7**
- 每一个块称为一个向量寄存器，可存放一个长度（即元素个数）不超过**64**的向量。
- 每个向量寄存器可以每拍向功能部件提供一个数据元素，或者每拍接收一个从功能部件来的结果元素。

# CRAY-1结构

## ■ 标量寄存器S和快速暂存器T

- 标量寄存器有8个：**S0~S7** 64位
- 快速暂存器T用于在标量寄存器和存储器之间提供缓冲。

## ■ 向量屏蔽寄存器VM

- 64位，每一位对应于向量寄存器的一个单元。
- 作用：用于向量的归并、压缩、还原和测试操作、对向量某些元素的单独运算等。

# CRAY-1特点

- 每个向量寄存器  **$V_i$**  都有连到 **6** 个向量功能部件的单独总线。
- 每个向量功能部件也都有把运算结果送回向量寄存器组的总线。
- 只要不出现  **$V_i$**  冲突和功能部件冲突，各  **$V_i$**  之间和各功能部件之间都能并行工作，大大加快了向量指令的处理。

# CRAY-1特点

- **$V_i$ 冲突**: 并行工作的各向量指令的源向量或结果向量使用了相同的 $V_i$ 。例如: 源向量相同

$$V_3 \leftarrow V_1 + V_2$$

$$V_5 \leftarrow V_4 \wedge V_1$$

- **功能部件冲突**: 并行工作的各向量指令使用同一个功能部件。例如: 都需使用乘法功能部件

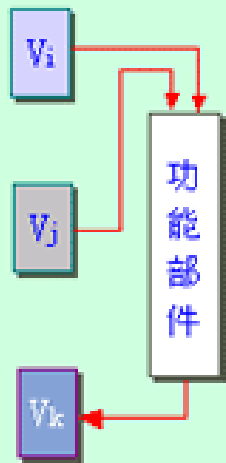
$$V_3 \leftarrow V_1 \times V_2$$

$$V_5 \leftarrow V_4 \times V_6$$

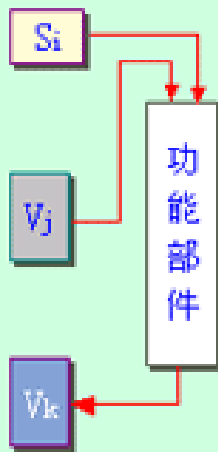


# CRAY-1 向量指令类型

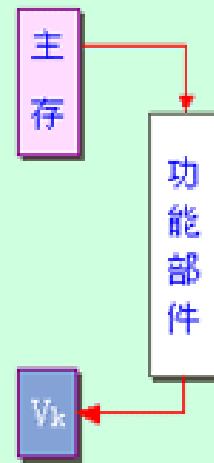
$V_k \leftarrow V_i \text{ op } V_j$   
 $V_k \leftarrow S_i \text{ op } V_j$   
 $V_k \leftarrow \text{主存}$   
 $\text{主存} \leftarrow V_i$



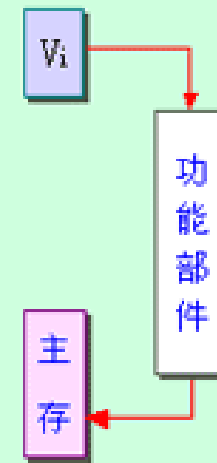
$V_k \leftarrow V_i \text{ op } V_j$



$V_k \leftarrow S_i \text{ op } V_j$



$V_k \leftarrow \text{主存}$



$\text{主存} \leftarrow V_i$

# 提高向量处理机性能的方法

- 设置多个功能部件，使它们并行工作。
- 向量与标量性能的平衡
  - 向量平衡点(vector balance point):
    - ◆ 为了使向量硬件设备和标量硬件设备的利用率相等，一个程序中向量代码所占的百分比。
- 采用链接技术，加快一串向量指令的执行。
- 采用循环开采技术，加快循环的处理。
- 采用多处理机系统，进一步提高性能。

# 学习内容

- **5.1 重叠方式**
- **5.2 流水方式**
- **5.3 向量的流水处理与向量处理机**
- **5.4 指令级高度并行的超级处理机**
- **5.5 ARM流水线处理器举例**

## 5.4 指令级高度并行的超级处理机

- 自20世纪80年代兴起RISC之后，又出现了提高指令级并行（**ILP**）的新一代处理机，让单处理机在每个时钟周期内解释多条指令。由代表性的例子是：
  - 超标量（**Superscalar**）处理机
  - 超流水线（**Superpipelining**）处理机
  - 超长指令字（**VLIW**）处理机

## 5.4 指令级高度并行的超级处理机

### ■ 超标量处理机:

- Intel公司的i860, i960,
- Pentium处理机
- Motorola公司的MC88110
- IBM公司的Power 6000
- SUN公司的SuperSPARC等。

### ■ 超流水线处理机:

- SGI公司的MIPS R4000, R5000, R10000等。

### ■ 超标量超流水线处理机:

- DEC公司的Alpha等。

## 5.4 指令级高度并行的超级处理机

机器类型	$k$ 段流水线 基准处理机	$m$ 度 超标量	$n$ 度超 流水线	$(m,n)$ 度 超标量 超流水
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	$m$	$n$	$m \times n$

超标量、超流水、超标量超流水处理机的主要性能

## 5.4 指令级高度并行的超级处理机

- 指令级并行性(ILP): 指令序列中的并行性
- 思想: 可同时流出多个指令/操作
- 表示:

$$ILP(m, n) = m * n, \quad \text{不考表示方式}$$

其中,       

**m** — 每个时钟启动的次数;

**n** — 每次启动的指令/操作个数;

- 特征:  **$IPL * CPI = 1$**

## 5.4.1 超标量处理机

- 1987年提出，其本质就是在不同的流水线中执行不相关指令的能力。
- 常规的标量流水线单处理机是在每个 $\Delta t$ 时间内解释完一条指令。称这种流水机的度为1。
- 超标量处理机采用  $m$  条指令流水线（多指令流水线），在每个 $\Delta t$ 时间内同时解释完  $m$  条指令。称这种流水机的度为 $m$ 。



# 1. 超标量处理机基本结构

## ■ 一般流水线处理机:

- 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。

## ■ 多操作部件处理机:

- 一条指令流水线，多个独立的操作部件，操作部件可以采用流水线，也可以不流水。
- 多操作部件处理机的指令级并行度小于1。

# 1. 超标量处理机基本结构

## ■ 超标量处理机:

- 多条指令流水线。
- 先进的超标量处理机有：
  - ◆ 定点处理部件**CPU**
  - ◆ 浮点处理部件**FPU**
  - ◆ 图形加速部件**GPU**
  - ◆ 大量的通用寄存器
  - ◆ 两个一级高速**Cache**等
- 超标量处理机的指令级并行度大于**1**。

# 1. 超标量处理机基本结构

- 在超标量处理机中，配置多套功能部件、指令译码电路和多组总线，并且寄存器也备有多个端口和多组总线
- 程序运行时，由指令译码部件检测顺序取出的几条指令之间是否存在数据相关和功能部件争用，将可以并行执行的指令送往流水线，否则就逐条执行

# 1. 超标量处理机基本结构

## ■ Motorola公司的MC88110:

- 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;

  - ◆ 每个寄存器堆有8个端口, 分别与8条内部总线相连接, 有一个缓冲深度为4的先行读数栈和一个缓冲深度为3的后行写数栈。

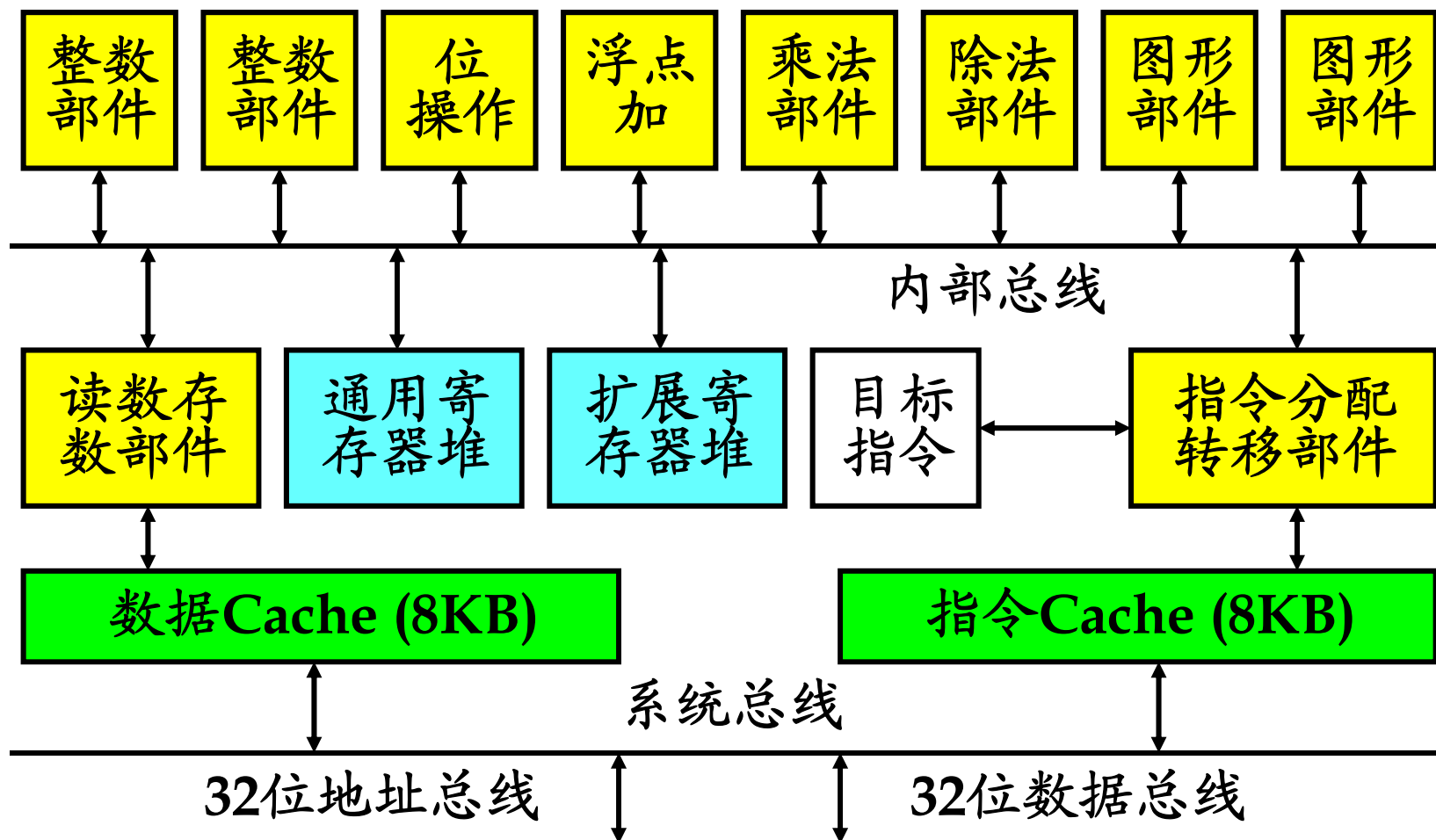
- 两个独立的高速Cache:

  - ◆ 各为8KB, 采用两路组相联方式。

- 转移目标指令Cache:

  - ◆ 在有二路分支时, 存放其中一路分支上的指令。

# 1. 超标量处理机基本结构



超标量处理机MC88110的结构

## 2. 单发射与多发射

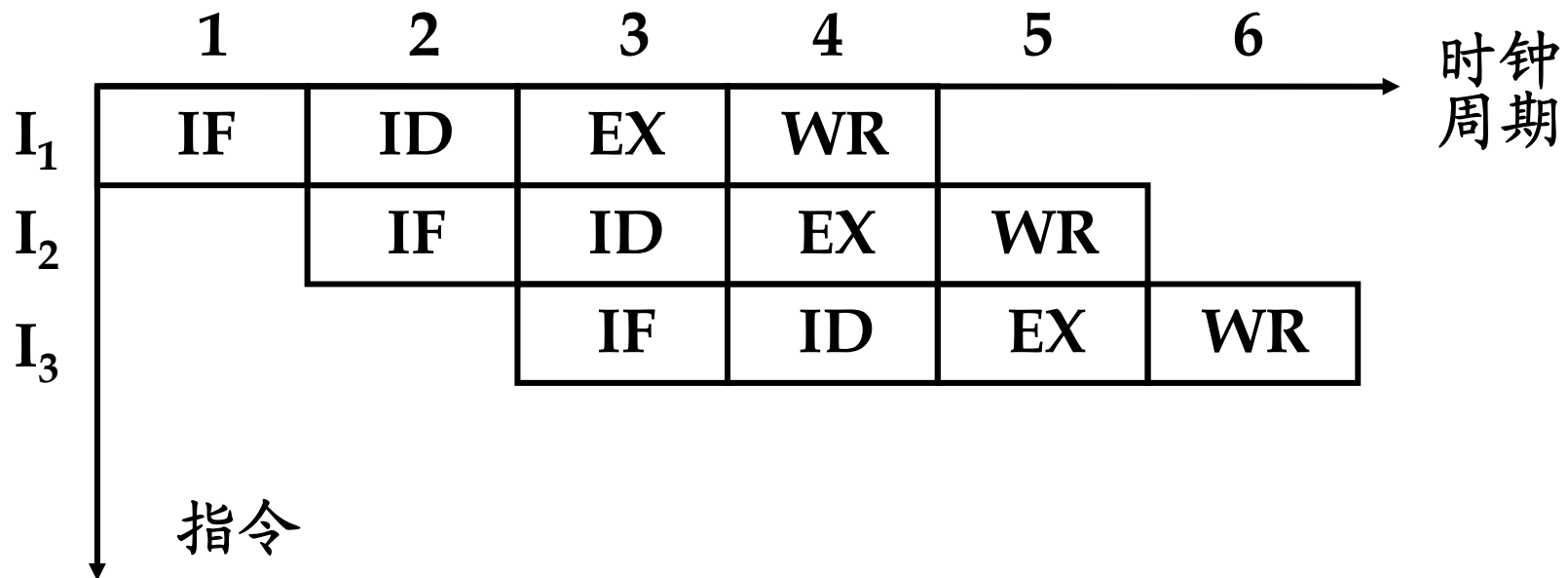
### ■ 单发射处理机:

- 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。
- 取指部件和译码部件各设置一套。
- 可以只设置一个多功能操作部件，也可以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。
- 设计目标：每个时钟周期平均执行一条指令，ILP的期望值1。

## 2. 单发射与多发射

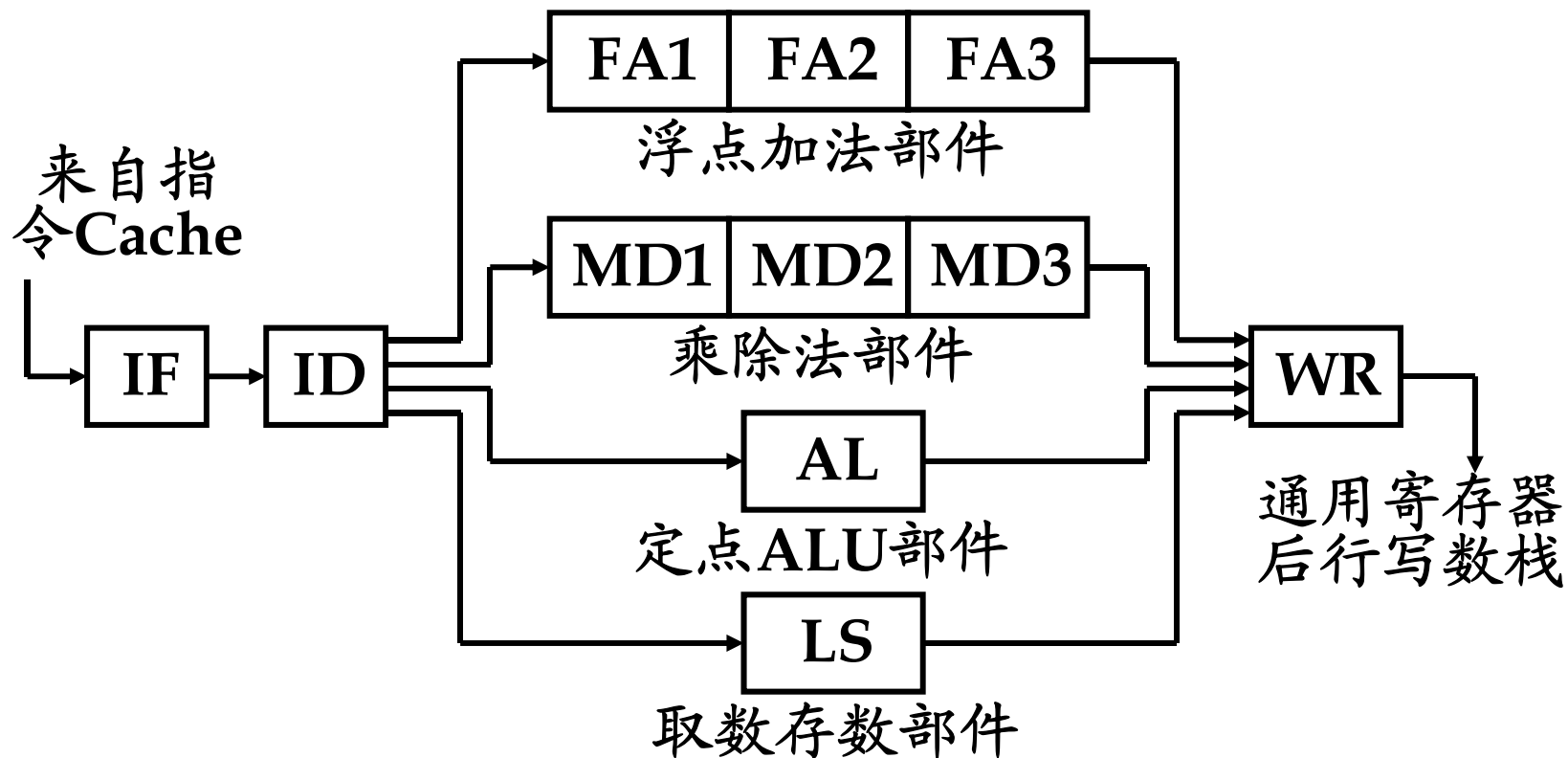
### ■ 单发射处理机:

单发射处理机的指令流水线时空图



## 2. 单发射与多发射

### ■ 单发射处理机：



单发射指令流水线



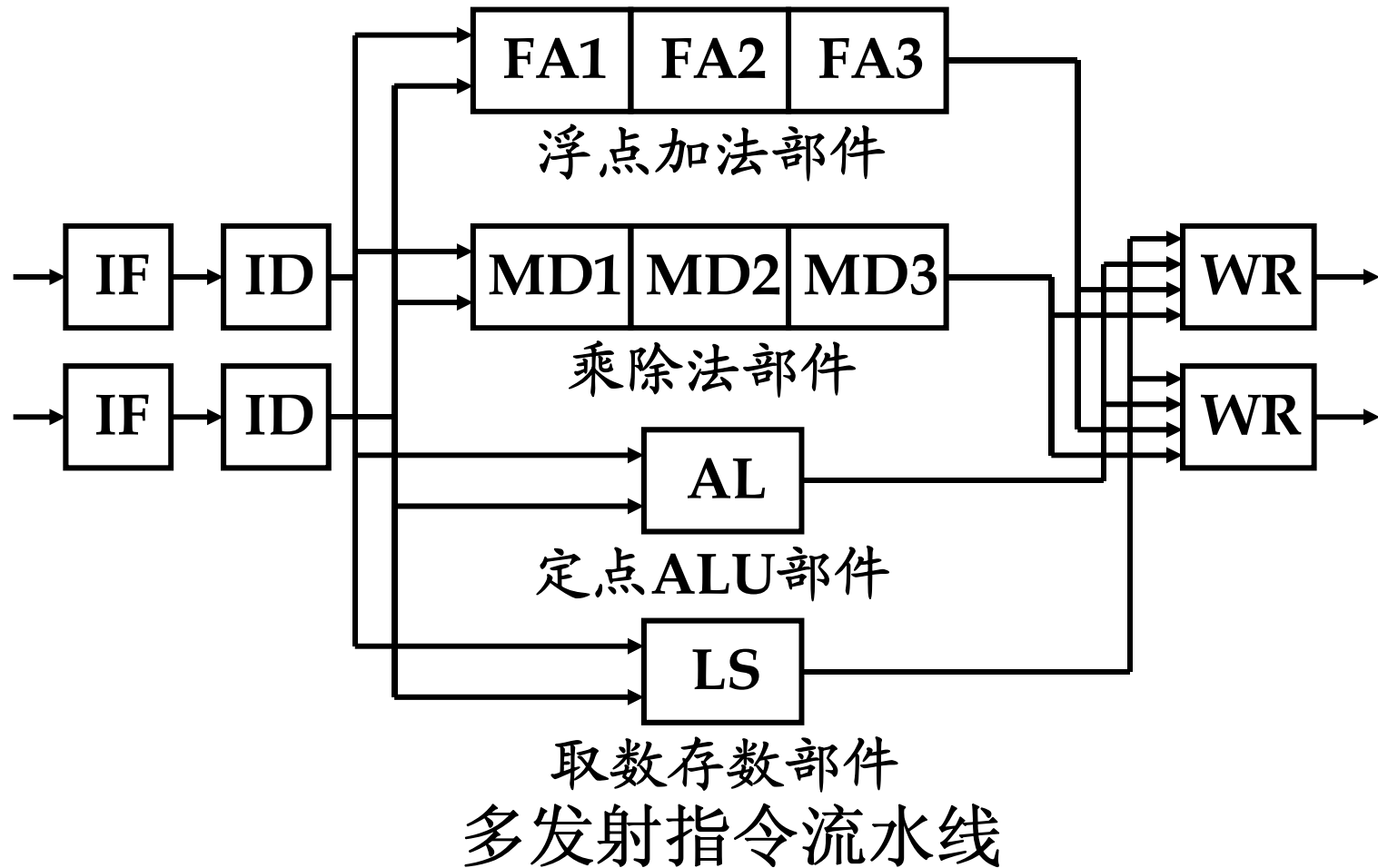
## 2. 单发射与多发射

### ■ 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。
- 需要多个取指令部件，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，复杂的指令执行部件一般采用流水线结构。
- 设计目标：每个时钟周期平均执行多条指令，ILP的期望值大于1。

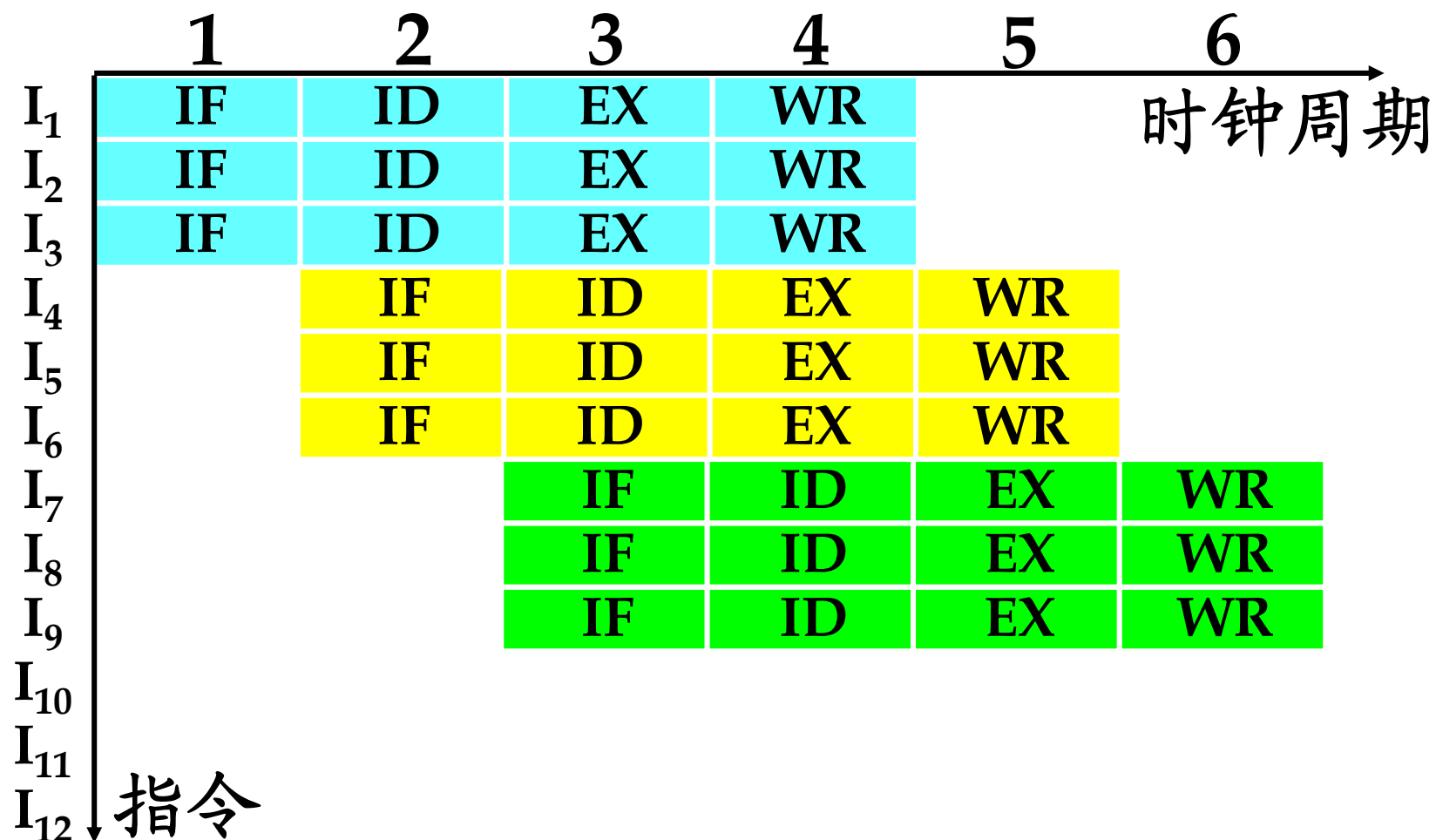
## 2. 单发射与多发射

### ■ 多发射处理机:



## 2. 单发射与多发射

3条4段流水线，每时钟周期同时发射3条指令



## 2. 单发射与多发射

### ■ 超标量处理机:

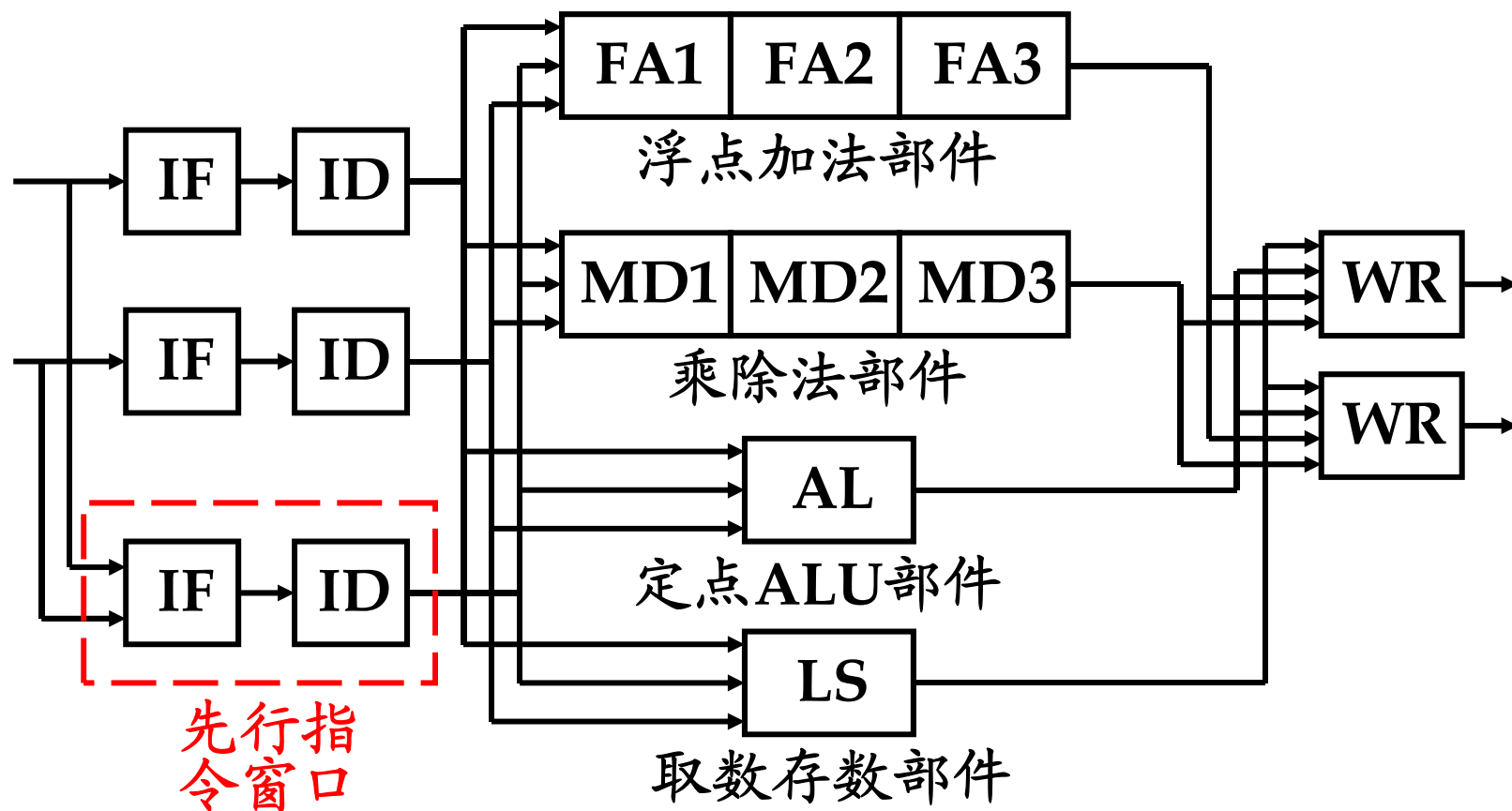
- 一个时钟周期内能够同时发射多条指令的处理机称为超标量处理机。
- 必须有两条或两条以上能够同时工作的指令流水线。
- 指令级并行度:  $1 < \text{ILP} < m$ 。  
 $m$  为每个周期同时发射的指令条数。

## 2. 单发射与多发射

### ■ 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的检测。
  - ◆ 窗口的大小：一般为**2至8**条指令。
- 采用目前的指令调度技术，每个周期发射**2至4**条指令比较合理。

## 2. 单发射与多发射



有先行指令窗口的多发射指令流水线

### 3. 超标量处理机性能

- 在指令级并行度为 **$(m,1)$** 、 **$k$** 段流水线的超标量处理机上，执行 **$N$** 条指令所用的时间为：

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

- 超标量处理机相对于单流水线普通标量处理机的加速比为：

期末考试不考

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

当 **$N \rightarrow \infty$**  时，在没有资源冲突、没有数据相关和控制相关的理想情况下，超标量处理机相对于单流水线普通标量处理机的加速比最大值为： **$S(m,1)_{\max} = m$**

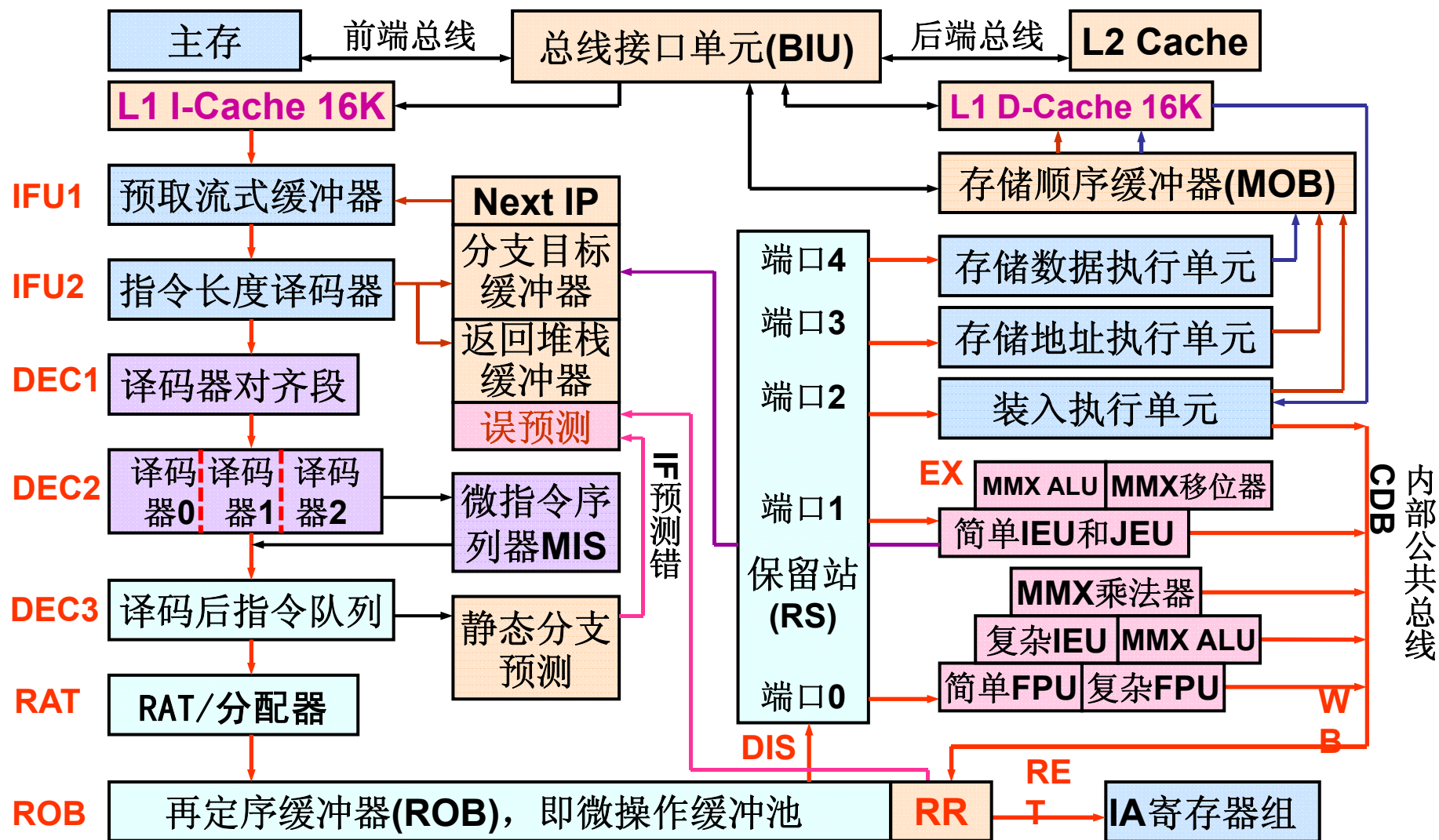
## 5.4.1 超标量处理机

- 典型的超标量流水线处理机有**IBM RS/6000**、**DEC 21064**、**Intel i960CA**、**Tandem**、**Cyclone**等。
- **1986年的Intel i960CA**时钟频率为**25 MHz**，度 $m=3$ ，有**7**个功能部件可以并发使用。**1990年的IBM RS/6000**使用**1 $\mu$ m CMOS**工艺，时钟频率为**30 MHz**。处理机中有转移处理、定点、浮点**3**种功能部件，它们可并行工作。转移处理部件每 $\Delta t$ 可执行多达**5**条指令，度 $m=4$ ，性能可达**34 MIPS**和**11 MFLOPS**。非常适合于在数值计算密集的科学工程上应用及在多用户商用环境下工作。许多基于**RS/6000**的工作站和服务器都是**IBM**生产的。如**POWER Station 530**。
- **1992年的DEC 21064**使用**0.75 $\mu$ m CMOS**，时钟频率为**150 MHz**，度 $m=2$ ，**10**段流水线，最高性能可达**300MIPS**和**150MFLOPS**。**Tandem**公司的**Cyclone(旋风)**计算机由**4**到**16**台超级标量流水处理机组成。每个处理机的寄存器组有**9**个端口(其中**5**个为读，**4**个为写)，有两个算术逻辑部件，度 $m=2$ 。由于程序中可开发的指令并行性有限，所以超标量流水线处理机的度 $m$ 比较低。

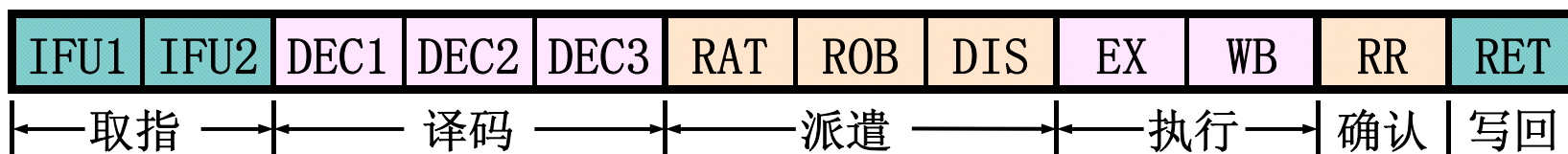


# PII CPU的超标量流水技术

特征—哈佛结构+DIB，3路超标量，动态执行技术



## 超标量流水线组成—12个段，以再定序缓冲器ROB为核心



### (1)IFU1 取指单元段1

每次从L1-I\$取出一个块(32B)，装入预取流式缓冲器(S=2个块)

预取流式缓冲器的空闲 $\geq 1$ 个块时

### (2)IFU2 取指单元段2

每次从预取流式缓冲器取出16B信息(起始位置任意);

对16B信息预译码、标志指令边界(3条指令);

若发现转移指令，则进行动态分支预测(指令地址送BTB)

指令被取走时

※取指段特征—按需动作(非按拍动作)，按序流动

### (3)DEC1 译码段1

每次从IF段取3条IA指令，按一定次序旋转(对应译码器结构)

### (4)DEC2 译码段2

每次同时译码3条指令，形成≤6个uop(118 bit/uop)

↓ ↓  
(结构为复杂/简单/简单) (4+1+1，复杂指令通过**MIS**翻造)

### (5)DEC3 译码段3

每次接收≤6个uop，按程序顺序排队(DIQ)；

若发现转移型uop、且BTB缺失，则进行静态分支预测；  
(含IF段误预测修正)

DIQ空闲≥6个uop时

※译码段特征—CISC→RISC，按需动作，按序流动

(6)**RAT** 寄存器别名表和分配器段 --按序发射

每拍取3个uop，将uop中的IA寄存器，转换为内部寄存器  
(如**ROB** “目的值” 字段)←┐

△此段消除/转化了**WAR**冒险、消除了**WAW**冒险

(7)**ROB** 再定序缓冲器段

每拍接收3个uop、按序存放在**ROB**中

△**ROB**为环形缓冲区，管理各个uop(收数据、改状态)

(8)**DIS** 派遣段 --乱序派遣

每拍RS从**ROB**以次序任意拷贝多个**OPD**就绪的uop到相应端口；  
**RS**在**EX**单元可用时，发送相应端口的uop至**EX**单元

△此段消除了**RAW**冒险

※派遣段特征—冒险处理，按需动作→按拍动作，  
按序流动→乱序流动

### (9)EX 执行段

各部件执行uop，结果送上CDB；若uop为分支操作，更新BTB

### (10)WB 写回段

ROB从CDB接收执行结果、修改状态

※执行段特征—乱序执行，部件时延可不等(动态流水线)

### (11)RR 回收就绪段(确认段)

按程序顺序、以IA指令为单位，对所含uop进行确认；  
处理分支误预测、异常(清除部分或全部ROB)

※确认段特征—RISC→CISC，乱序流动→按序流动

### (12)RET 回收段

将IA指令的结果写回IA寄存器，或通知MOB完成写L1\$；  
清除ROB中该IA指令对应的uop(有效位复位)

## 5.4.2 超流水线处理机

### ■ 两种定义:

- 一个周期内能够**分时发射**多条指令的处理机称为超流水线处理机。
- 指令流水线有8个或更多功能段的流水线处理机称为超流水线处理机。

## 5.4.2 超流水线处理机

- **不同于**超标量处理机和超长指令字处理机，超流水线处理机**每个 $\Delta t'$ 仍只流出一条指令**，但它的 $\Delta t'$ **很小**
- 一台度为 **$n$** 的超流水线处理机， $\Delta t'$ 只是基本机器周期 $\Delta t$ 的 **$1/n$**
- 因此，只要流水线的性能得以发挥，其并行度就可以达到 **$n$**

# 提高处理机性能的不同方法

- **超标量处理机**利用资源重复，设置多个执行部件寄存器端口
- **超流水线处理机**则侧重开发时间并行性，在公共硬件上采用较短的时钟周期、深度流水来提高速度

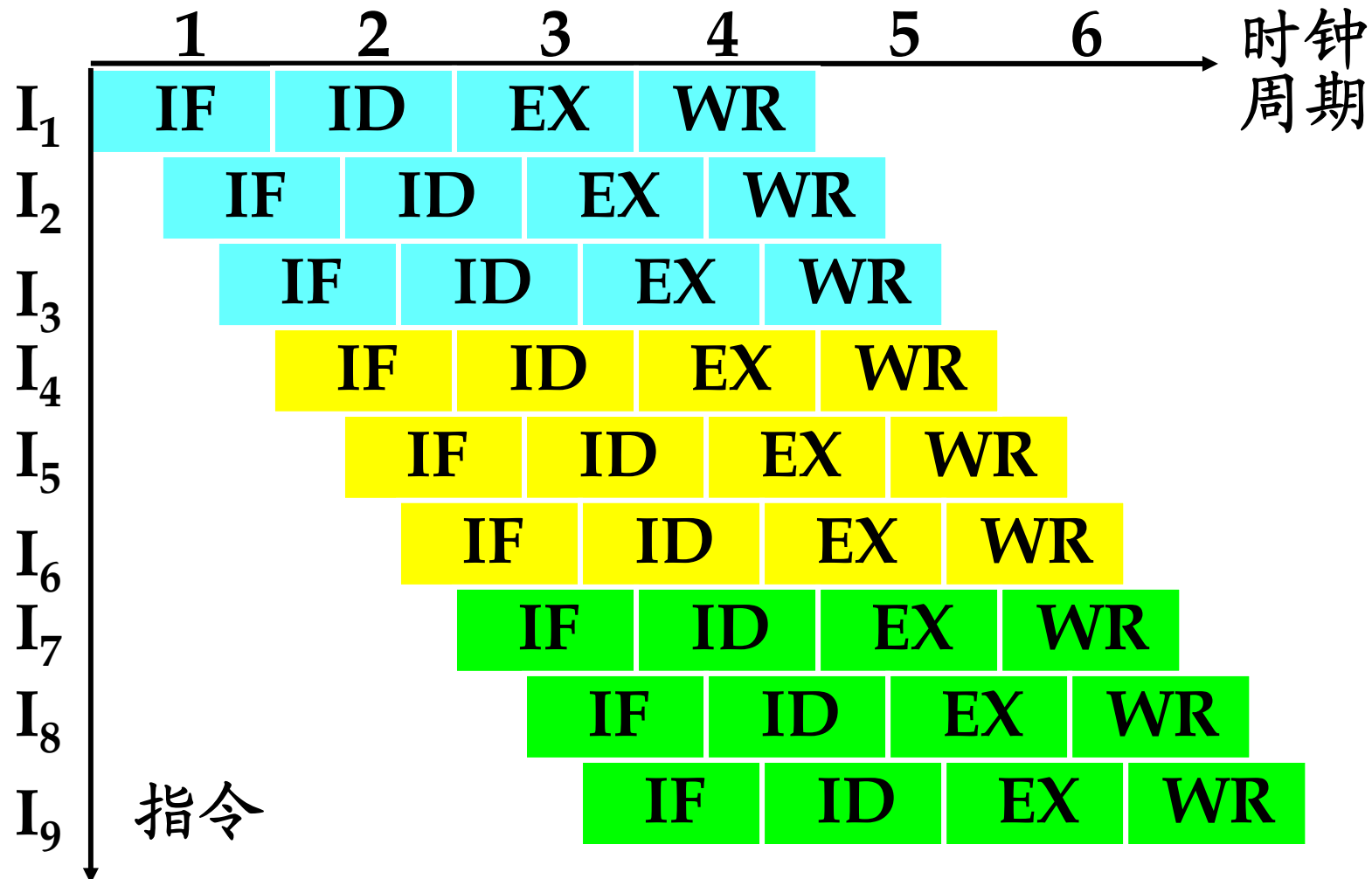


# 超流水线处理机指令执行序列

- 超流水线处理机需要使用多相时钟
- 没有高速时钟机制，超流水线处理机是无法实现的
- 指令执行时序
  - 每隔 $1/n$ 个时钟周期发射一条指令，流水线周期为 $1/n$ 个时钟周期

# 超流水线处理器指令执行序列

每个时钟周期分时发送3条指令的超流水线



# 超流水线处理机性能

- 在指令级并行度为(1, n)、k段流水线的超流水线处理机执行N条指令所的时间为：

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

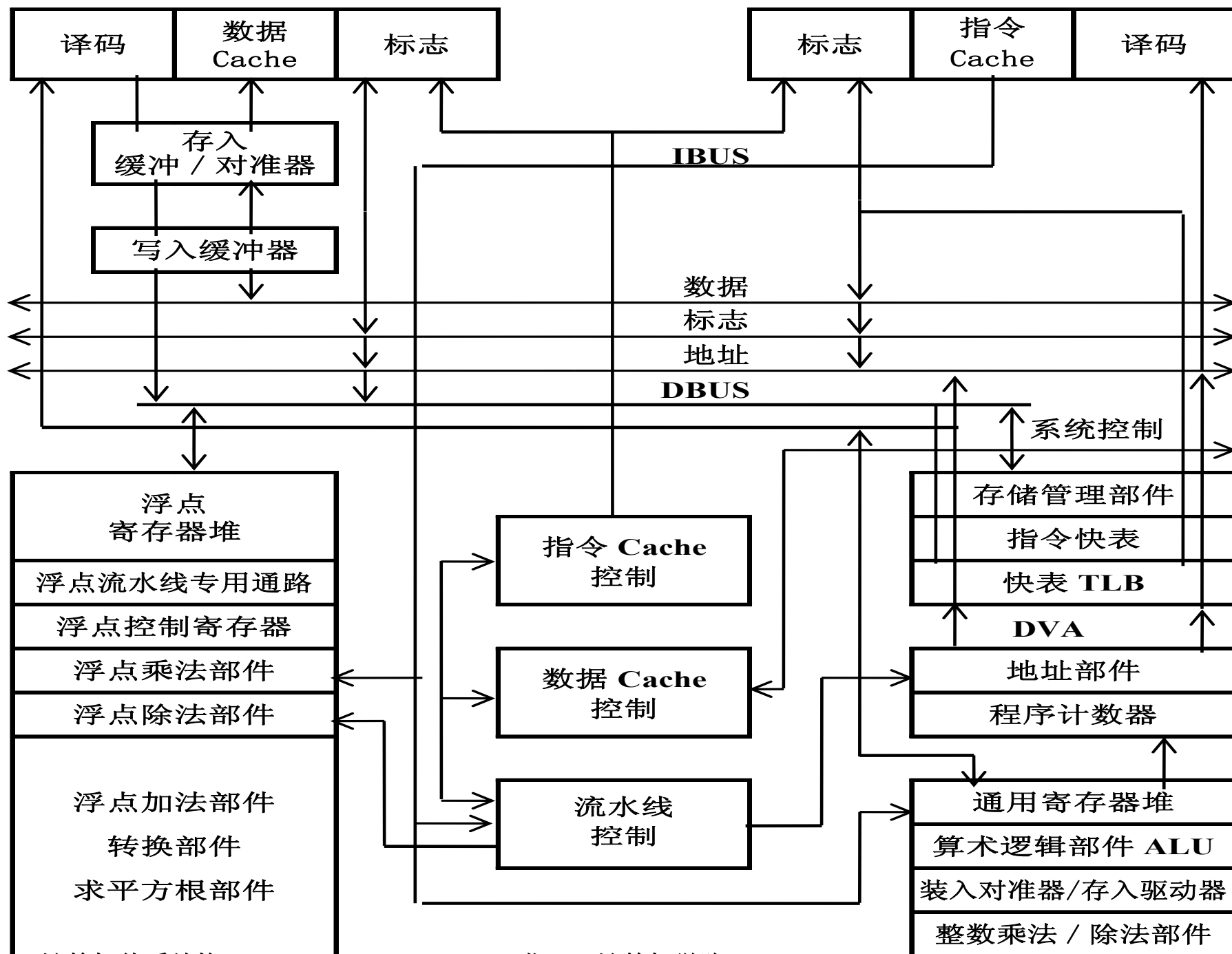
- 超流水线处理机相对于单流水线普通标量处理机的加速比为：

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

当 $N \rightarrow \infty$  时，在没有资源冲突、没有数据相关和控制相关的理想情况下，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最大值为： **$S(1, n)_{\max} = n$**

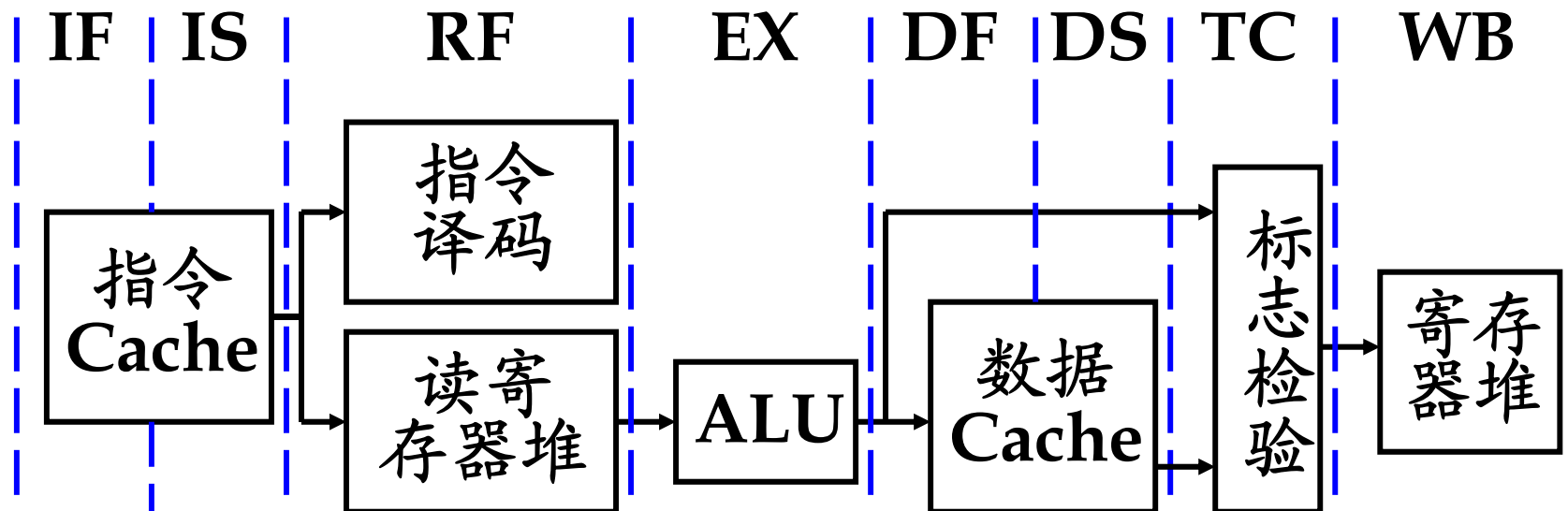
# 超流水线处理机： MIPS R4000

- 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。指令流水线有**8个流水段**。
- 有**两个Cache**：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，因此在一个时钟周期内可以从指令**Cache**中读出两条指令，从数据**Cache**中读出或写入两个数据。
- 主要运算部件有整数部件和浮点部件。



# 超流水线处理机： MIPS R4000

## MIPS R4000处理机的流水线操作



IF: 取第一条指令

IS: 取第二条指令

RF: 读寄存器堆, 指令译码

EX: 执行指令

DF: 取第一个数据

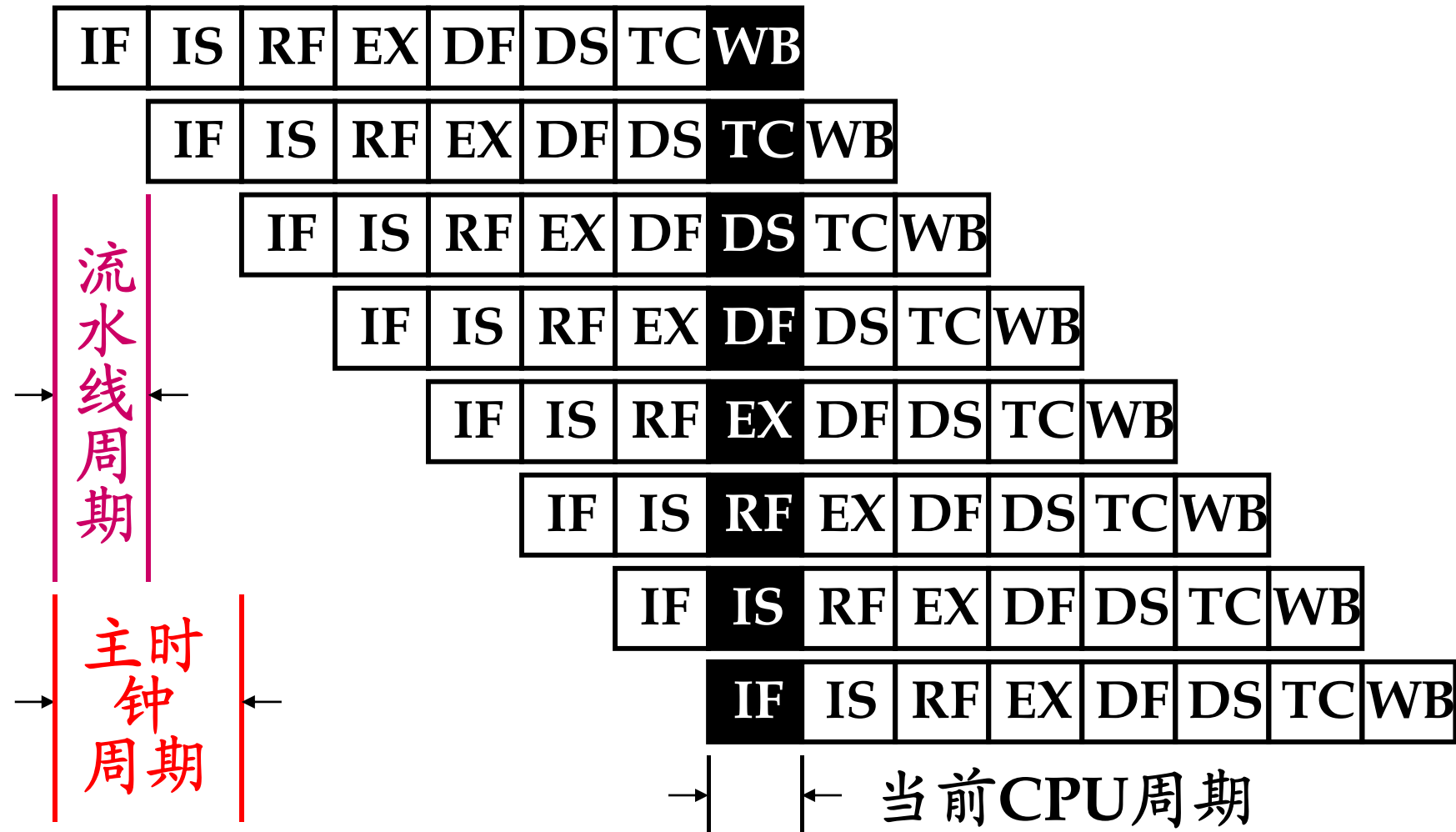
DS: 取第二个数据

TC: 数据标志校验;

WB: 写回结果

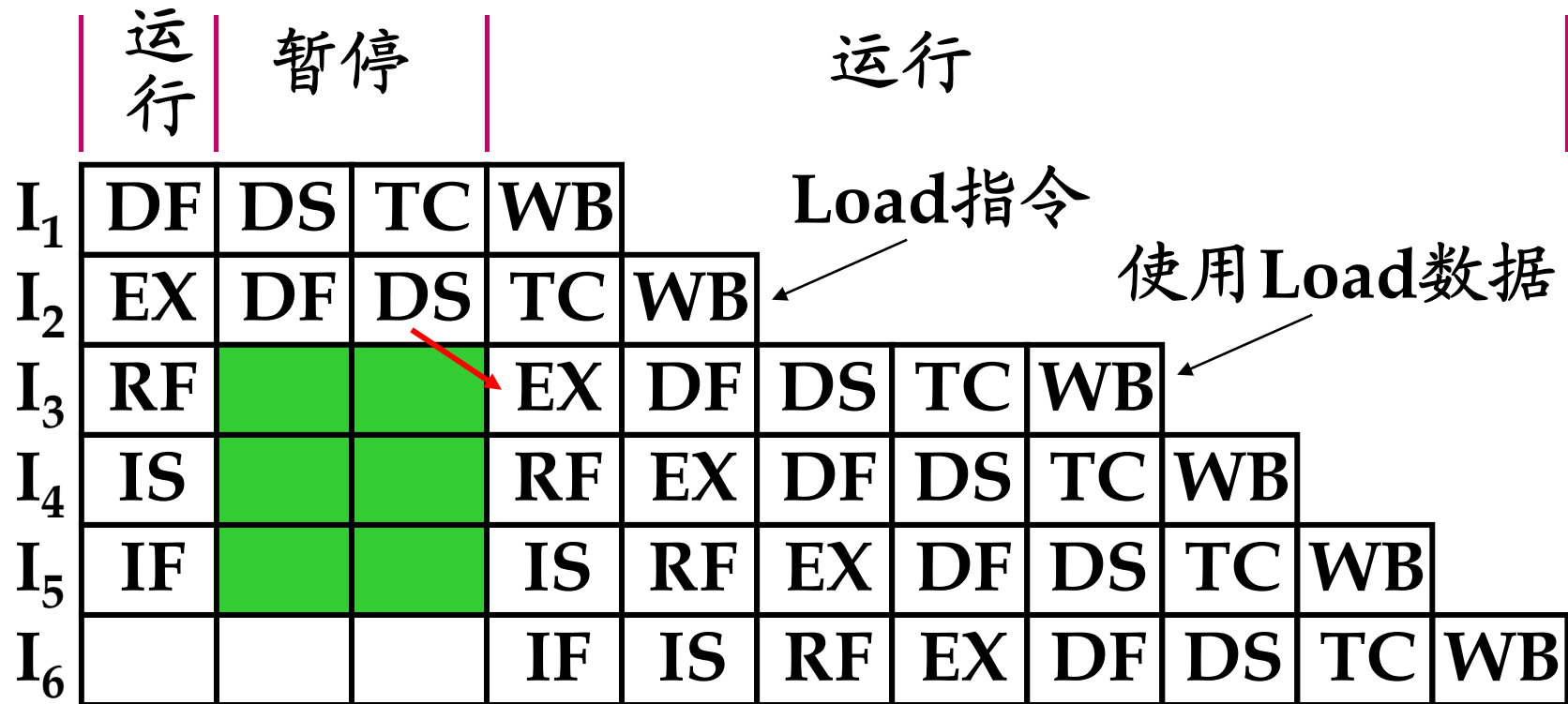
# 超流水线处理机： MIPS R4000

MIPS R4000正常指令流水线工作时序



# 超流水线处理机： MIPS R4000

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。 采用顺序发射方式。



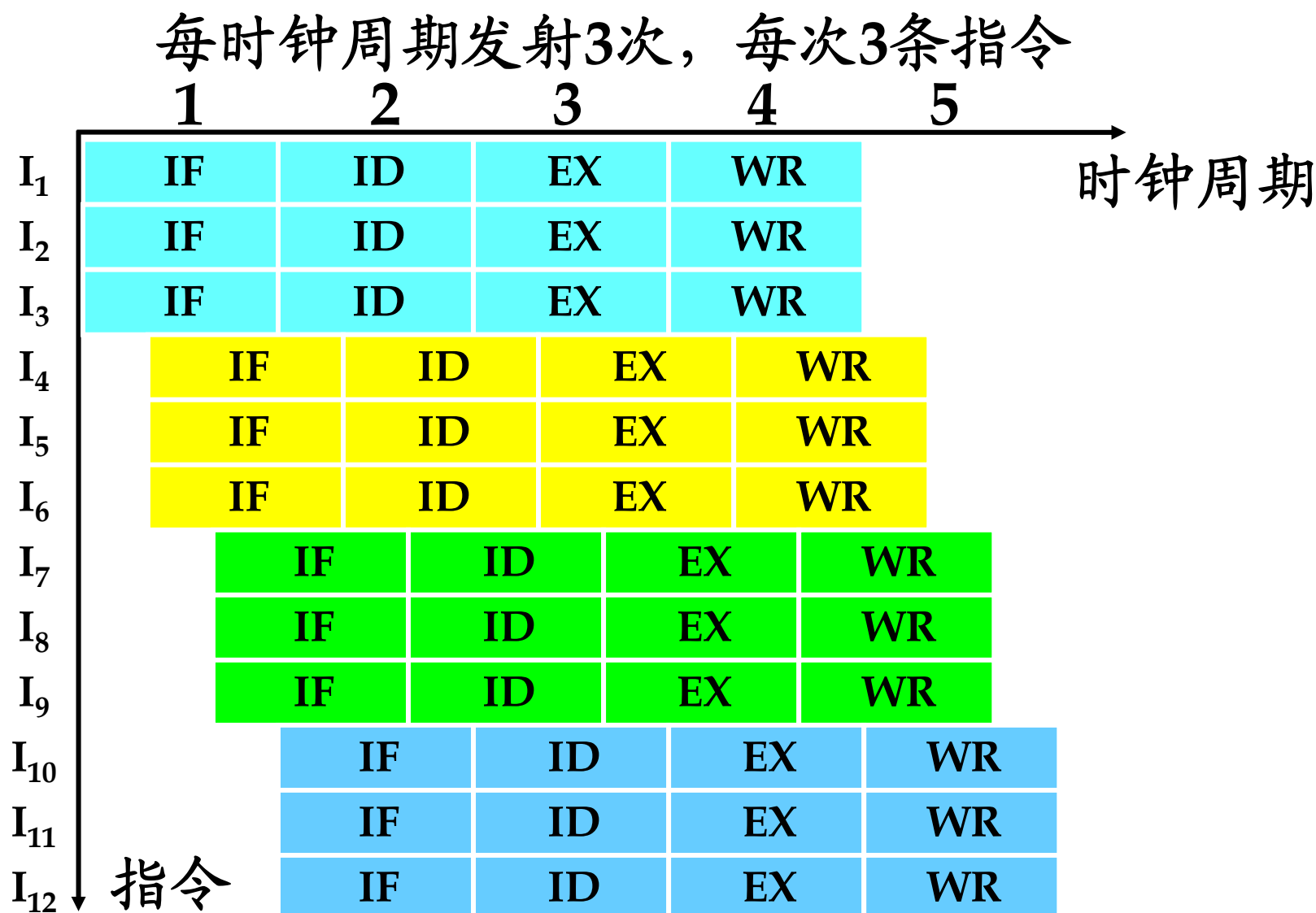
MIPS R4000 有暂停的指令流水线工作时序



## 5.4.3 超标量超流水线处理机

- 把超标量与超流水线技术结合在一起，就成为**超标量超流水线处理机**。
- 指令执行时序
  - 超标量超流水线处理机在一个时钟周期内分时发射指令 $n$ 次，每次同时发射指令 $m$ 条，每个时钟周期总共发射指令 $m \cdot n$ 条。

# 指令执行序列



# 超标量超流水线处理机性能

- 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：

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

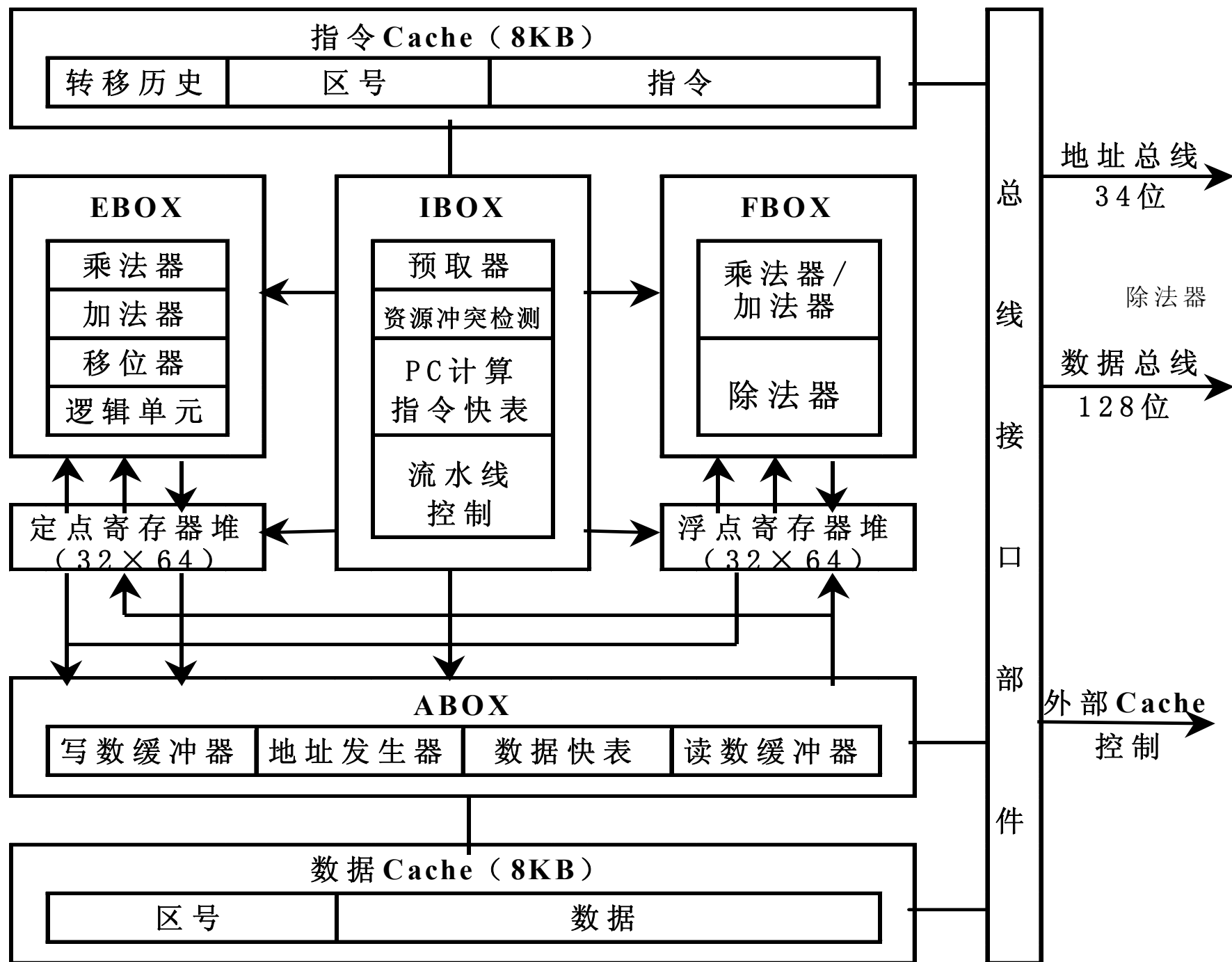
当 $N \rightarrow \infty$  时，在没有资源冲突、没有数据相关和控制相关的理想情况下，超标量超流水线处理机相对于单流水线普通标量处理机的加速比最大值为： **$S(m, n)_{\max} = m \times n$**

# 超标量超流水线处理机： DEC Alpha

- 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**和中央控制部件**IBOX**。
- 中央控制部件**IBOX**可以同时从指令**Cache**中读入两条指令，同时对读入的两条指令进行译码，并且对这两条指令作资源冲突检测，进行数据相关性和控制相关性分析。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**三个指令执行部件中的两个。
- 指令流水线采用顺序发射乱序完成的控制方式。在指令**Cache**中有一个转移历史表，实现条件转移的动态预测。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。

# 超标量超流水线处理机： DEC Alpha

- **Alpha 21064**处理机共有**三条指令流水线**  
整数操作流水线和访问存储器流水线分为**7个流水段**，其中，取指令和分析指令为**4个流水段**，运算**2个流水段**，写结果**1个流水段**。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**。
- 所有指令执行部件**EBOX**、**IBOX**、**ABOX**和**FBOX**中都设置由专用数据通路。
- **Alpha 21064**处理机的三条指令流水线的平均段数为**8段**，每个时钟周期发射两条指令。因此，**Alpha 21064**处理机是超标量超流水线处理机。



# 超标量超流水线处理机：DEC Alpha

7个流水段的整数操作流水线

0	1	2	3	4	5	6
IF	SWAP	I <sub>0</sub>	I <sub>1</sub>	A <sub>0</sub>	A <sub>1</sub>	WR

**IF**      取值

**SWAP** 交换双发射指令、转移预测

**I<sub>0</sub>**      指令译码

**I<sub>1</sub>**      访问通用寄存器堆，发射校验

**A<sub>1</sub>**      计算周期1，**IBOX**计算新的**PC**值

**A<sub>2</sub>**      计算周期2，查指令快表

**WR**      写整数寄存器堆，指令**Cache**命中检测

# 超标量超流水线处理机： DEC Alpha

7个流水段的访问存储器流水线

0	1	2	3	4	5	6
IF	SWAP	I <sub>0</sub>	I <sub>1</sub>	AC	TB	HM

**IF**      取值

**SWAP** 交换双发射指令、转移预测

**I<sub>0</sub>**      指令译码

**I<sub>1</sub>**      访问通用寄存器堆，发射校验

**AC**      **ABOX**计算有效数据地址

**TB**      查数据快表

**HM**      写读数缓冲栈，数据**Cache**命中/  
不命中检测



# 超标量超流水线处理机： DEC Alpha

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	I <sub>0</sub>	I <sub>1</sub>	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	FWR

**IF**      取值

**SWAP** 交换双发射指令、转移预测

**I<sub>0</sub>**      指令译码

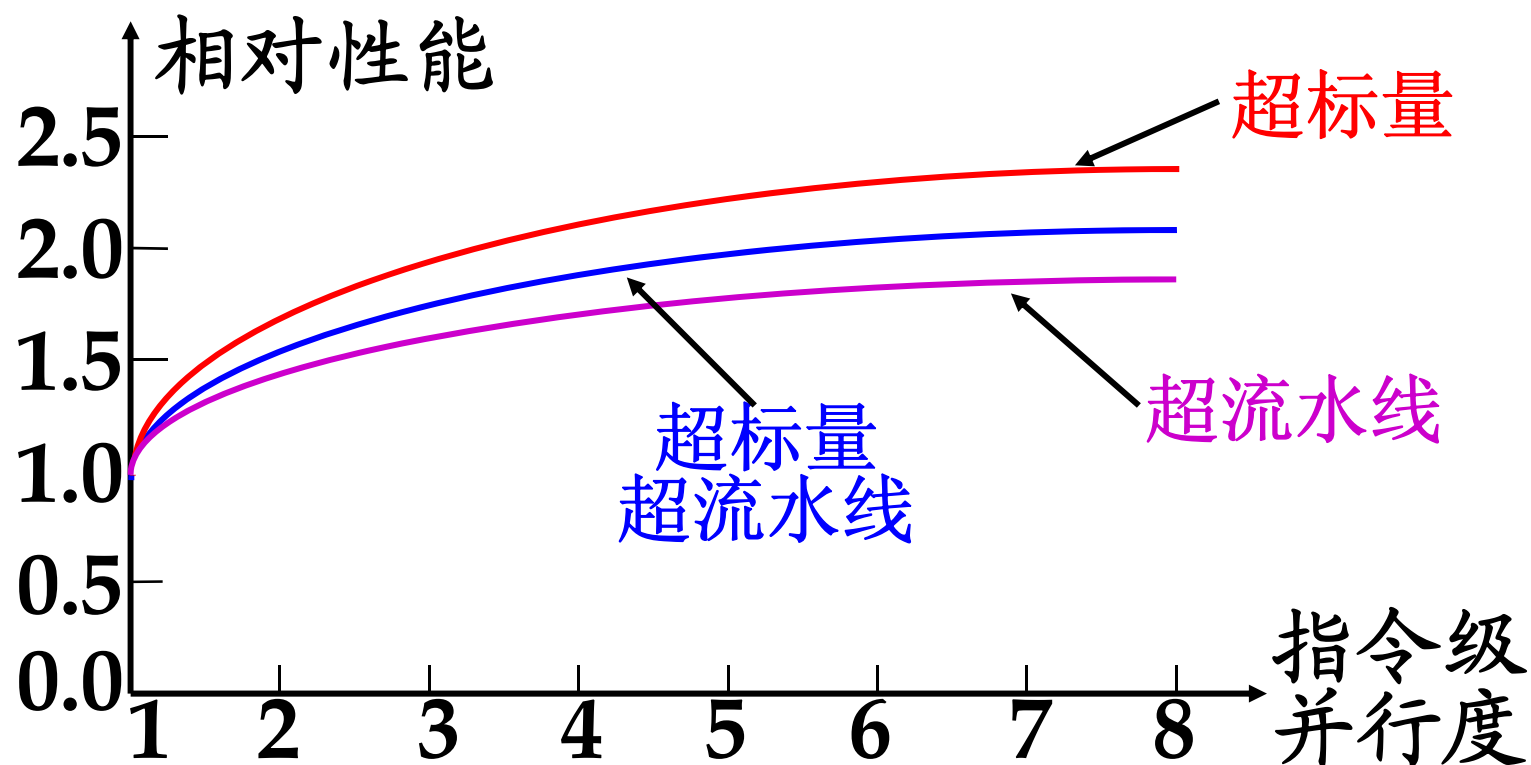
**I<sub>1</sub>**      访问通用寄存器堆，发射校验

**F<sub>1</sub>-F<sub>5</sub>** 浮点计算流水线

**FWR**    写回浮点寄存器堆

## 5.4.4 三种指令级并行处理机性能比较

### ■ 性能比较



## 5.4.4 三种指令级并行处理机性能比较

### ■ 性能比较

- 最高：超标量处理机；
- 其次：超标量超流水线处理机；
- 最低：超流水线处理机。

#### 主要原因：

- (1) 超流水线处理机的启动延迟比超标量处理机大。
- (2) 条件转移造成的损失，超流水线处理机要比超标量处理机大。
- (3) 超标量处理机指令执行部件的冲突要比超流水线处理机小。

## 5.4.4 三种指令级并行处理机性能比较

### ■ 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。

## 5.4.4 三种指令级并行处理机性能比较

### ■ 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，是有一个确定的值。
- 这个最大值主要由程序自身的语义来决定，与这个程序运行在那一种处理机上无关。
- 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。当然，对于各个不同程序，这个收拢点的位置也是不同的。

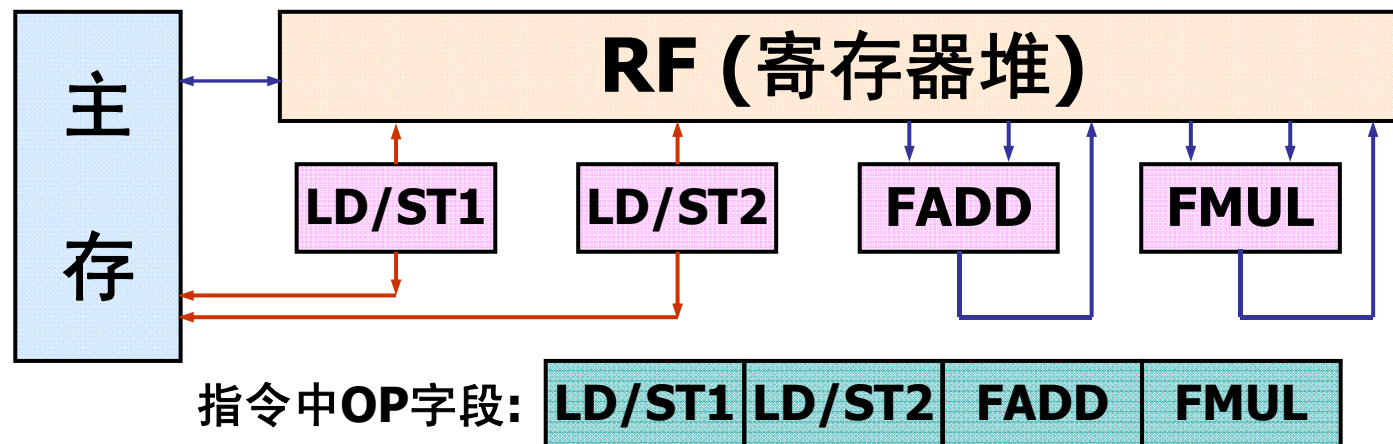
## 5.4.5 超长指令字处理机

- 1983年，Yale大学Fisher教授首先提出。
- **VLIW (Very Long Instruction Word)** 结构将水平型微码与超标量处理两者相结合，指令字长可达数百位，多个功能部件并发工作，共享大容量寄存器堆
- 与超标量处理机不同的是，编译时，编译程序找出指令间潜在的并行性，将多个能并行执行的不相关或无关的操作先行压缩，组合在一起，形成一条有多个操作段的超长指令
- 运行时，这条超长指令控制多个相互独立的功能部件并行操作

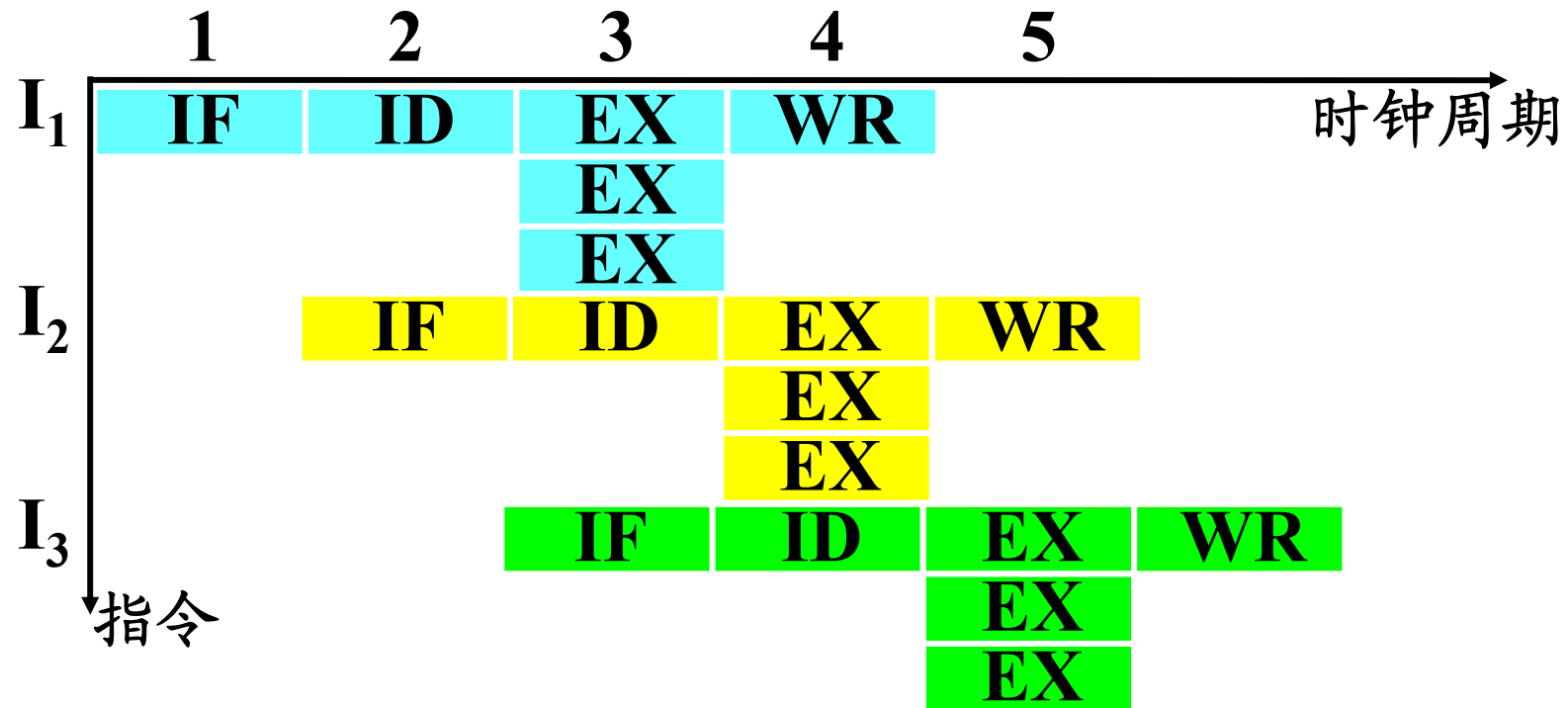
## 5.4.5 超长指令字处理机

### ■ 特点:

- 指令字长很长，可达数百位；
- 有多个功能部件并发工作；
- 用一条长指令来实现多个操作的并行执行；



## 5.4.5 超长指令字处理机



超长指令字处理机流水线时空图

每拍启动1条指令，要求并行度=3



## 5.4.2 超长指令字处理机

### ■ 优点

- 每条指令所需拍数比超标量处理机少
- 指令译码容易
- 开发标量操作间的随机并行性更方便

### ■ 缺点

- 取决于压缩的效率
- 其结构的目标码与一般的计算机不兼容
- 指令字很长，但操作段各是固定，容易浪费存贮空间

# 例题

- 在下列不同结构的处理器上运行 $8 \times 8$ 的矩阵乘法  $C=A \times B$ ，计算所需要的最短时间。
- 只计算乘法指令和加法指令的执行时间，不计算取操作数、数据传送和程序控制等指令的执行时间。
- 加法部件和乘法部件的延迟时间都是3个时钟周期，另外，加法指令和乘法指令还要经过一个“取指令”和“指令译码”的时钟周期，每个时钟周期为20ns，C的初始值为“0”。
- 各操作部件的输出端有直接数据通路连接到有关操作部件的输入端，在操作部件的输出端设置有足够容量的缓冲寄存器。

# 例题

- 1.** 处理器内只有一个通用操作部件，采用顺序方式执行指令。
- 2.** 单流水线标量处理器，有一条两个功能的静态流水线，流水线每个功能段的延迟时间均为一个时钟周期，加法操作和乘法操作各经过**3**个功能段。
- 3.** 多操作部件处理器，处理机内有独立的乘法部件和加法部件，两个操作部件可以并行工作。只有一个指令流水线，操作部件不采用流水线结构。
- 4.** 单流水线标量处理器，处理机内有两条独立的操作流水线，流水线每个功能段的延迟时间均为一个时钟周期。

# 例题

5. 超标量处理器，每个时钟周期同时发射一条乘法指令和一条加法指令，处理机内有两条独立的操作流水线，流水线的每个功能段的延迟时间均为一个时钟周期。
6. 超流水线处理器，把一个时钟周期分为两个流水级，加法部件和乘法部件的延迟时间都为**6**个流水级，每个时钟周期能够分时发射两条指令，即每个流水级能够发射一条指令。
7. 超标量超流水线处理器，把一个时钟周期分为两个流水级，加法部件和乘法部件延迟时间都为**6**个流水级，每个流水级能够同时发射一条乘法指令和一条加法指令。

# 例题解答

要完成两个 $8 \times 8$ 矩阵相乘，共要进行 $8 \times 8 \times 8 = 512$ 次乘法， $8 \times 8 \times 7 = 448$ 次加法。典型的C代码如下：

```
int k;
for(int i=0; i<8; i++)
    for(int j=0; j<8; j++)
    {
        sum=0;
        for(k=0; k<8; k++)
        {
            sum+=A[i][k]×B[k][j]
        }
        C[i][j]=sum;
    }
```

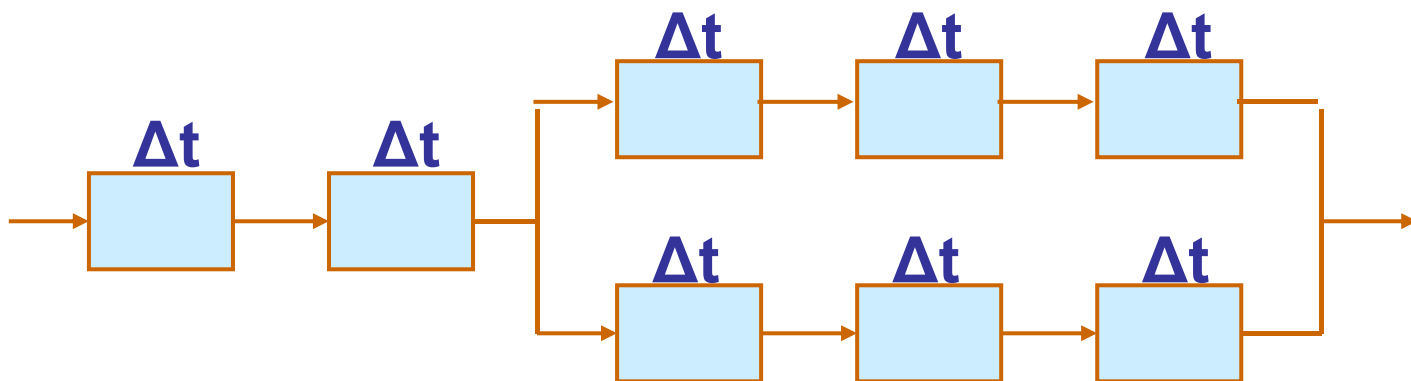
## 例题解答

- 1. 顺序执行时，每个乘法和加法指令都需要5个时钟周期，计算所需要的时间为：**

$$T = (512 + 448) \times 5 \times 20 \text{ ns} = 96000 \text{ ns}$$

- 2. 单流水线标量处理机，采用两功能静态流水线，结构如下。**
- 因为有足够的缓冲寄存器，所以我们可以首先把所有的乘法计算完，排空后再通过调度使加法流水线不出现停顿。计算所需要的最短时间为：

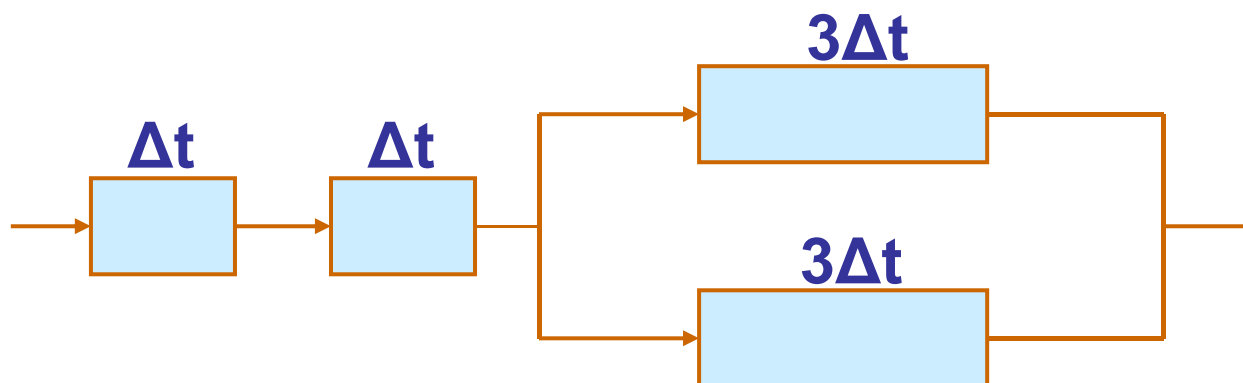
$$T = [(2+3+512-1) + (3+448-1)] \times 20\text{ns} = 19320\text{ns}$$



# 例题解答

3. 多处理部件处理机，只有一条指令流水线，结构如下图。因为加法总共执行**448**次，而乘法共执行**512**次，因此加法的执行过程一定能与乘法在某些段内并行执行，同时要考虑可能出现的乘法流水线的乘积与加法流水线的加数之间的“先写后读”数据相关。由于存在数据相关，最后一次加法运算结束与最后一次乘法运算结束的时间差为一次完整的加法流水线操作过程，为**3**个时钟周期。计算所需要的最短时间为：

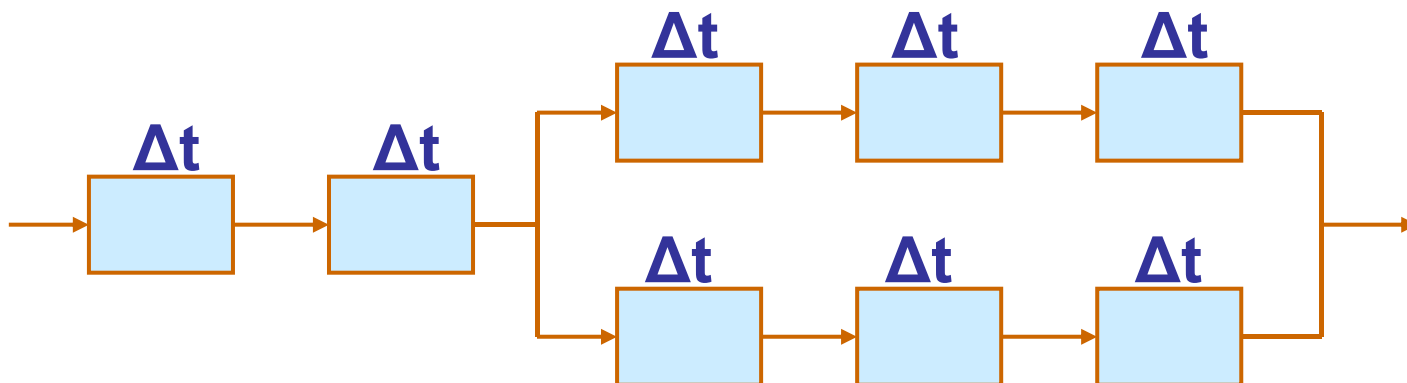
$$T=[2+3+(512-1)\times 3+3]\times 20\text{ns}=30820\text{ns}$$



# 例题解答

4. 单流水线标量处理机，有两条独立的操作流水线，结构如下图。  
（分析方法同2），但不需进行排空处理。计算所需要的最短时间为：

$$T=[5+(512-1)+3]\times 20\text{ns}=10380\text{ns}$$

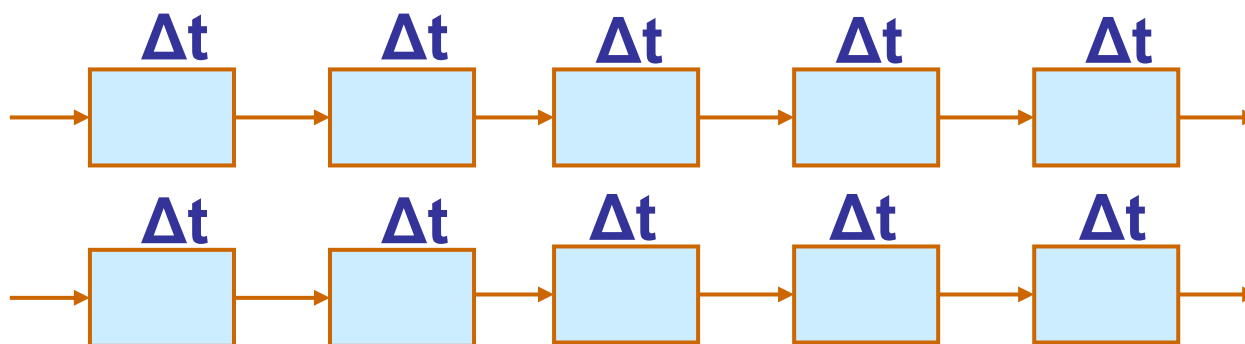




# 例题解答

5. 超标量处理机，能同时发射一条加法和一条乘法指令，有两条独立的操作流水线，结构如下图。（分析方法同3），不同之处在于乘与加操作均是流水化，且是在不同的流水线上并行执行的。计算所需要的最短时间为：

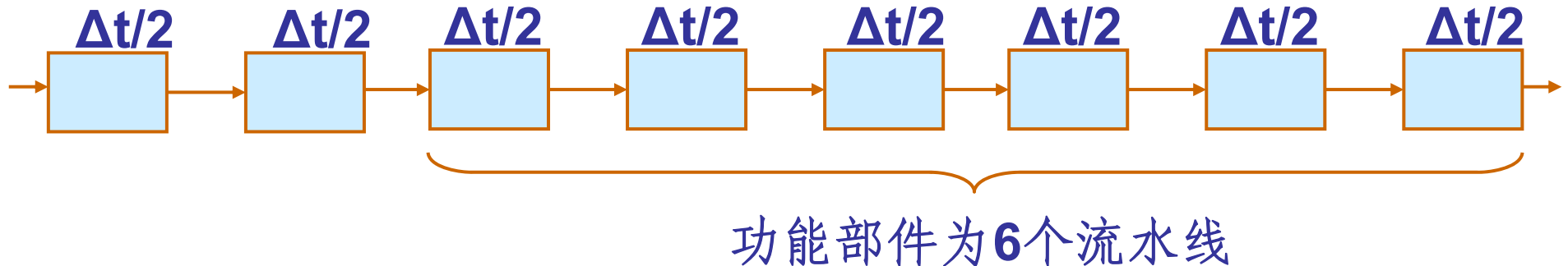
$$T=[2+3+(512-1)+448]\times 20\text{ns}=19280\text{ns}$$



# 例题解答

6. 超流水线处理机，每个时钟周期分时发射两条指令，加法部件和乘法部件都为6个流水线，‘取指令’和‘指令译码’仍分别为一个流水级，结构如下图。（分析方法同4），不同之处在于时钟周期变成了**10ns**，且流水线已细化。计算所需要的最短时间为：

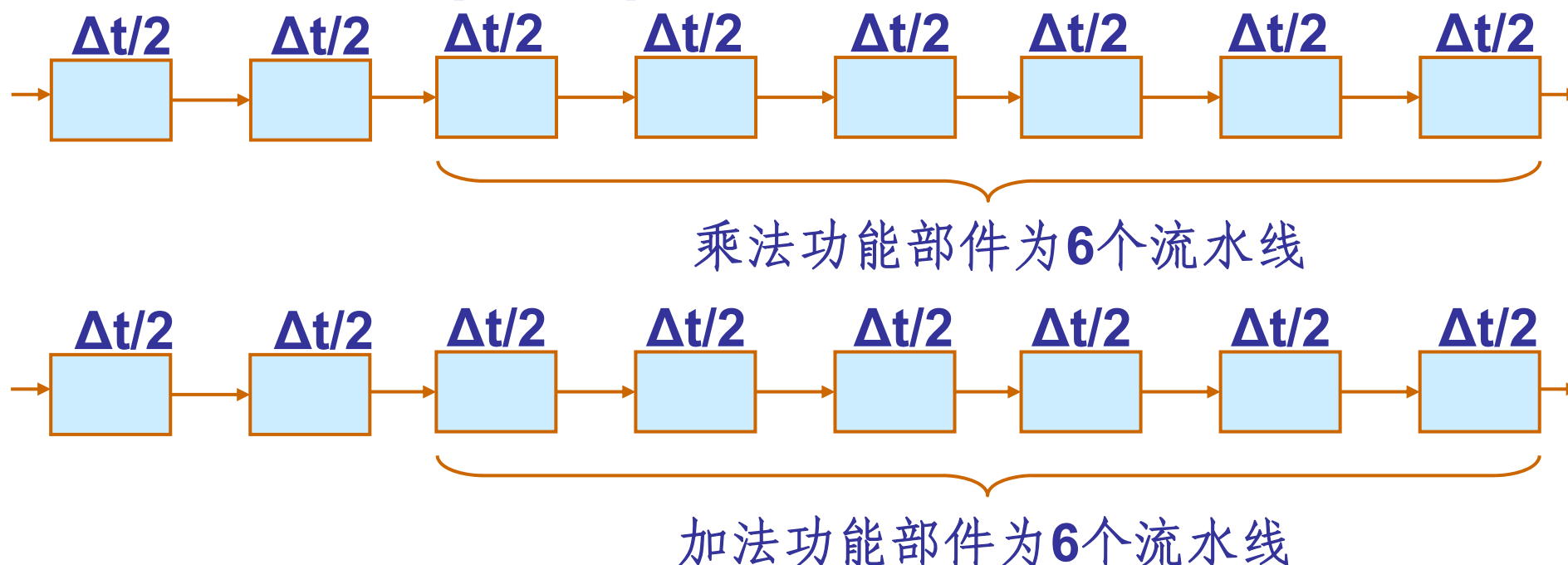
$$T=[2+6+(512-1)+448]\times 10\text{ns}=9670\text{ns}$$



# 例题解答

7. 超标量超流水线处理机，一个时钟周期分为两个流水级，加法部件和乘法部件都为6个流水线，“取指令”和“指令译码”仍分别为一个流水级，每个流水级能同时发射一条加法和一条乘法指令，结构如下图。综合5)和6)的分析，我们可以知道，计算所需要的最短时间为：

$$T = [2 + 6 + (512 - 1) + 6] \times 10\text{ns} = 5250\text{ns}$$



# 本章重点

- 重叠解释方式
- 流水线的分类
- 流水线处理机的主要性能（吞吐率、加速比、效率）
- 流水线的时空图、流水线瓶颈段的处理
- 流水机器的相关处理
- 非线性流水线的调度

# 本章重点

- 向量的流水处理
- 向量流水处理机
- 向量指令之间的链接技术
- 单发射，多发射
- 超标量处理机的指令执行时序及性能
- 超流水线处理机的指令执行时序及性能
- 超标量超流水线处理机的指令执行时序及性能

# 第5章 作业5