第五章 标量处理机

Dr. Feng Li

fli@sdu.edu.cn

https://funglee.github.io

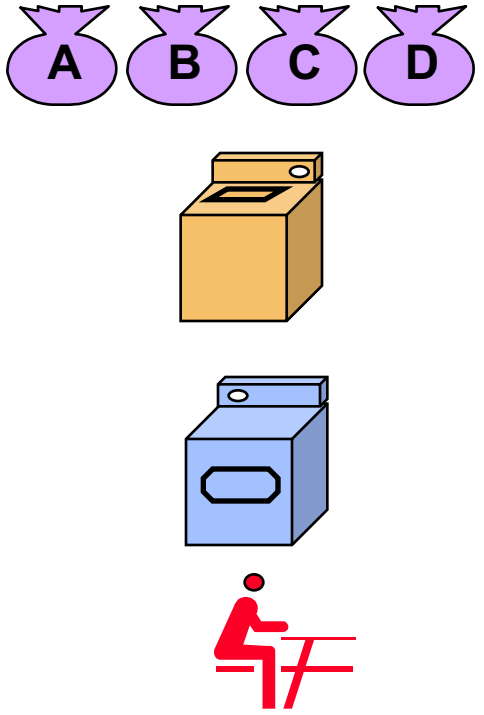
加速机器语言解释的两种方式

- 通过选用更高速的器件，采用更好的运算方法、提高指令内各微操作的并行程度，减少解释过程所需要的拍数，以加快每条指令的解释。
- 通过控制机构采用同时解释两条、多条以至整段程序的控制方式，加快整个机器语言程序的解释。

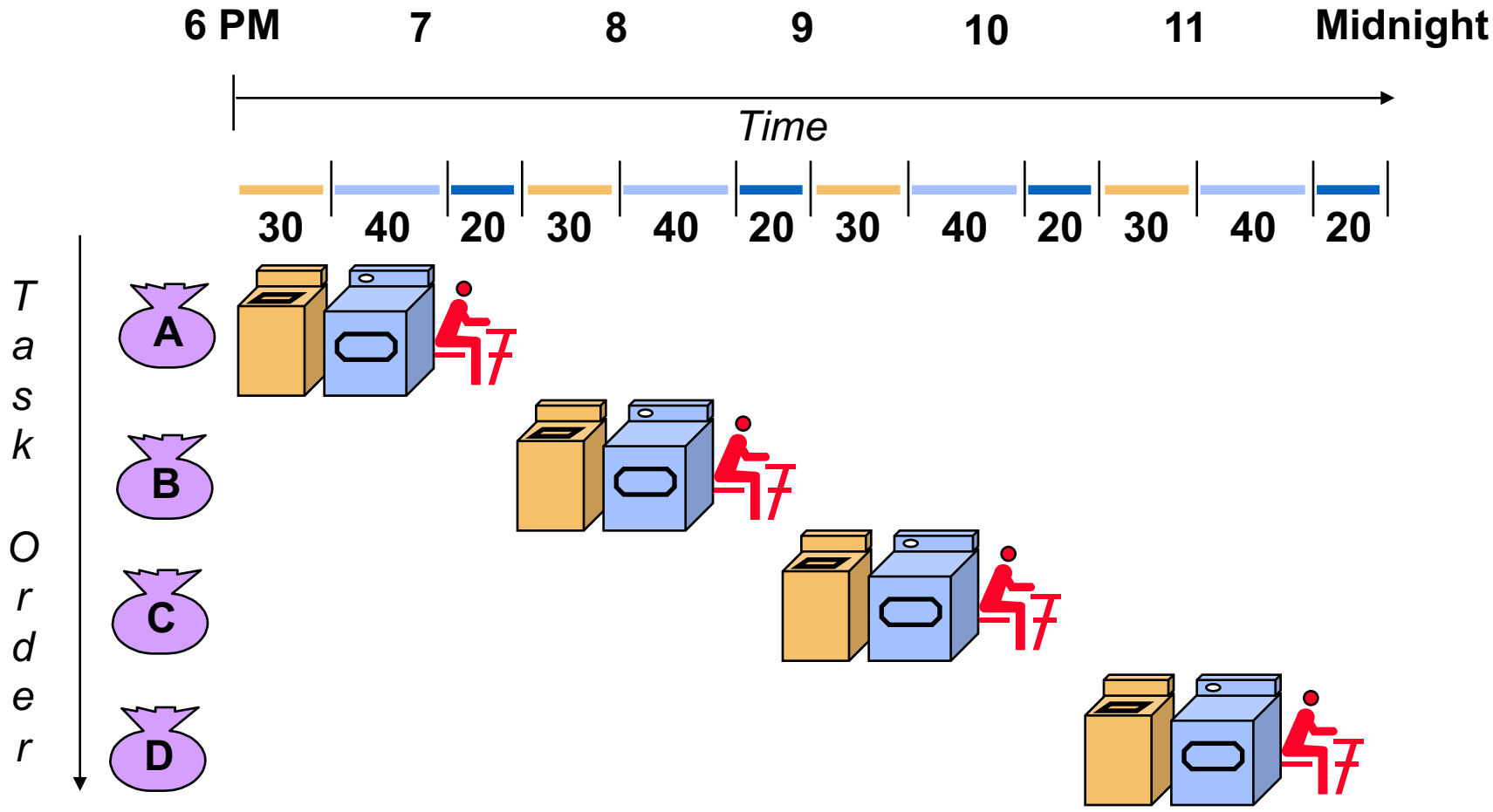
- 重叠、流水。

What Is Pipelining

- Laundry（洗衣） Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes



What Is Pipelining

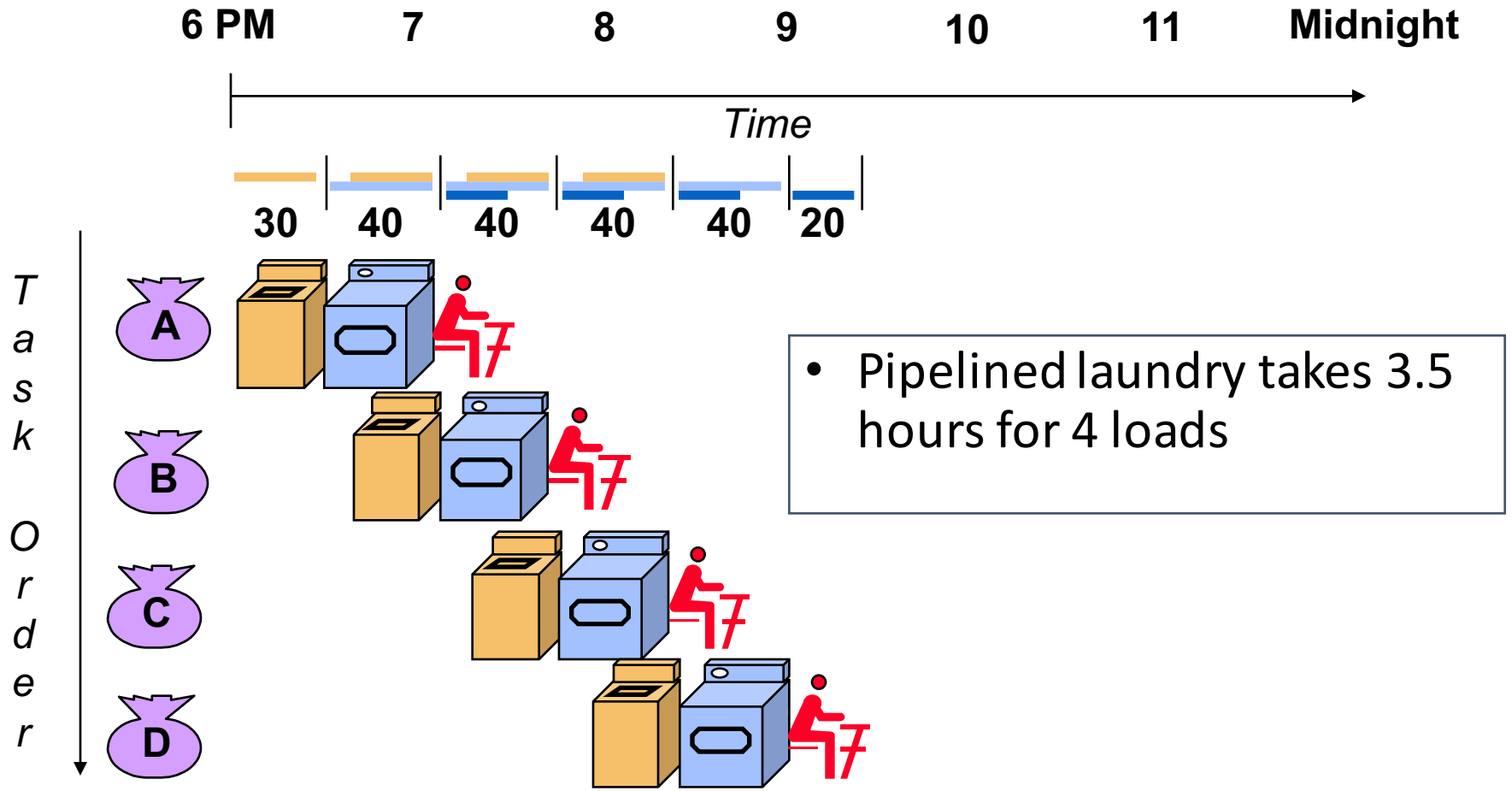


Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

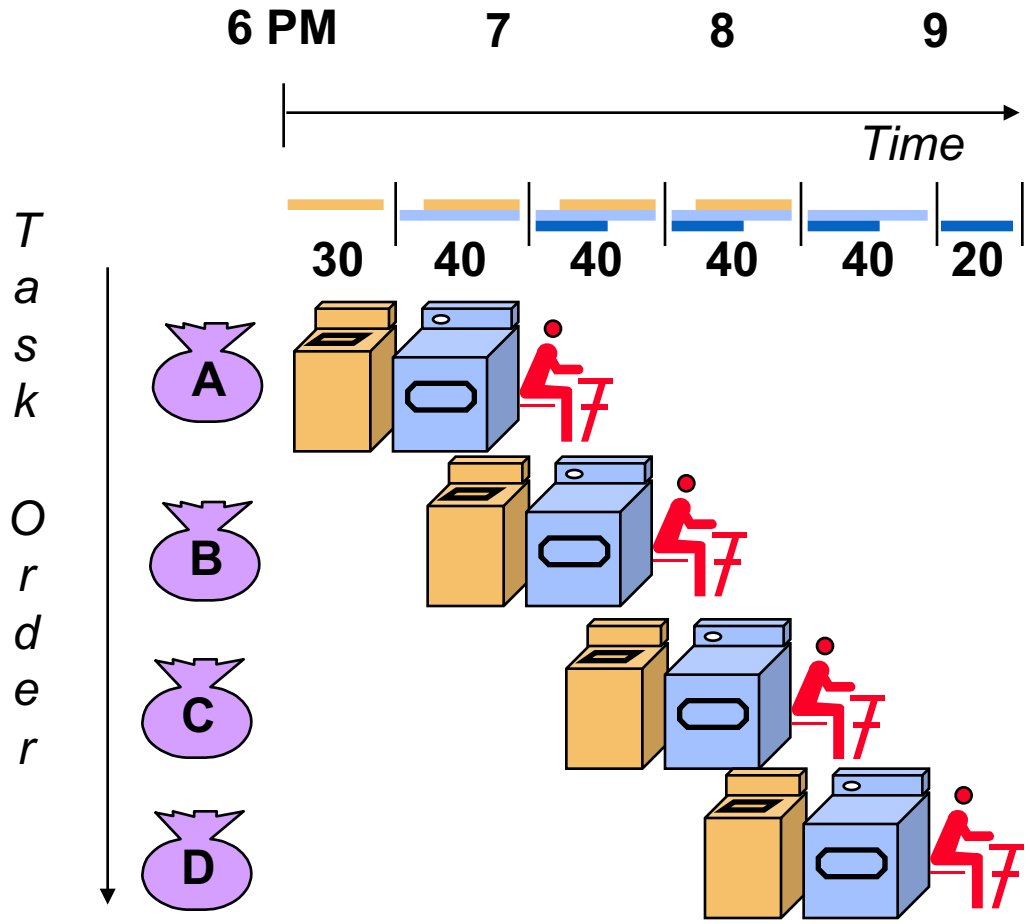
What Is Pipelining

Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

What Is Pipelining



Pipelining Lessons

- **Pipelining doesn't help** latency **of single task, it helps** throughput **of entire workload**
- **Pipeline rate limited by** slowest **pipeline stage**
- Multiple **tasks operating simultaneously**
- **Potential speedup =** Number pipe stages
- **Unbalanced lengths of pipe stages reduces speedup**
- **Time to "**fill**" pipeline and time to "**drain（排出）**" it reduces speedup**

目录

- 重叠方式
- 流水方式
- 向量的流水处理与向量流水处理机
- 指令级高度并行的超级处理机

顺序解释

- 解释一条机器指令的微操作
  - 取指令：按照计数器的内容访主存，取出该指令送到指令寄存器
  - 分析：对指令的操作码进行译码，按寻址方式和地址字段形成操作数真地址，根据该地址去取操作数，并形成下一条指令的地址
  - 执行：对操作数进行运算、处理，或存储运算结果

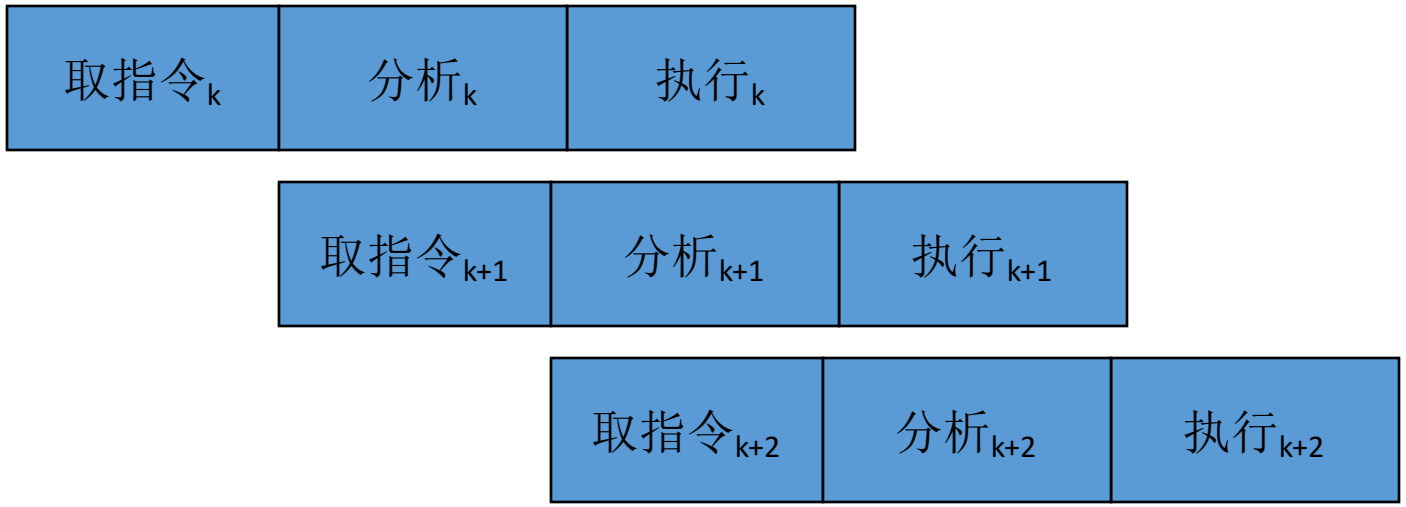
顺序解释

- 顺序解释：各条指令之间顺序串行地进行，每条指令内部的各个微操作也顺序串行执行
- 其优点是控制简单，方便进行时序控制，缺点是利用率低，速度低

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

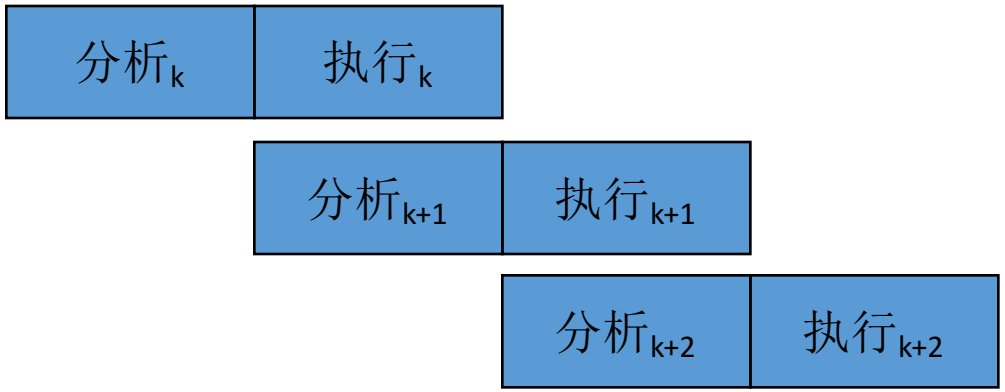
顺序解释与重叠解释

- 重叠解释：在解释第k条指令的操作完成之前，就可开始解释第$k+1$条指令
  - 不能加快一条指令的实现
  - 但能加快相邻两条以至一段程序的解释



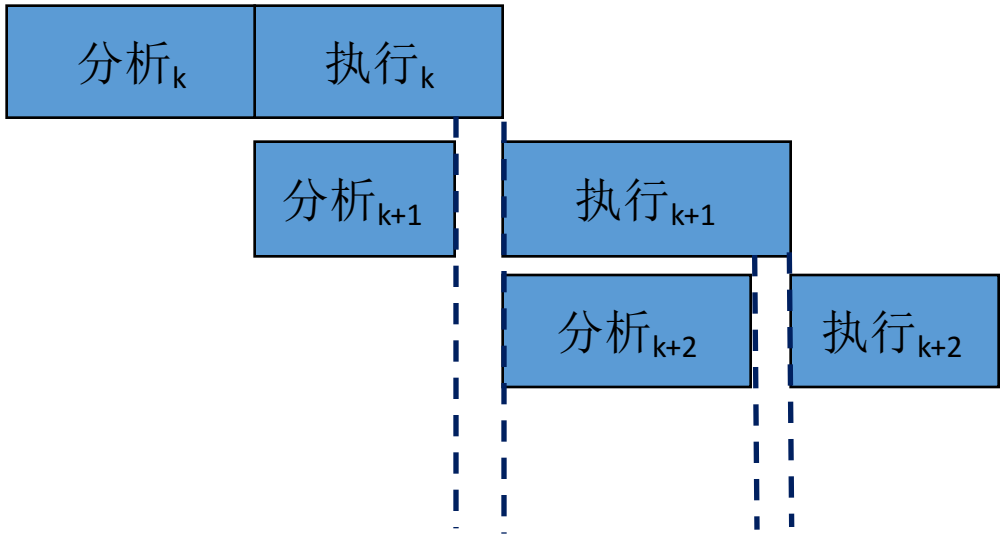
实现指令重叠解释需要满足的要求

- 要解决访主存冲突
  - “取址”和“分析”在时间上重叠，两者都需要访存
  - 由于操作数和指令混合存储于同一主存内，而主存同时只能访问一个存储单元
  - 解决方法1：操作数和指令分别存放于两个独立编址且可同时访问的存储器中，有利于实现指令的保护，但是增加了主存总线控制的复杂性和软件设计的麻烦
  - 解决方法2：采用多体交叉主存结构，当第k条指令的操作数和k+1条指令的不再同一个体内，则可以在一个主存周期内取得，因此仍有一定局限性
  - 解决方法3: 增设采用先进先出方式工作的指令缓冲寄存器（指缓），趁主存有空时，预取下一条或下几条指令存于指缓中



实现指令重叠解释需要满足的要求

- 硬件上应用独立的指令分析部件和指令执行部件，解决“分析”和“执行”操作的并行。
  - 例如，在加法操作中，设置单独的地址加法器用于地址计算，执行部件也有单独的加法器完成操作数相加
- 解决“分析”和“执行”操作控制上的同步
  - “分析”和“执行”所需要的时间不同，需要在硬件中解决控制上的同步问题
  - 保证任何时候都只是“执行k”与“分析$k+1$”重叠执行
  - 一次重叠：指令分析部件和指令执行部件任何时候都只有相邻两条指令在重叠解释
  - 节省硬件，简化控制

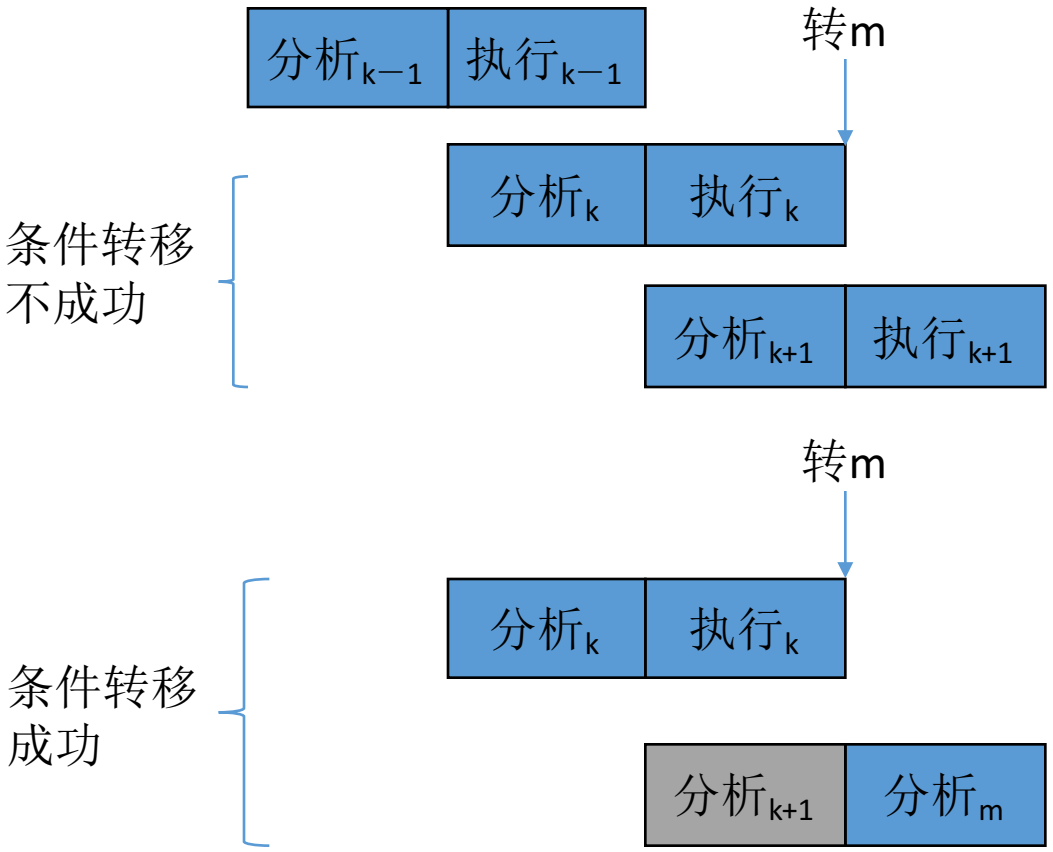


实现指令重叠解释需要满足的要求

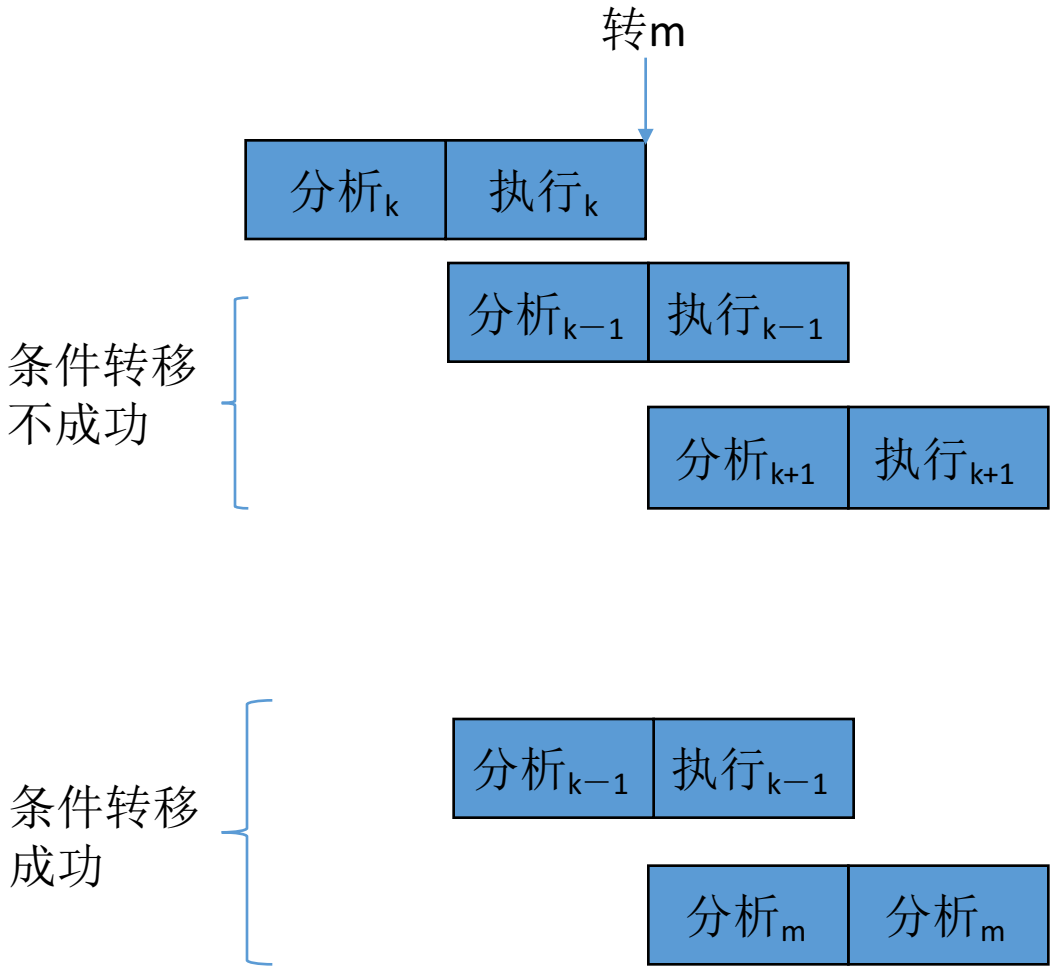
- 解决指令间各种相关问题
  - 数据相关：在执行本条指令过程中，如果用到的指令、操作数、变址偏移量等正好是前面指令的执行结果，则必须等待前面的指令完成，并把结果写到主存或者通用寄存器之后，本条指令才能开始。
    - 指令相关
    - 主存操作数相关
    - 通用寄存器相关
  - 控制相关：由条件分支指令、转子程序指令、中断等引起的相关

转移指令的处理

- 延迟转移技术



条件转移成功时成了顺序解释



延迟转移

指令相关的处理

- 若第k+1条指令本身的内容取决于第k条指令的执行结果，则产生指令相关

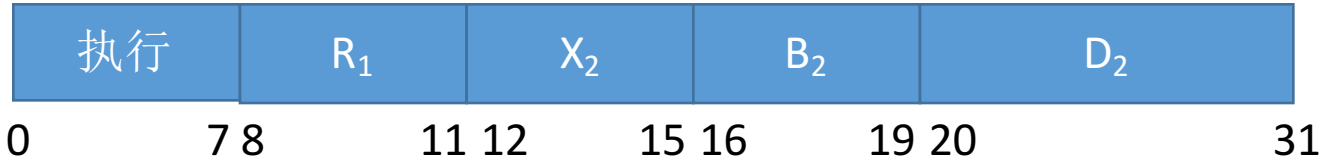
k:　STORE　　R1, k+1

k+1:　　… …

结果地址(k)＝指令地址(k+1)

- 解决方法：在程序执行过程中，不准修改指令，除可解决指令相关，也可实现程序的可再入性和程序的递归调用，也有利于指令的调试和诊断

- 在IBM 370中，有一条“执行”指令既能够解决指令相关，又允许在程序执行过程中修改指令



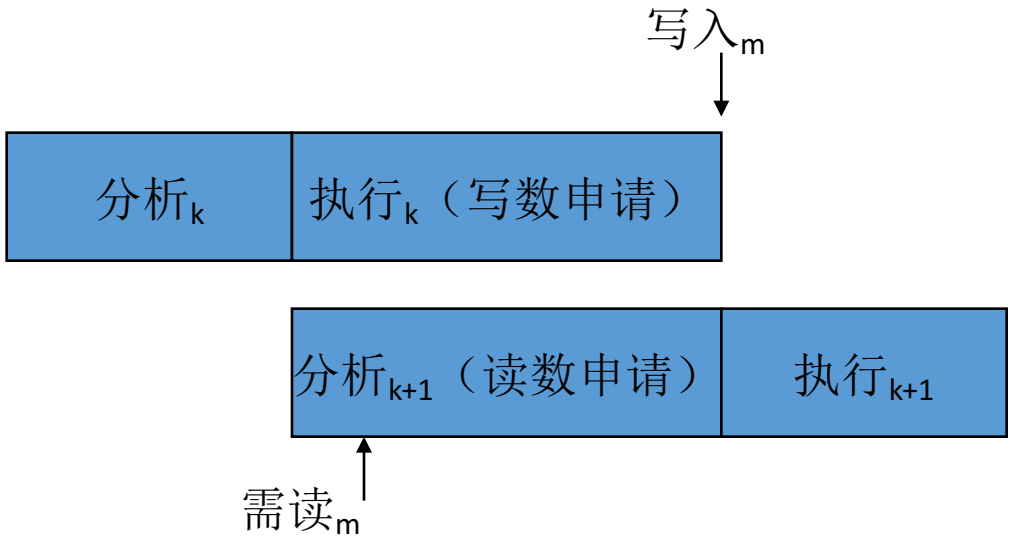
- 该指令执行的是由第二地址$(X_2)+(B_2)+D_2$决定的主存单元中的指令，这个主存单元在数据区而不是指令区
- 程序执行过程中，可先修改这条位于数据区的指令，再执行“执行”指令
- “执行”指令在执行数据区指令的时候，还要用第一地址(R_1)指定的通用寄存器的后8位，与位于数据区的即将被执行的指令的8～15位相“或”

主存空间数相关的处理

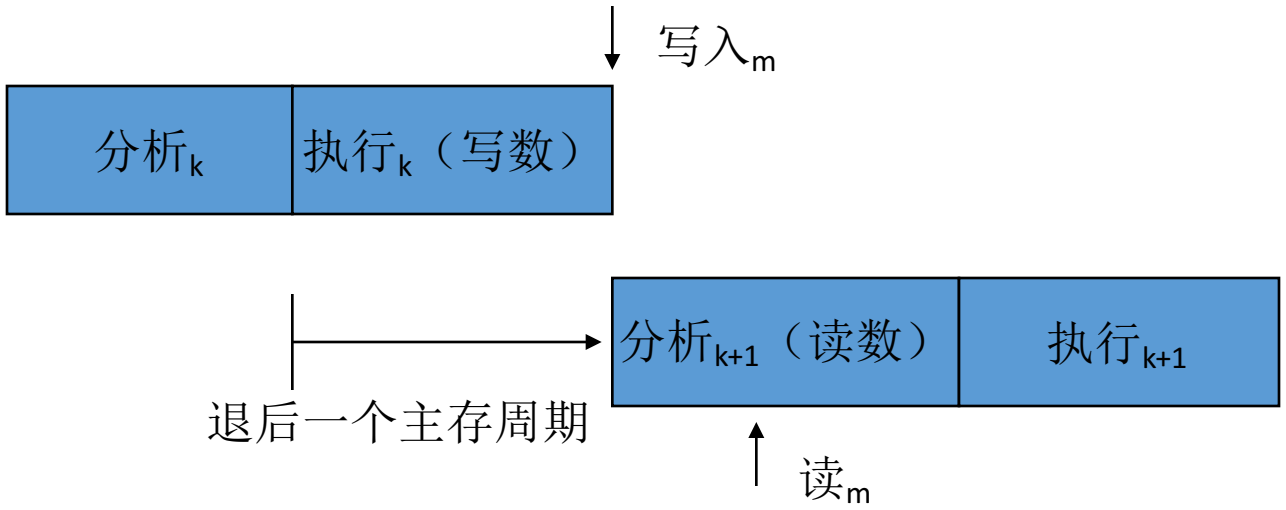
- 相邻两条指令之间要求对主存同一单元先写入而后再读出的关联

结果地址(k) ＝ 主存操作数地址(k+1)

- 解决：延迟。



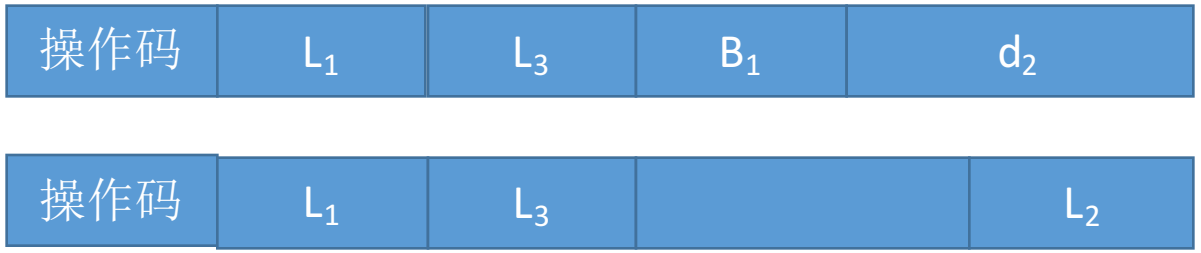
主存数相关的时间关系



主存数相关的处理

通用寄存器数相关处理

- 通用寄存器组相关：操作数相关、变址值或基址值相关

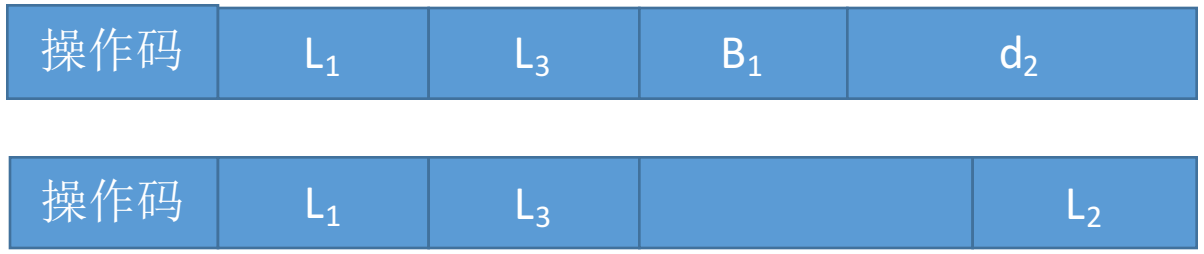


L1、L3 分别是存放第一操作数和结果数的通用寄存器号，B2是基址寄存器号，d2是相对位移

- 延迟和设置“相关专用通路”是解决重叠方式相关处理的两种基本方法。
  - 前者设备不变，降低速度；
  - 后者增加设备，效率不变。

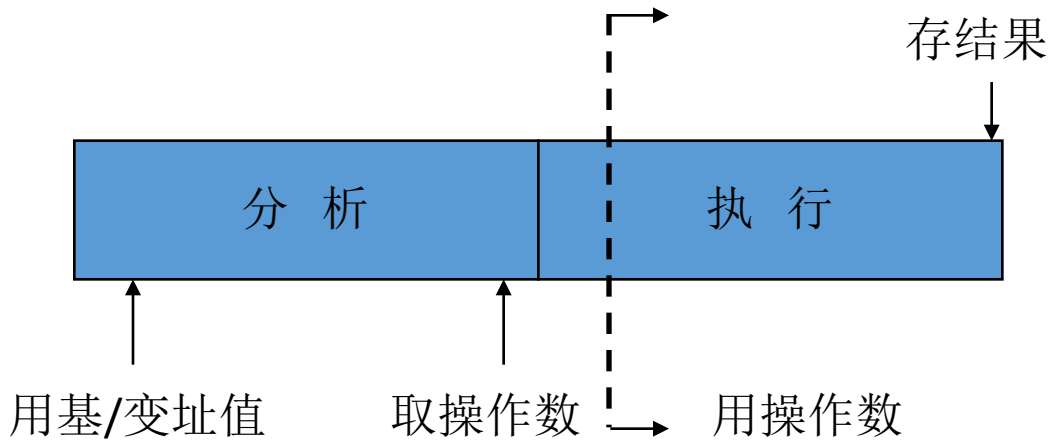
通用寄存器数相关处理

- 通用寄存器组相关：操作数相关、变址值或基址值相关



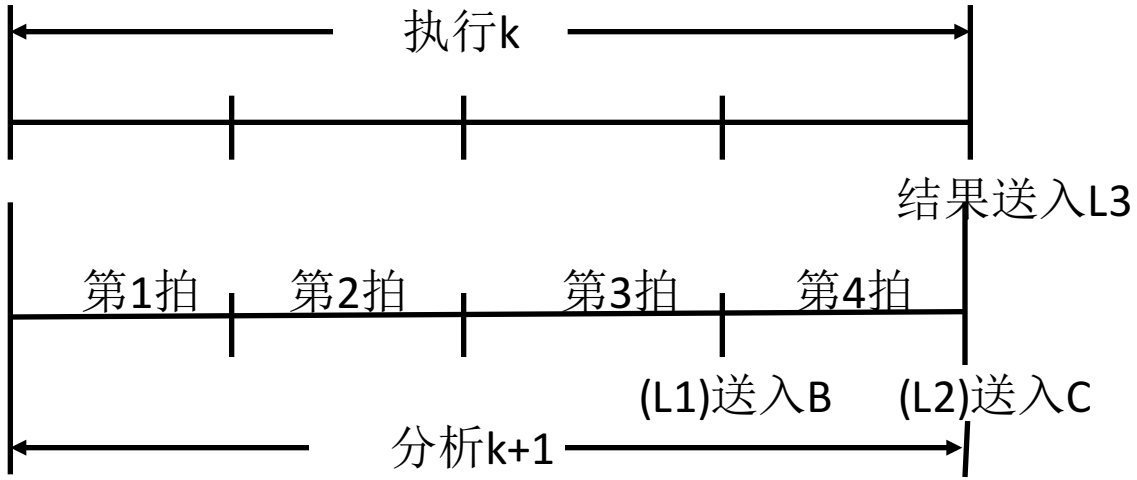
L1、L3 分别是存放第一操作数和结果数的通用寄存器号，B2是基址寄存器号，d2是相对位移

- 使用通用寄存器作不同用途所需要的微操作的时间不同，因而通用寄存器数相关和基址（变址）相关的处理方法不同

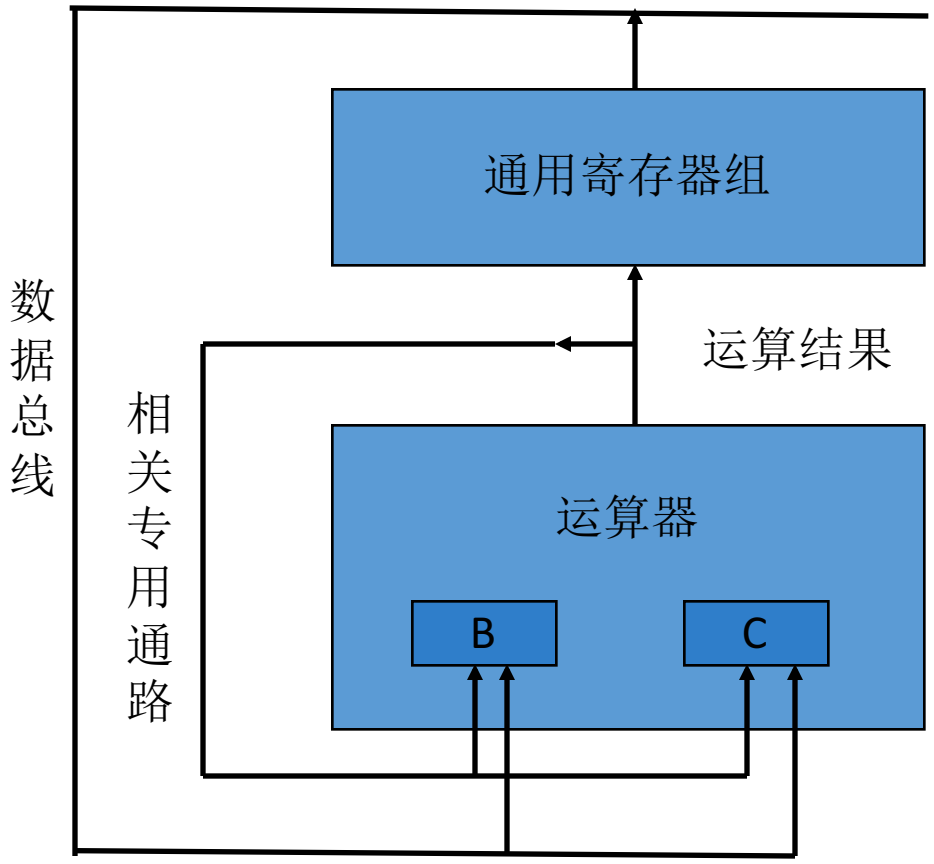


通用寄存器组数相关

- $L_{1(k+1)}=L_{3(k)}$时，L_1相关；$L_{2(k+1)}=L_{3(k)}$时，L_2相关
- 解决方案1：延迟
  - 推后“分析k+1”的读操作到“执行k”结束时开始，也可以推后到“执行k”把结果送入L_3
  - 前者使得重叠变成了完全的串行执行，速度下降明显；而后者虽然允许部分重叠，但增加了控制的复杂度



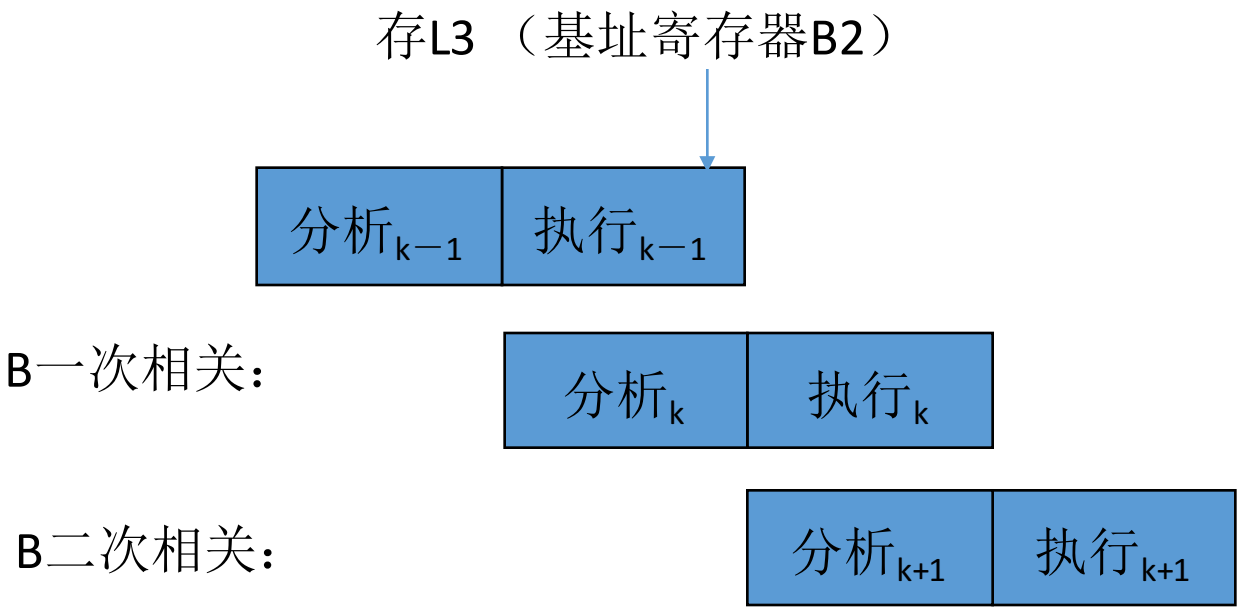
- 解决方法2：相关专用通路
  - 尽管将通用寄存器的旧内容经数据总线分别在“分析k+1”的第3拍或者第4拍末送入了操作数寄存器B或C中，但之后经过相关专用通路在“执行k+1”真正使用它之前，操作数寄存器B或C重新获得第k条指令送来的新结果
  - 保证相关发生时不推后，使重叠效率不下降，又可以保证指令重叠解释时数据不出错



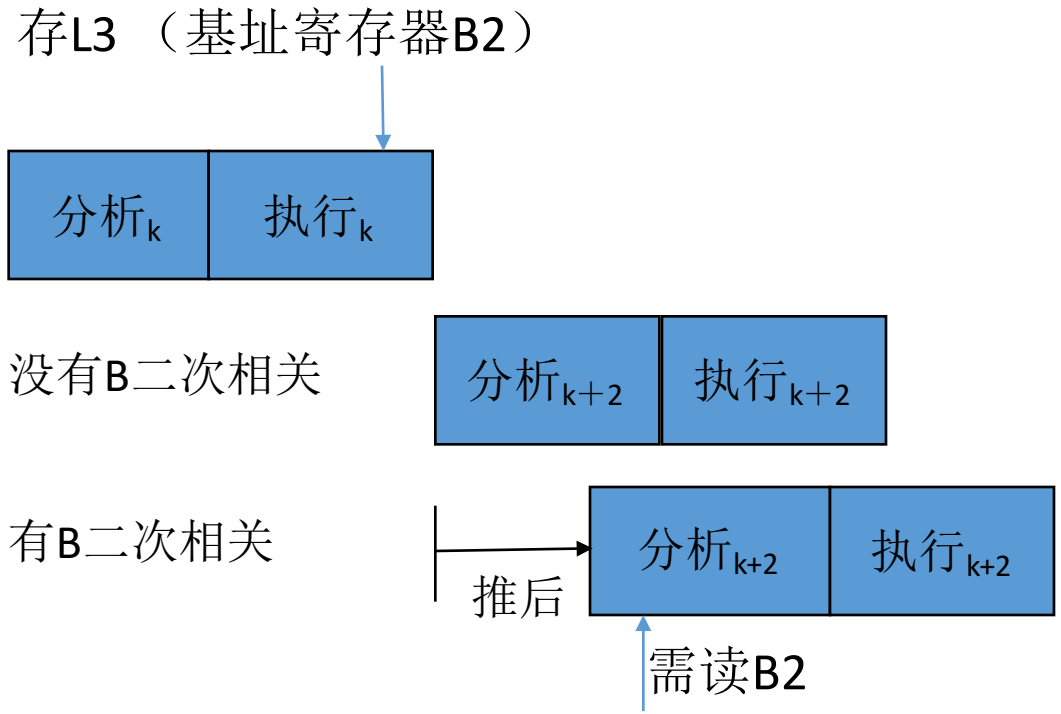
- 推后读法以降低速度为代价，使设备基本不增加；而相关专用通路需要增加设备，但可以使重叠效率不下降
- 若相关概率低，则不宜采用“相关专用通路法”，既节省了设备，也不会使重叠效率有明显的下降
- 由于主存数相关发生的概率比通用寄存器组数相关的概率低得多，因此一般不使用“相关专用通路法”，而采用推后读法

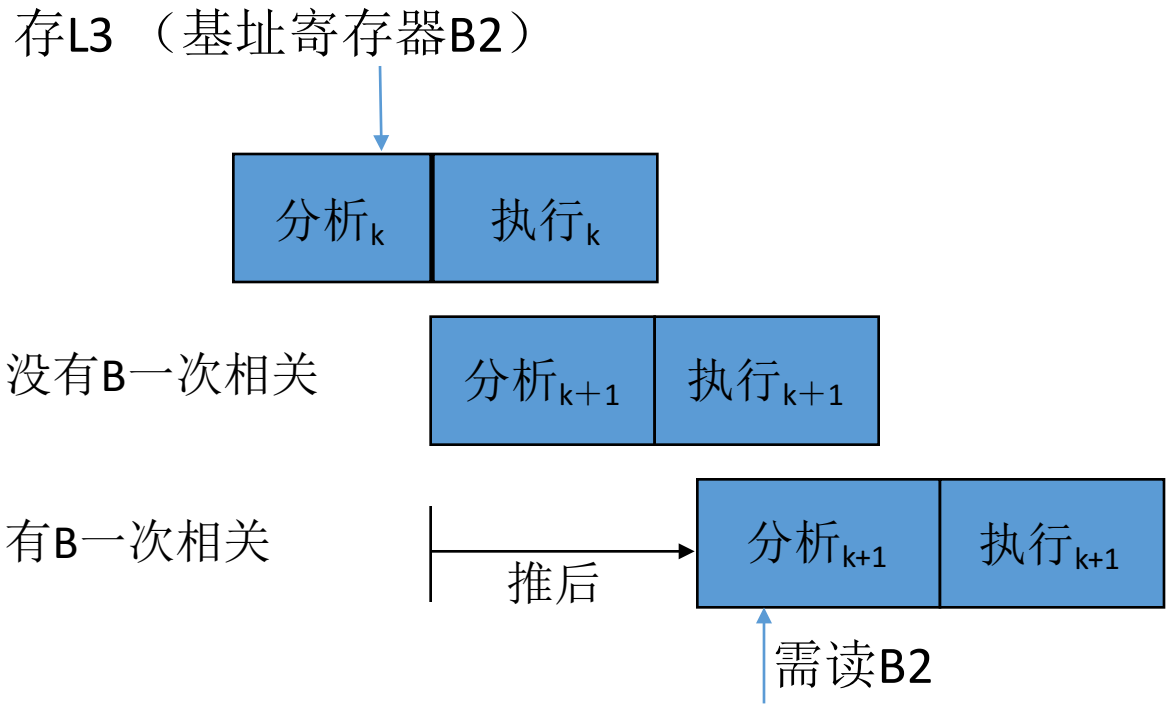
通用寄存器基址或变址相关（以基址为例）

- 假设操作数的有效地址$(X_d)+(B_2)+d_2$由分析器中的地址加法器形成
- 运算结果在“执行”周期的末尾才送入通用寄存器组，不能立刻出现在通用寄存器输出总线上，因此来不及为“分析k+1”和“分析k+2”提供基址值



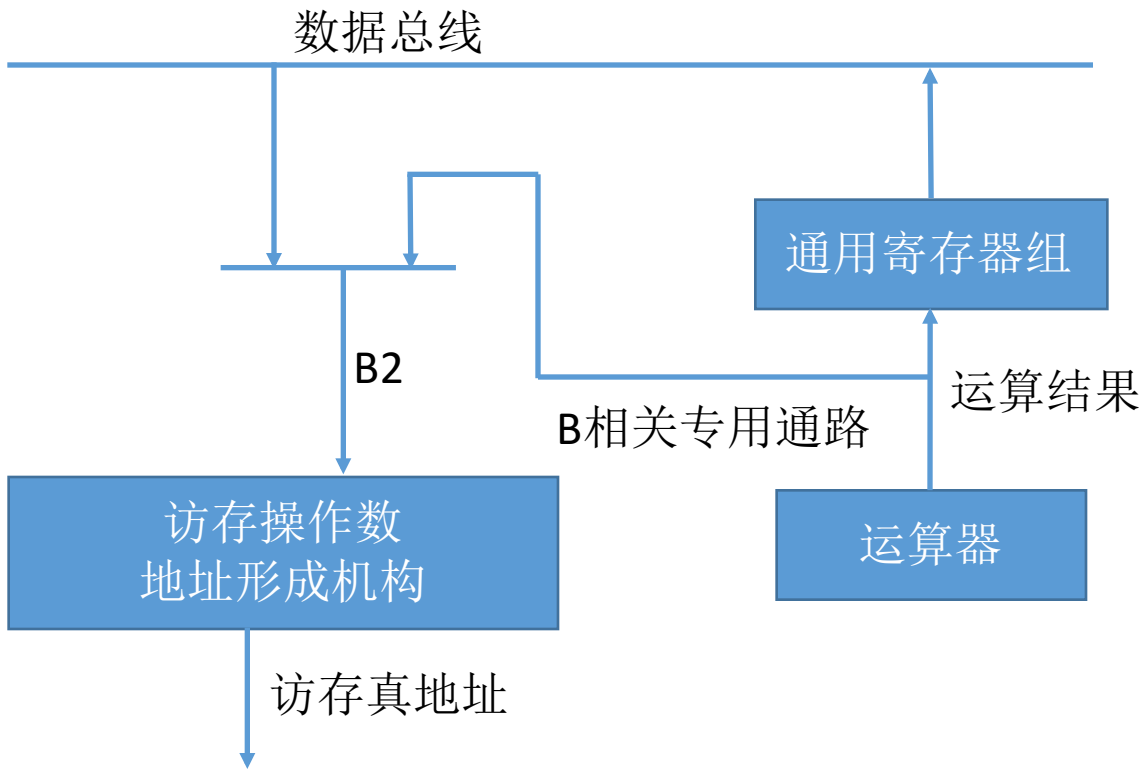
B一次相关与二次相关的推后处理





B相关专用通路法

- 在B二次相关时，“执行k”得到的运算结果在送入通用寄存器组的同时，也经B相关专用通路直接送到访存操作数地址形成机构，缩短传送延迟，使得不用推后“分析k+2”就能使用正确的基址值
- 对于一次相关，仍然需要推后“分析k+1”到“执行k”后开始
- 使用延迟转移技术，调整指令顺序，使之尽可能发生二次相关，不发生一次相关

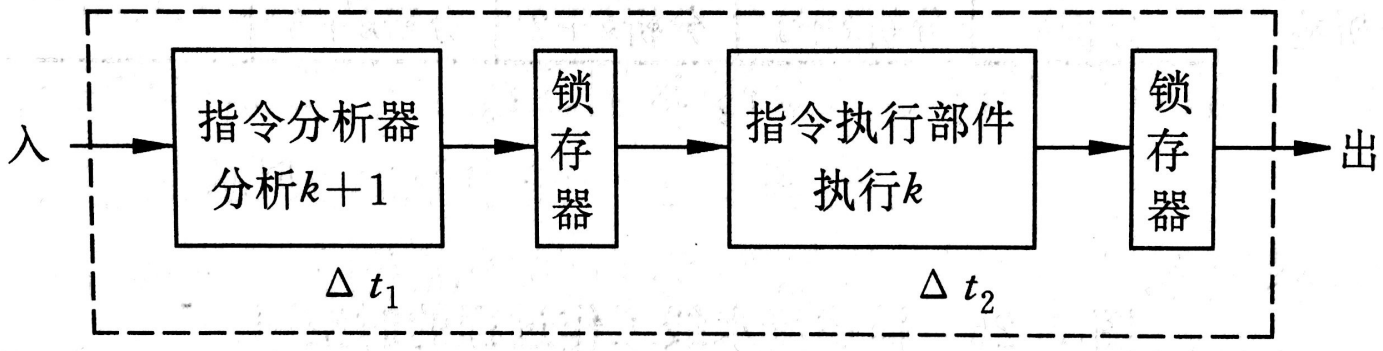


流水方式

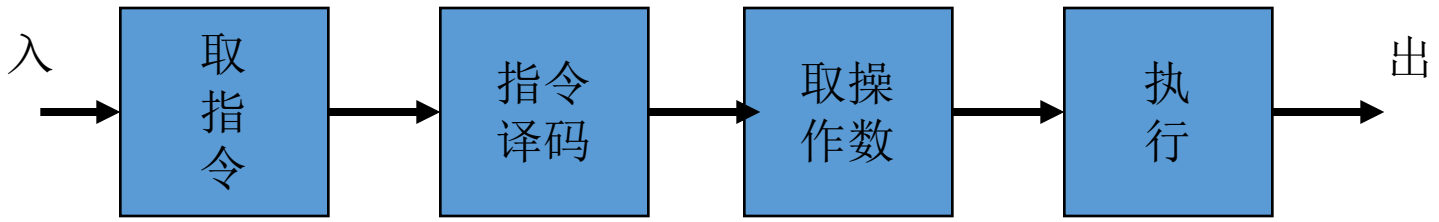
- 基本概念
- 流水线处理机的主要性能
- 流水机器的相关处理和控制机构

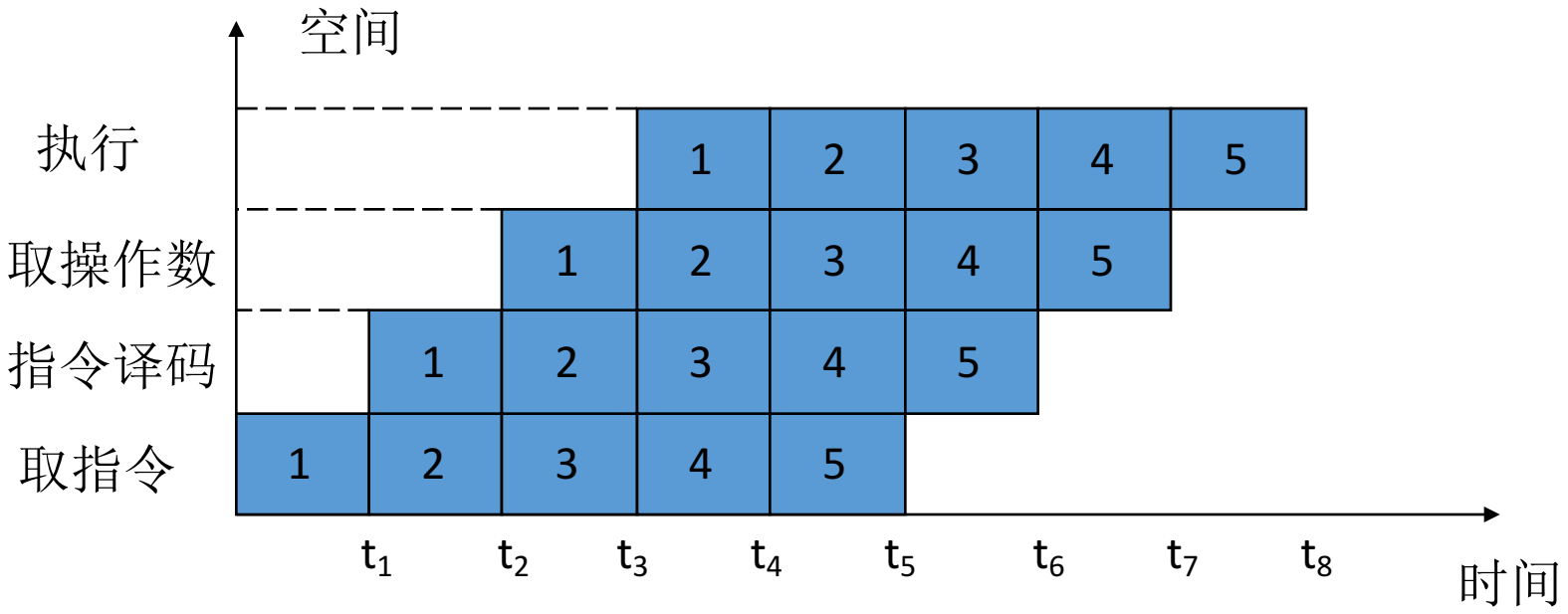
从重叠到流水线

- 采用指缓的处理机中，一条指令的解释可以分解为“分析”和“执行”两个子过程
- 在指令分析器和指令执行部件的输出端各有一个锁存器，可以分别保存指令“分析”和指令“执行”的结果



- 流水将一条指令的解释分解为更多的子过程





- 如能把一条指令的解释分解成时间相等的m个子过程，则每隔$\Delta t = T/m$就可以处理一条指令
- 流水的最大吞吐率取决于子过程的经过时间Δt， Δt越小，流水的最大吞吐率（即流水线满负荷时每隔Δt 流出一个结果所达到的吞吐率）就越高
- 由于流水线从开始启动到流出第一个结果，需要经过一段流水线建立时间，因此实际吞吐率总是低于其最大吞吐率
- 部件之间设有锁存器，受同一时钟信号控制，以实现各子部件信息流的同步推进
- 时钟信号周期不得低于速度最慢子部件的经过时间与锁存器的存取时间之和，还要考虑时钟信号到各锁存器可能存在的时延
- 子过程的细分会因锁存器数增多而增大任务或指令流过流水线的时间，这在一定程度上会抵消子过程细分使吞吐率提高的好处

流水线的特点

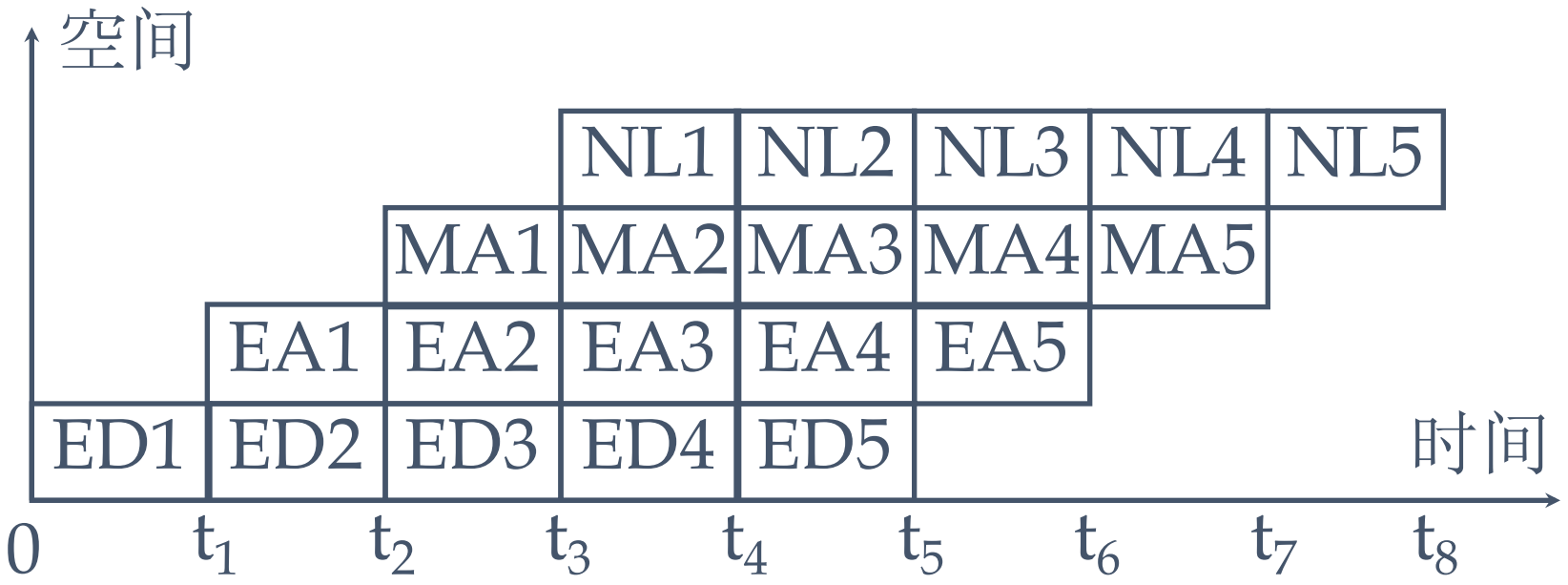
- 在流水线中处理的必须是连续任务，只有连续不断地提供任务才能充分发挥流水线的效率
- 把一个任务（一条指令或一个操作）分解为几个有联系的子任务，每个子任务由一个专门的功能部件来实现
- 在流水线的每一个功能部件后面都要有一个锁存器，用于保存本段的执行结果
- 流水线中各段的时间应尽量相等，否则将引起“堵塞”、“断流”等
- 流水线需要有“装入时间”和“排空时间”

流水线的分类

- 从不同角度，有不同的分类
- 依据向下扩展和向上扩展思路，可分类出在计算机系统不同等级上使用的流水线
  - 向下扩展：子过程细分
  - 向上扩展：多个处理机之间进行流水

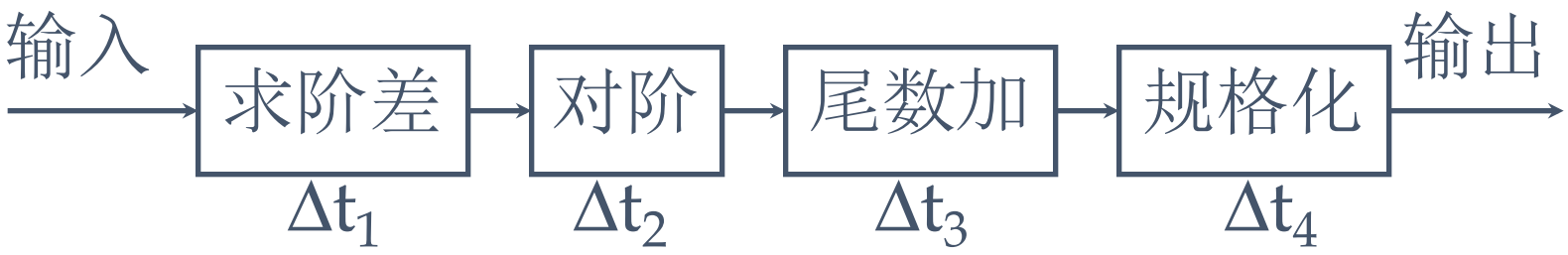
向下扩展：一个浮点加法器流水线的时空图

- 由求阶差、对阶、尾数加和规格化4个流水段组成



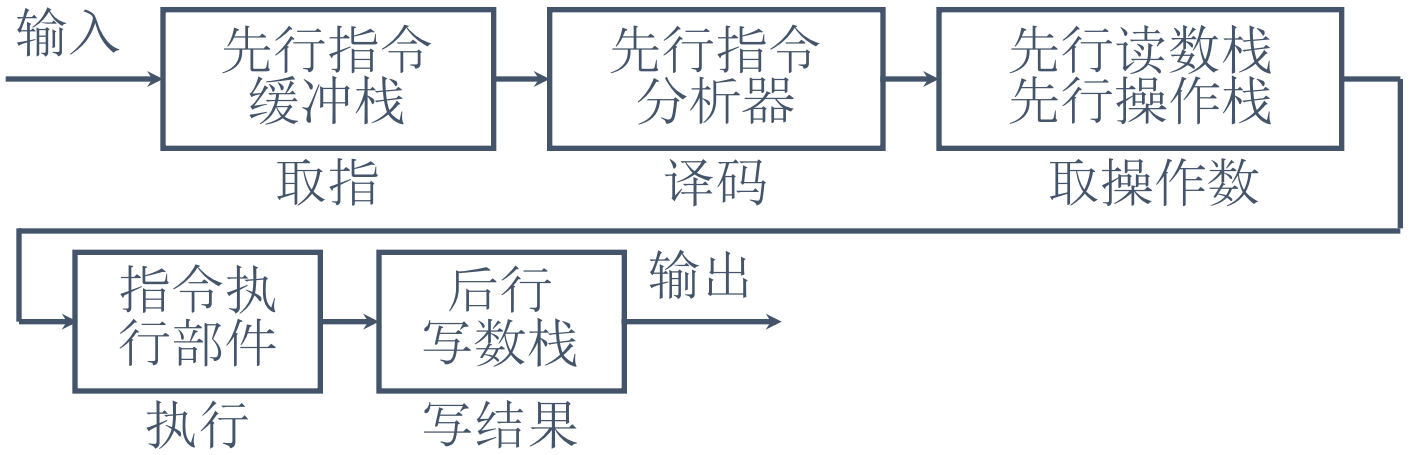
流水线的分类（按流水处理级别）

- 功能部件级流水线也称为运算操作流水线（arithmetic pipelining）
- 每一个部件内部也可以采用流水线来实现
- 例如，一些比较复杂的运算操作部件（如浮点加法器、浮点乘法器等），一般要采用多级流水线来实现



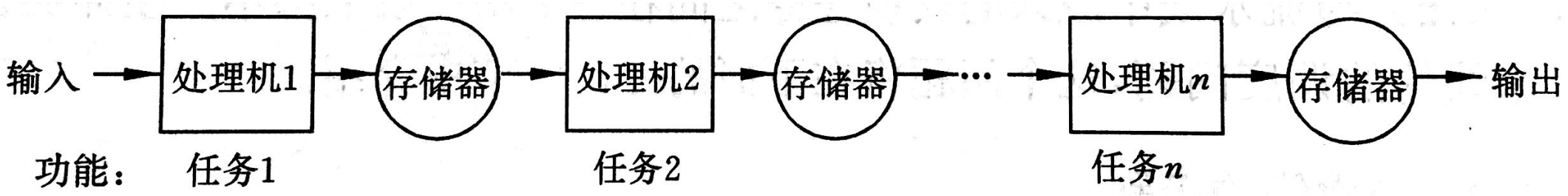
流水线的分类（按流水处理级别）

- 处理机级流水线也称指令流水线 (Instruction Pipelining)
- 一条指令的执行过程分解为多个子过程，每个子过程在一个独立的功能部件中完成



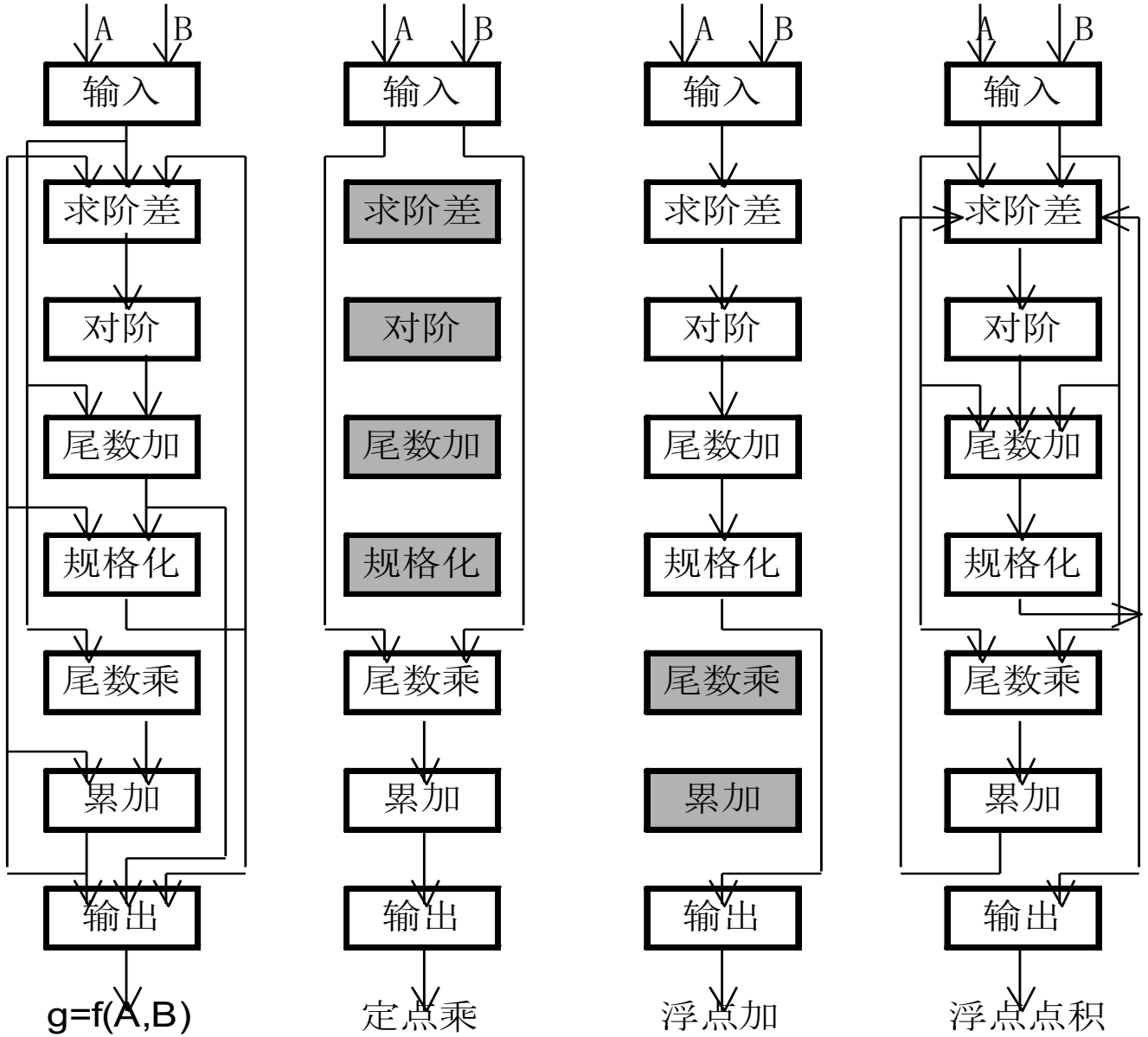
流水线的分类（按流水处理级别）

- 处理机间流水线也称为宏流水线（macro pipelining）
- 由两个或两个以上处理机通过存储器串行连接起来，每个处理机对同一个数据流的不同部分分别进行处理，使得前一个处理机的输出结果存入存储器中，作为后一个处理机的输入，每个处理机完成整个任务的一部分



流水线的分类（按功能多少）

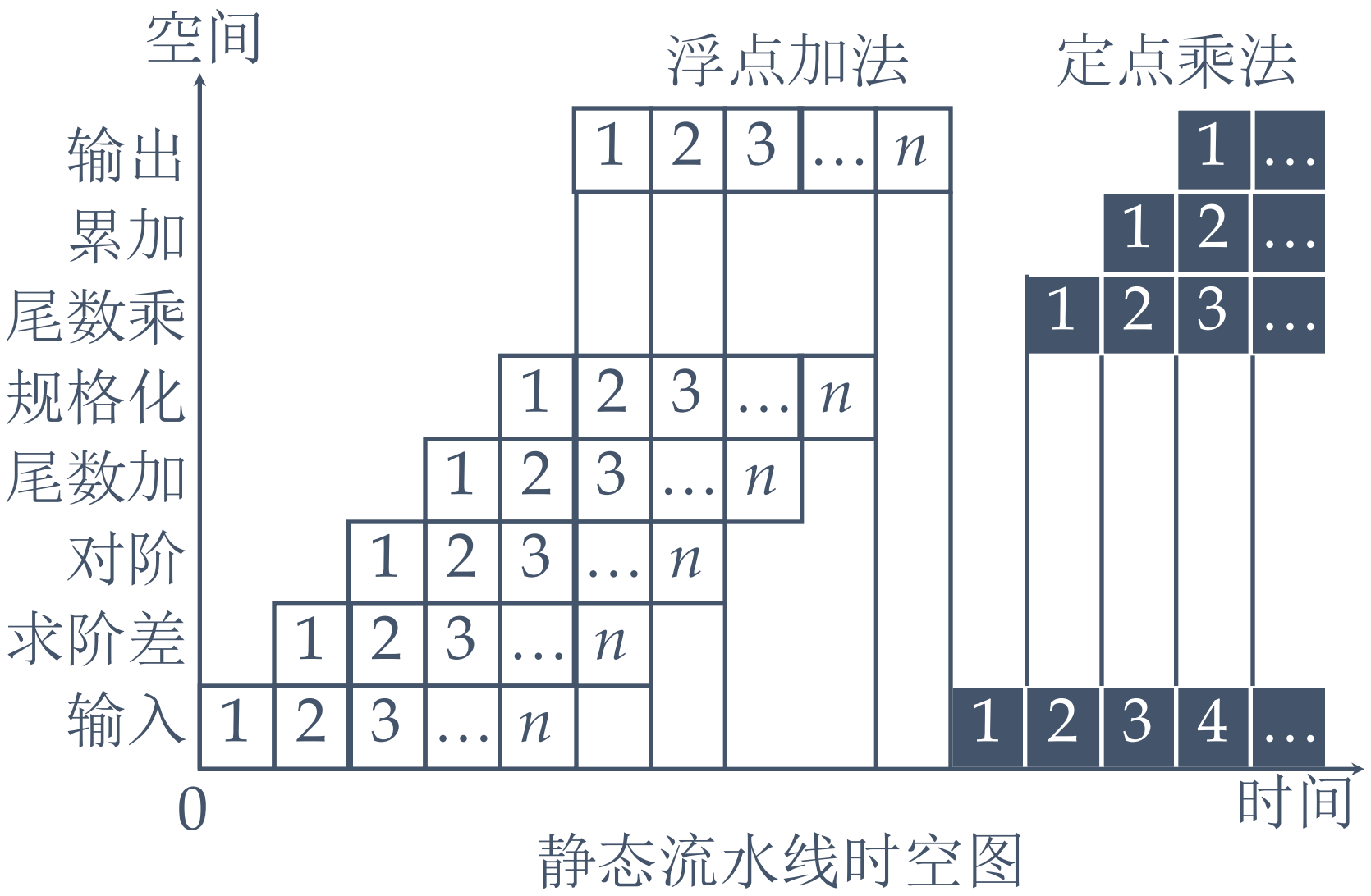
- 单功能流水线（unifunction pipelining）：一条流水线只能完成一种固定的功能
- 多功能流水线（multifunction pipelining）：流水线的各段可以进行不同的连接，在不同时间内，或在同一时间内，通过不同的连接方式实现不同的功能



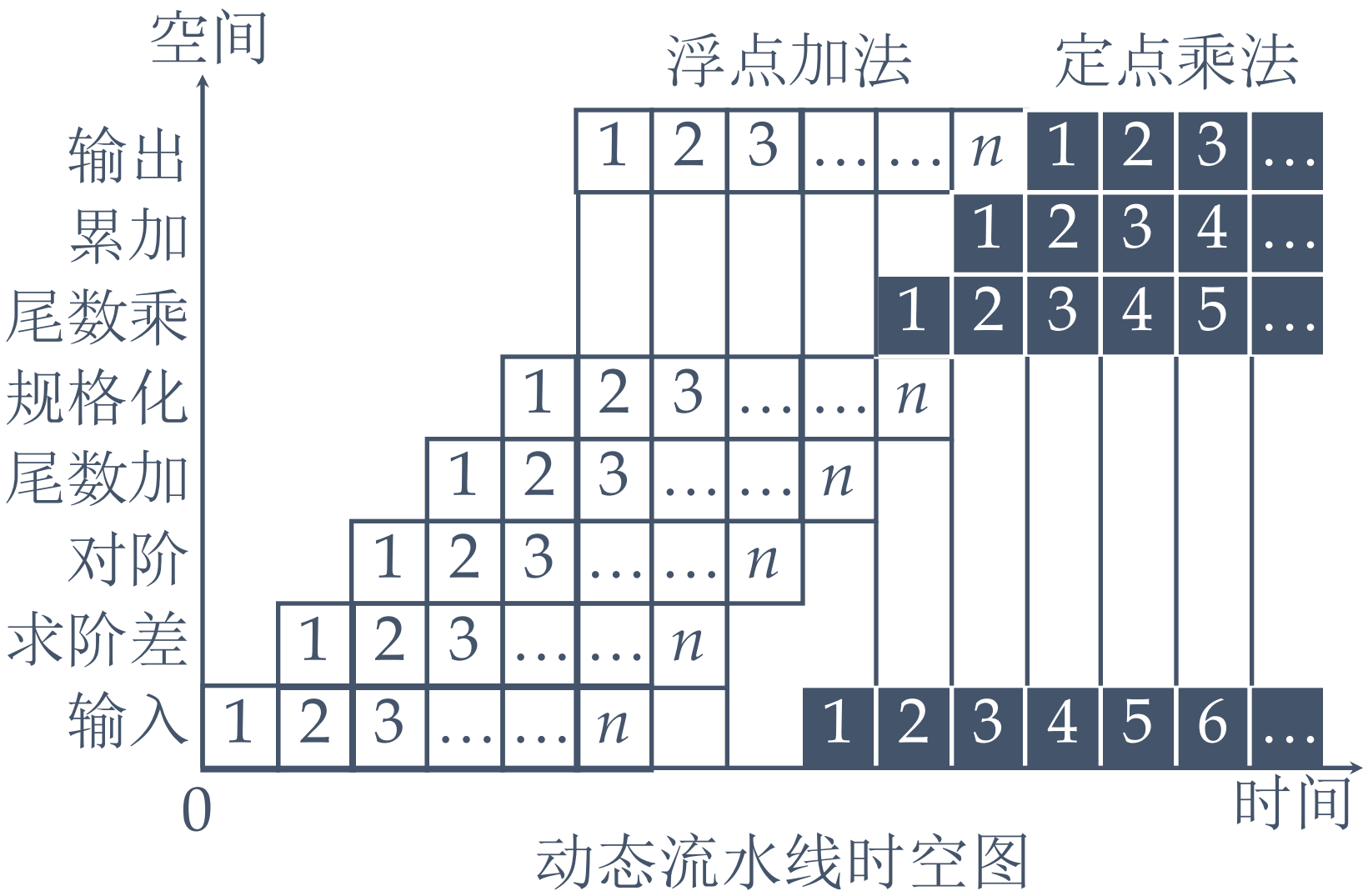
(a) 功能段间的互连　(b) 定点乘法　(c) 浮点加法　(d) 浮点点积

流水线的分类（按多功能的连接方式）

- 静态流水线（static pipelining）：在同一段时间内，多功能流水线中的各个功能只能按照一种固定的方式连接，实现一种固定的功能，只有当按照这种连接方式工作的所有任务都流出流水线之后，多功能流水线才能重新连接，以实现其它功能
- 动态流水线（dynamic pipelining）：在同一段时间内，多功能流水线中的各段可以按照不同的方式连接，在各个功能部件之间不发生冲突的前提下，同时执行多种功能



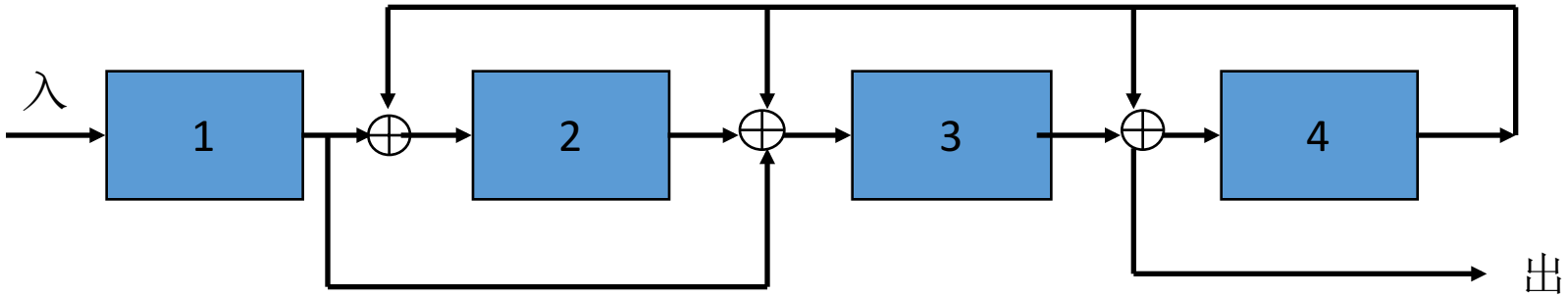
静态流水线时空图


空间
浮点加法
定点乘法
输出
累加
尾数乘
规格化
尾数加
对阶
求阶差
输入
0
时间
1 2 3 … … n
1 2 3 4 … n
1 2 3 4 5 … n
1 2 3 … … n
1 2 3 … … n
1 2 3 … … n
1 2 3 … … n
1 2 3 … … n
1 2 3 4 5 6 …

动态流水线时空图

流水线的分类

- 按照流水线的各个功能段之间是否有反馈信号，可以把流水线分为线性流水线和非线性流水线
  - 线性(Linear Pipelining)：每个流水段都流过一次，且仅流过一次，通常只完成一种固定的功能
  - 非线性(Nonlinear Pipelining)：在流水线的某些流水段之间有反馈回路或前馈回路



非线性流水线

流水线的分类

- 按照不同的数据表示方式，分为标量流水线和向量流水线
- 在线性流水线中，根据对流水线控制方式的不同，分为同步流水线和异步流水线
- 按照流水线输出端流出的任务和流水线输入端的任务流入顺序是否相同，分为顺序流水线和乱序流水线

流水线处理机的主要性能

- 吞吐率T_p（Throughput Rate）
- 加速比S_p（Speed Ratio）
- 效率η（Efficiency）

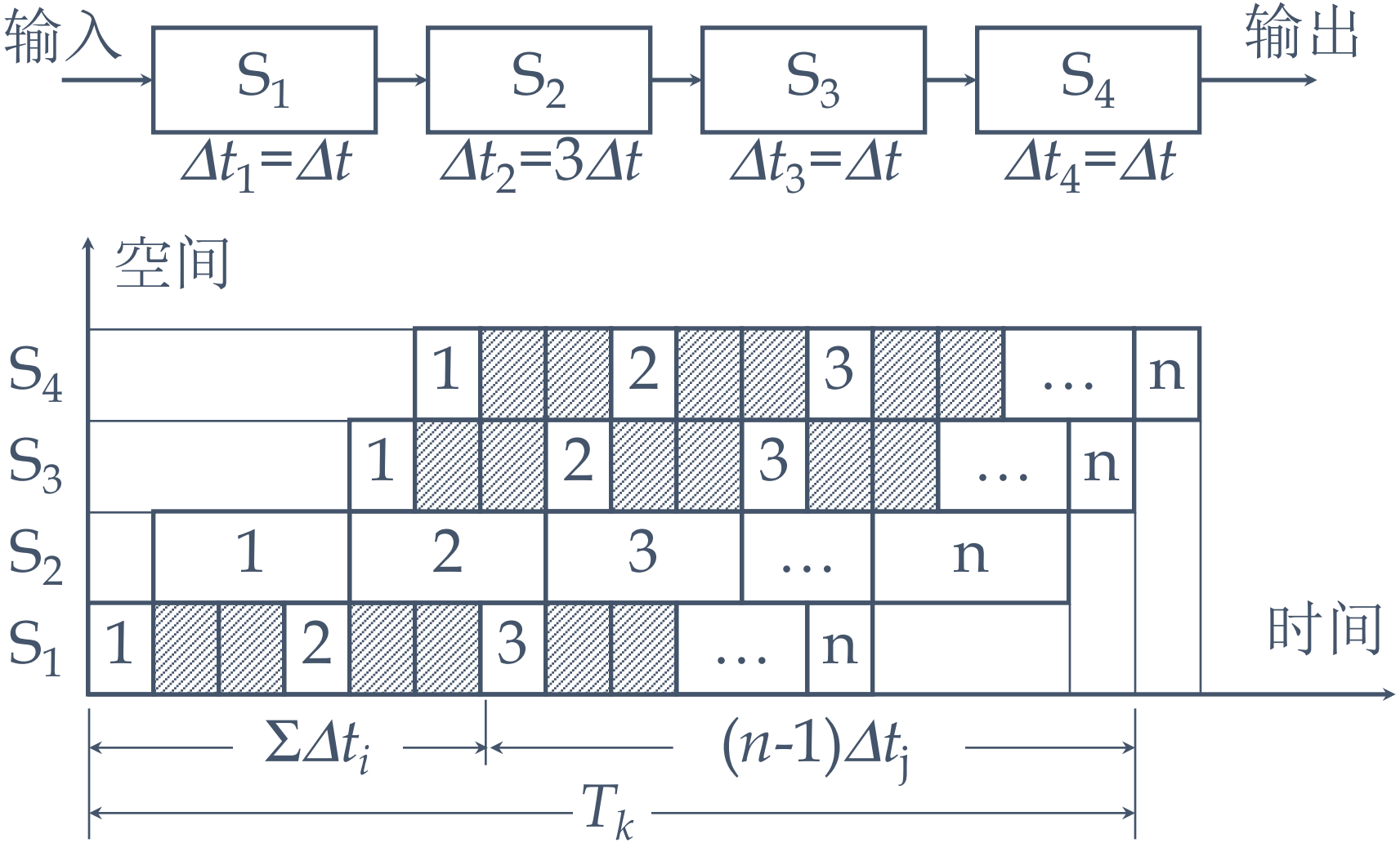
吞吐率（T_p，Throughput Rate）

- 单位时间内流水线所完成的任务数量或输出的结果数量
- 若各个子过程经过的时间都是Δt_2，则最大吞吐率为

$$T_{p_{max}} = 1/\Delta t_2$$

- 若各个子过程进行的工作不同，所经过的时间也不一定相同，所以在子过程之间设置接口锁存器，让各个锁存器都受同一时钟的同步

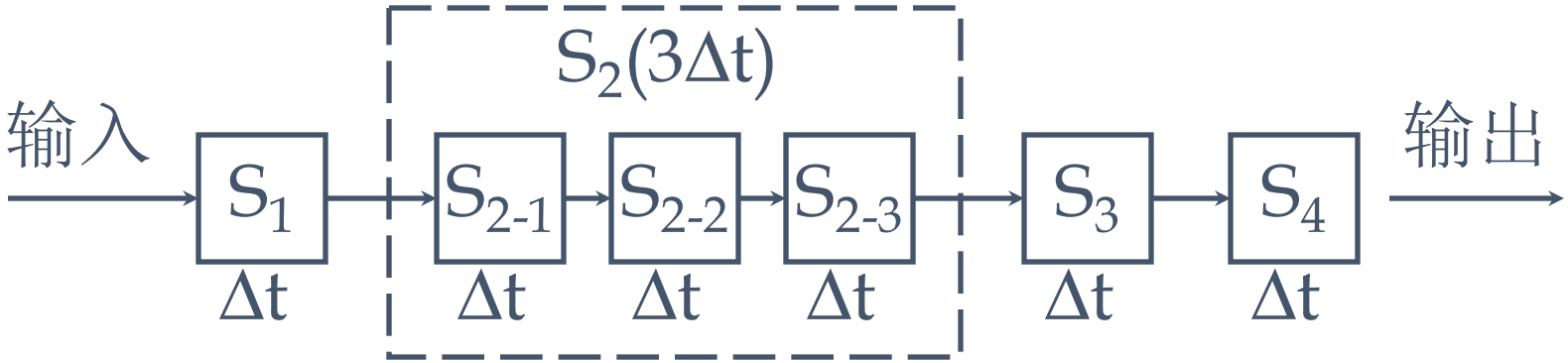
$$T_{p_{max}} = \frac{1}{\max\{\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4\}}$$



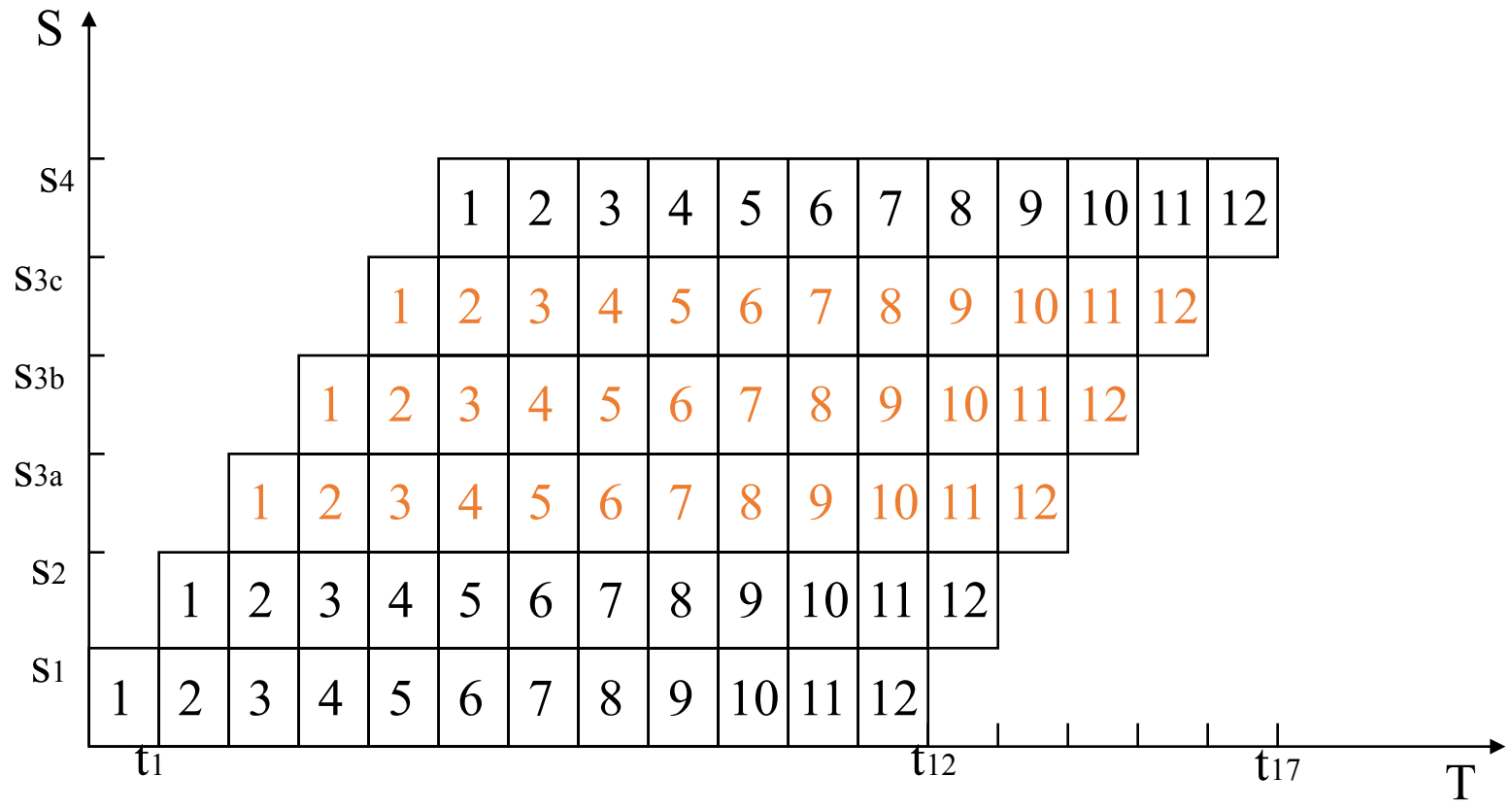
$$T_p = \frac{n}{\sum_{i-1}^{m} \Delta t_i + (n-1)\Delta \mathrm{t_j}}$$

吞吐率（续）

- 解决瓶颈子过程的办法
  - 细分

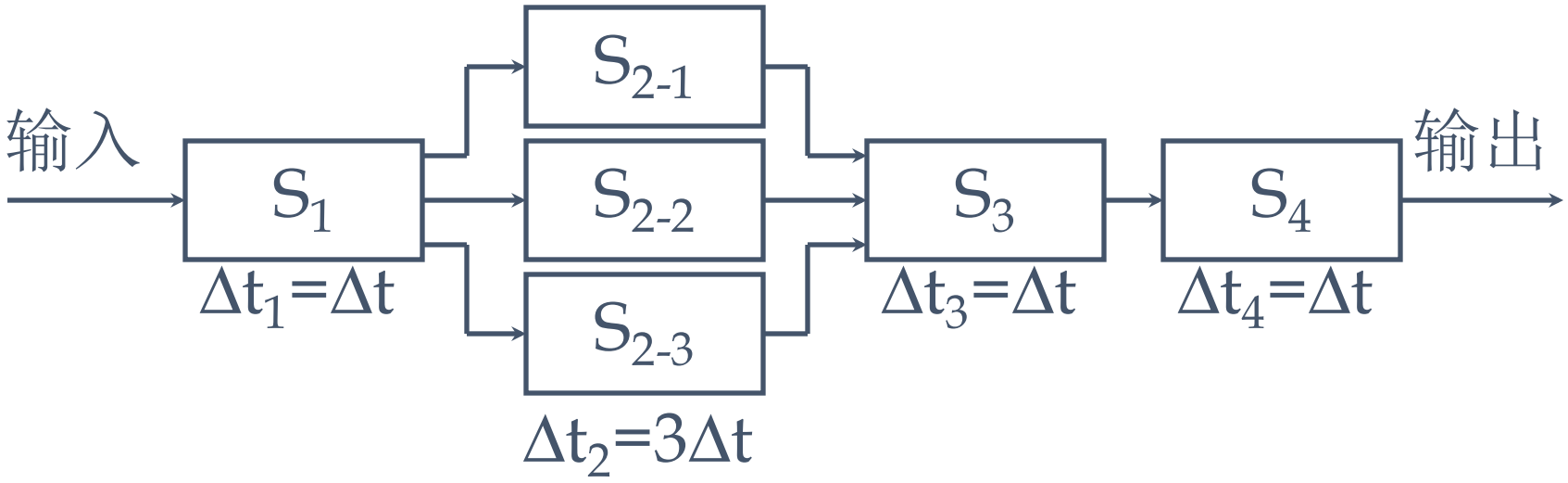


细分

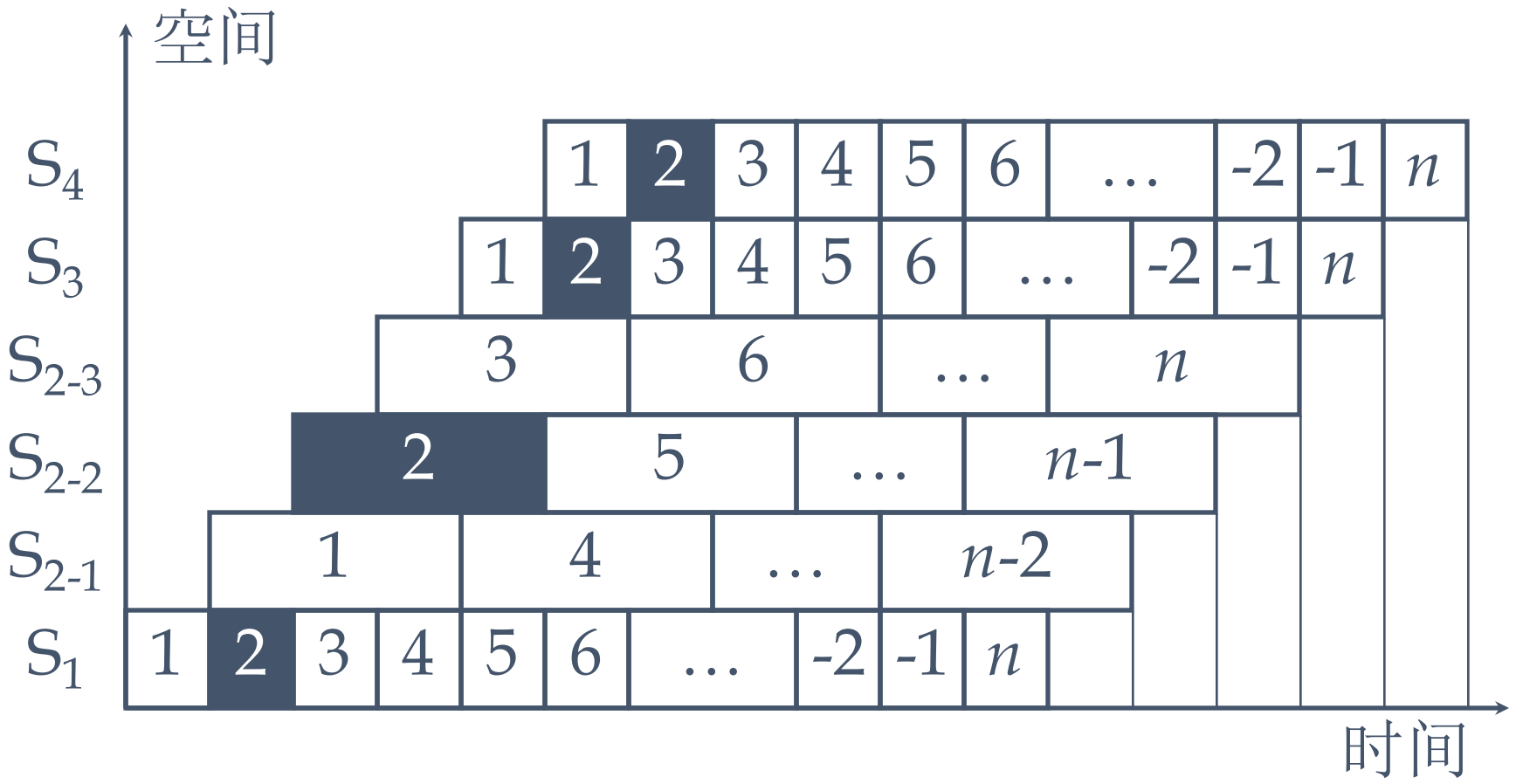


吞吐率（续）

- 瓶颈段并联


输入
S_1
$\Delta t_1=\Delta t$
$S_{2\text{-}1}$
$S_{2\text{-}2}$
$S_{2\text{-}3}$
$\Delta t_2=3\Delta t$
S_3
$\Delta t_3=\Delta t$
S_4
$\Delta t_4=\Delta t$
输出

并联


空间
S_4
S_3
$S_{2\text{-}3}$
$S_{2\text{-}2}$
$S_{2\text{-}1}$
S_1
1 2 3 4 5 6 … -2 -1 n
1 2 3 4 5 6 … -2 -1 n
3 6 … n
2 5 … n-1
1 4 … n-2
1 2 3 4 5 6 … -2 -1 n
时间

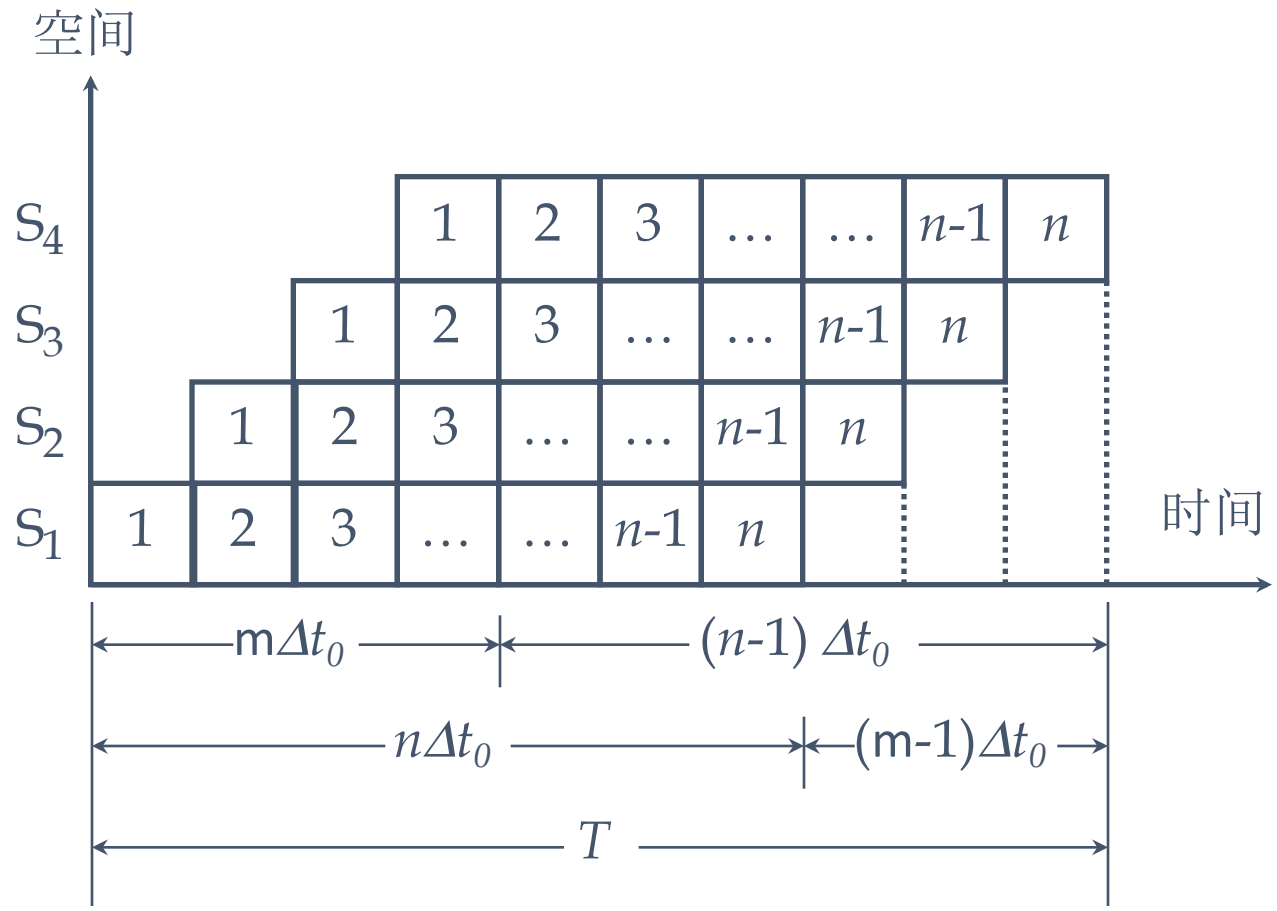
- m段流水线的各段经过时间均为Δt_0
- 完成n个任务的解释需要的时间为

$$T = m\Delta t_0 + (n-1)\Delta t_0$$

- 流水线的实际吞吐量为

$$T_p = \frac{n}{T} = \frac{1}{\Delta t_0\left(1 + \frac{m-1}{n}\right)} = \frac{T_{p_{max}}}{1 + \frac{m-1}{n}}$$

- 只有当$n \gg m$时，才能使实际吞吐率接近于最大吞吐率



加速比（Speedup Ratio）

- 指流水线的速度与等效的非流水线的速度之比。

$$T_{非流水} = n * m * \Delta t_0$$

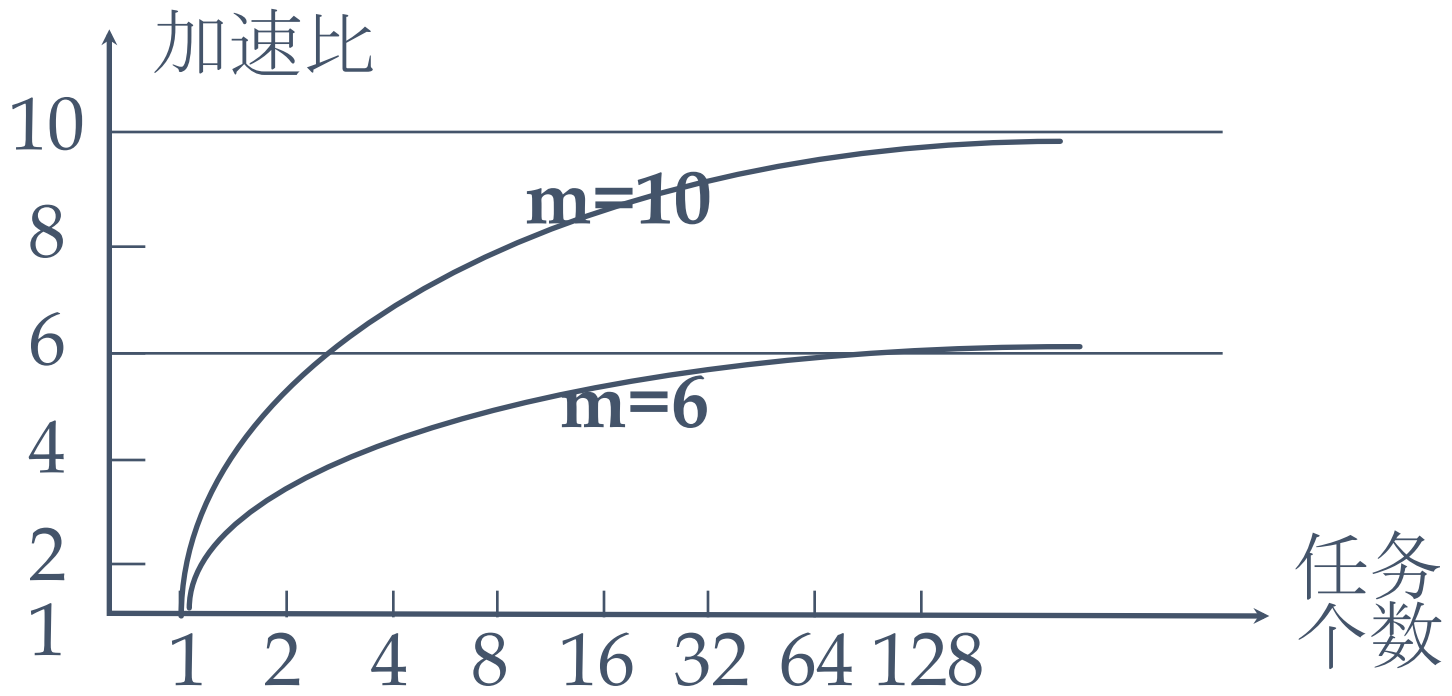
$$S_p = \frac{T_{非流水}}{T} = \frac{n * m * \Delta t_0}{m\Delta t_0 + (n-1)\Delta t_0} = \frac{m}{1 + \frac{m-1}{n}}$$

- 当$n \gg m$时，即连续流入的任务数n远多于流水线子过程数m时，其加速比最大＝m
- 只通过细分子过程，增大m来缩短Δt_0，则实际吞吐率将大大低于最大吞吐率
- 如果线性流水线每段经过的时间Δt_i不等，其瓶颈段时间为Δt_j，则

$$T_p = \frac{n}{\sum_{i=1}^{m} \Delta t_i + (n-1)\Delta \mathrm{t}_j}$$

$$S_p = \frac{n \sum_{i=1}^{m} \Delta t_i}{\sum_{i=1}^{m} \Delta t_i + (n-1)\Delta \mathrm{t}_j}$$

加速比（续）



效率（Efficiency）

- 是指流水线中的设备实际使用时间占整个运行时间之比，也称流水线设备的时间利用率
- 如为线性流水线、任务不相关且各段经过的时间相同Δt_0，则在T时间内，流水线各段效率相同

$$\eta_1 = \eta_2 = \cdots = \eta_m = \frac{n\Delta t_0}{T} = \frac{n}{m + (n-1)} = \eta_0$$

- 整条流水线的效率

$$\eta = \frac{\sum_{i=1}^{m} \eta_i}{m} = \frac{mn\Delta t_0}{mT}$$

效率（续）

- 分母$m*T$是时空图中m段与T所围成的总面积，而分子是n个任务实际占用的面积，效率是两者面积之比

$$\eta = \frac{n * \Delta t_0}{T} = \frac{n}{n + (m-1)} = T_p * \Delta t_0$$

- 流水线的效率正比与吞吐率

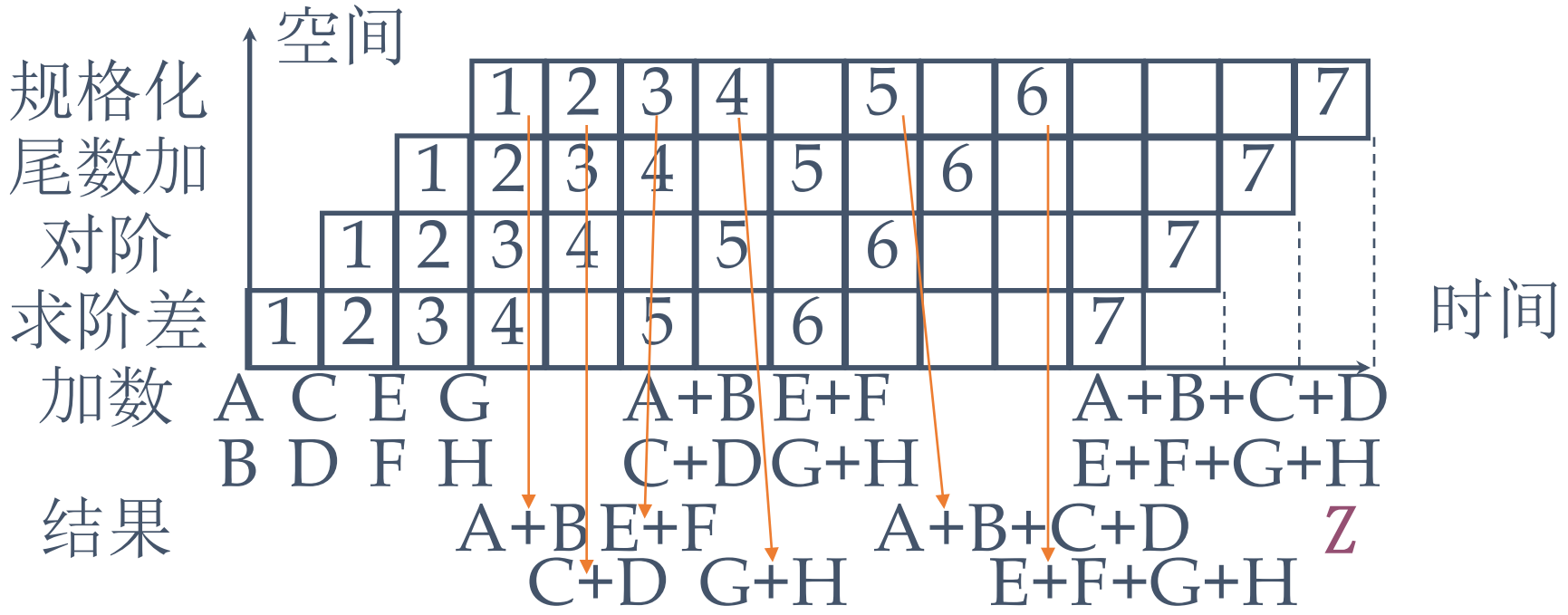
$$\eta = \frac{n}{n + (m-1)} = \frac{1}{1 + \frac{m-1}{n}} = \frac{S_p}{m}$$

- 当为非线性流水或为线性流水但各段经过时间不等或任务间有相关性时，正比关系不成立

$$\eta = \frac{n\text{个任务实际占用的时空区}}{m\text{个段的总时空区}} = \frac{n\sum_{i-1}^{m}\Delta t_i}{m[\sum_{i-1}^{m}\Delta t_i + (n-1)\Delta t_j]}$$

举例1：

- 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
- 解：Z = [(A+B) + (C+D)] + [(E+F) + (G+H)]



- 7个浮点加法共用了15个时钟周期。
- 流水线的吞吐率为：

$$T_p = \frac{n}{T_k} = \frac{7}{15 * \Delta t} = \frac{0.47}{\Delta t}$$

- 流水线的加速比为：

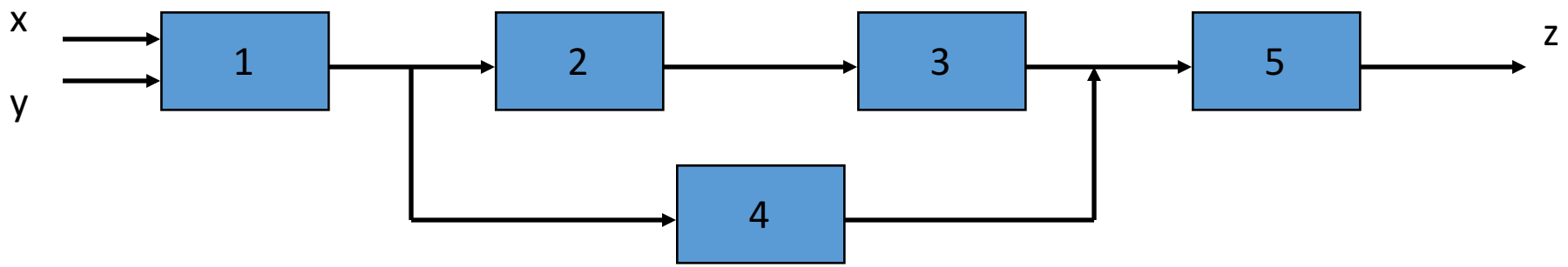
$$S = \frac{T_0}{T_k} = \frac{4 \times 7 \times \Delta t}{15 * \Delta t} = 1.87$$

- 流水线的效率为

$$E = \frac{T_0}{kT_k} = \frac{4 \times 7 \times \Delta t}{4 \times 15 \times \Delta t} = 0.47$$

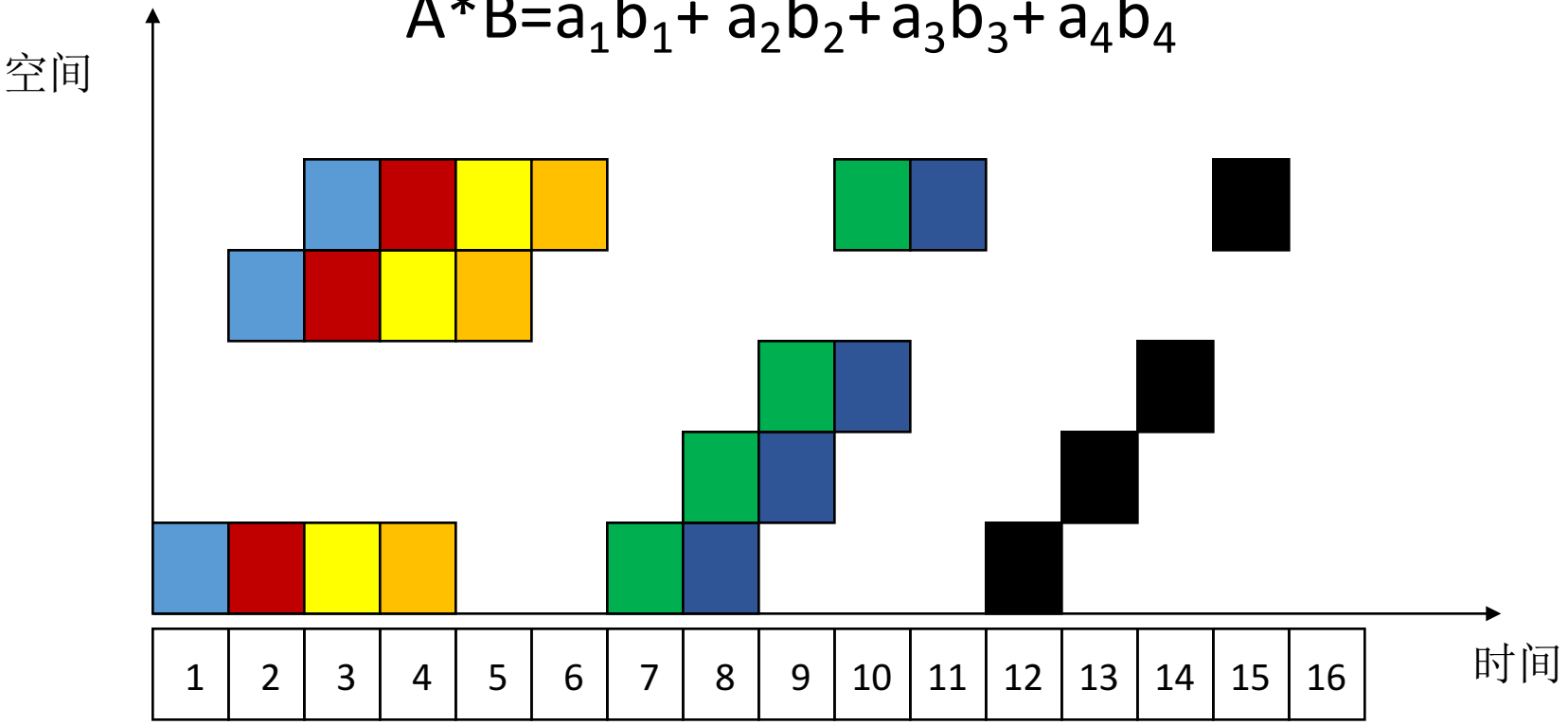
举例2:（静态多功能流水线）

- 设有两个向量A和B，各有4个元素，要在如图所示的静态双功能流水线上，计算向量点积AB，其中1-〉2-〉3-〉5组成加法流水线，1-〉4-〉5组成乘法流水线，又设每个流水线所经过的时间均为Δt，而且流水线的输出结果可以直接返回到输入或存于相应的缓冲寄存器中，其延迟时间和功能切换所需的时间都可以忽略不计。



静态多功能流水线

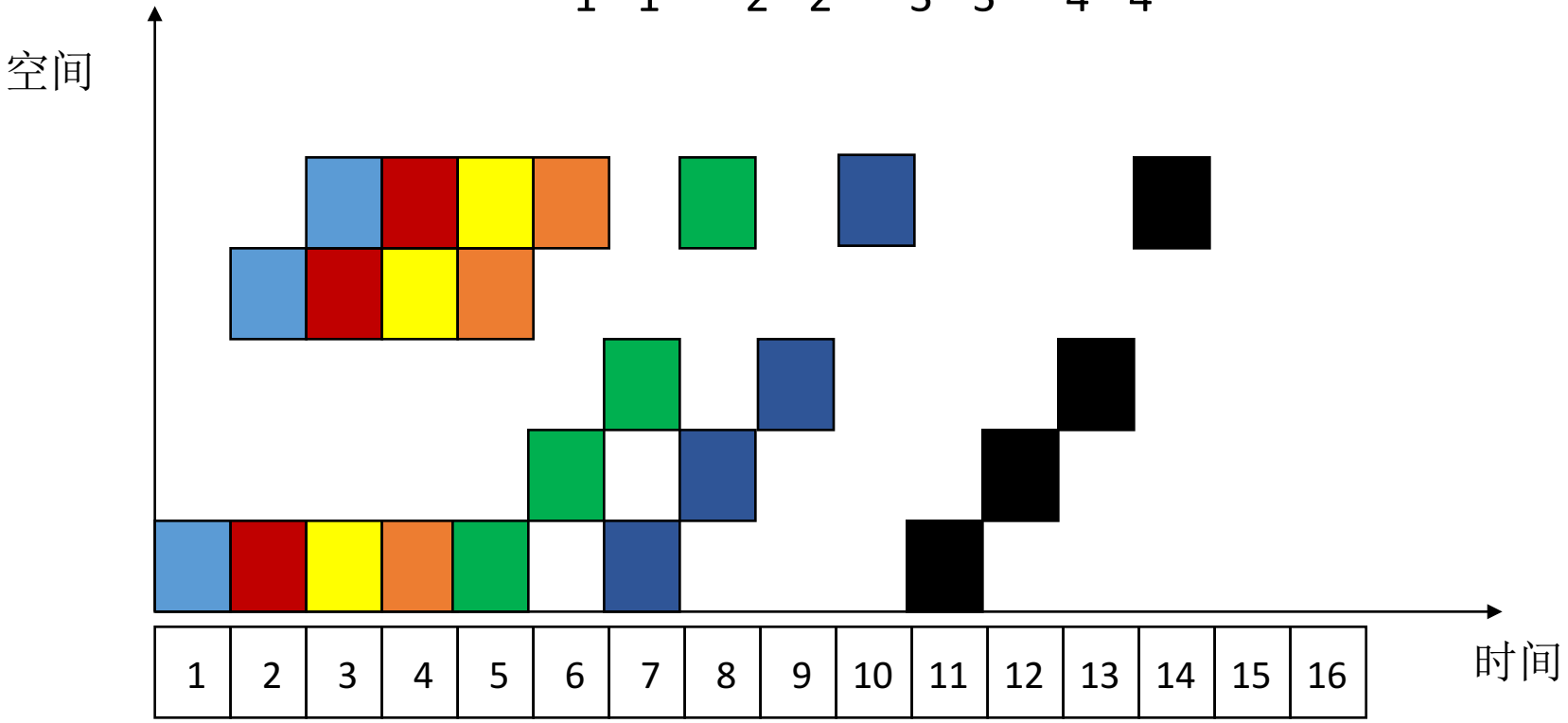
$A*B=a_1b_1+a_2b_2+a_3b_3+a_4b_4$



- 实际吞吐率$T_p = \frac{7}{15\Delta t}$
- 加速比$S_p = 24\Delta t/(15\Delta t) = 1.6$
- 效率$\eta =$ （$3 * 4\Delta t + 4 * 3\Delta t$）/（$5 * 15\Delta t$) $= 0.32 = 32\%$

动态多功能流水线

$$A*B=a_1b_1+a_2b_2+a_3b_3+a_4b_4$$



- 实际吞吐率$T_p = \frac{7}{14\Delta t}$
- 加速比$S_p = 24\Delta t/(14\Delta t) = 1.714$
- 效率$\eta =$ （$3 * 4\Delta t + 4 * 3\Delta t$）/（$5 * 14\Delta t$) $= 0.343 = 34.3\%$

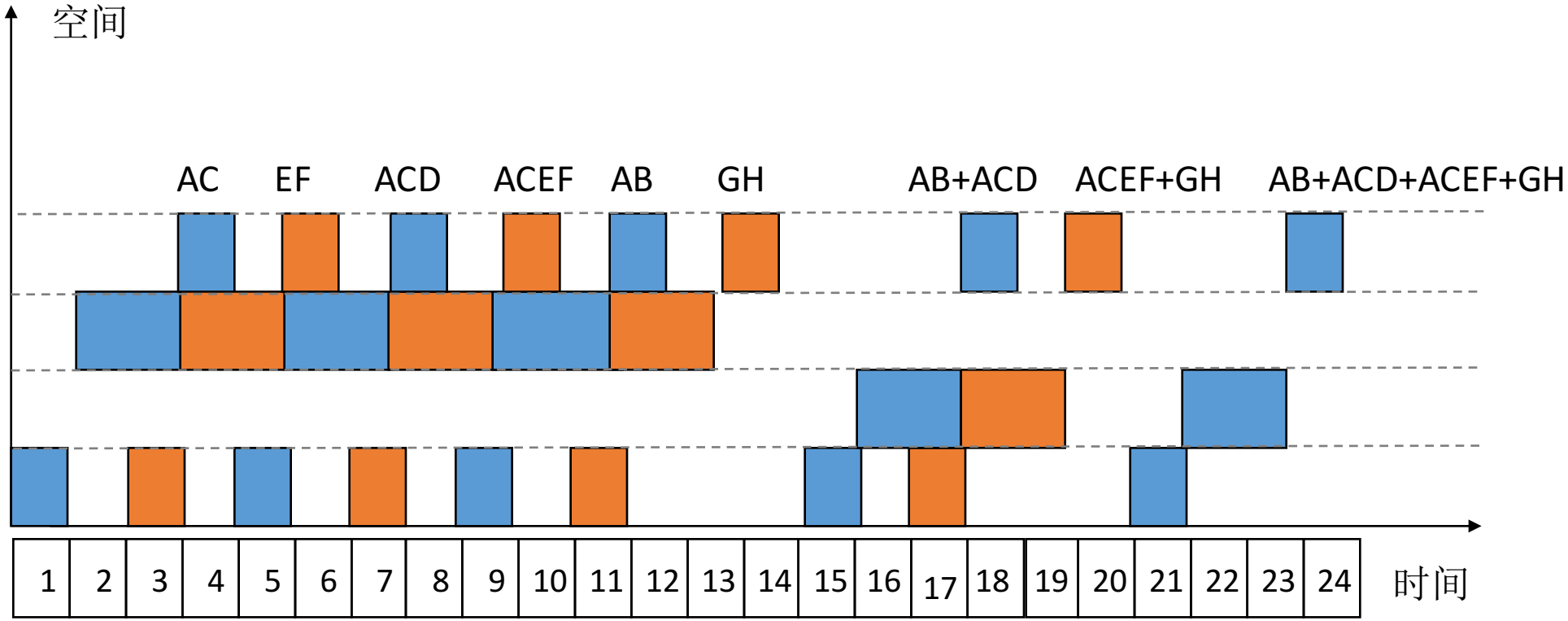
举例3：

- 有一个双输入端的加-乘双功能静态流水线，由经过时间分别为Δt、2Δt、2Δt、Δt的1、2、3、4四个子过程构成，加法时按1-〉2-〉4连接，乘法时按1-〉3-〉4连接。流水线输出设有缓冲器，也可将数据直接回授到流水线输入端。现要执行

  A*(B+C*(D+E*F))+G*H

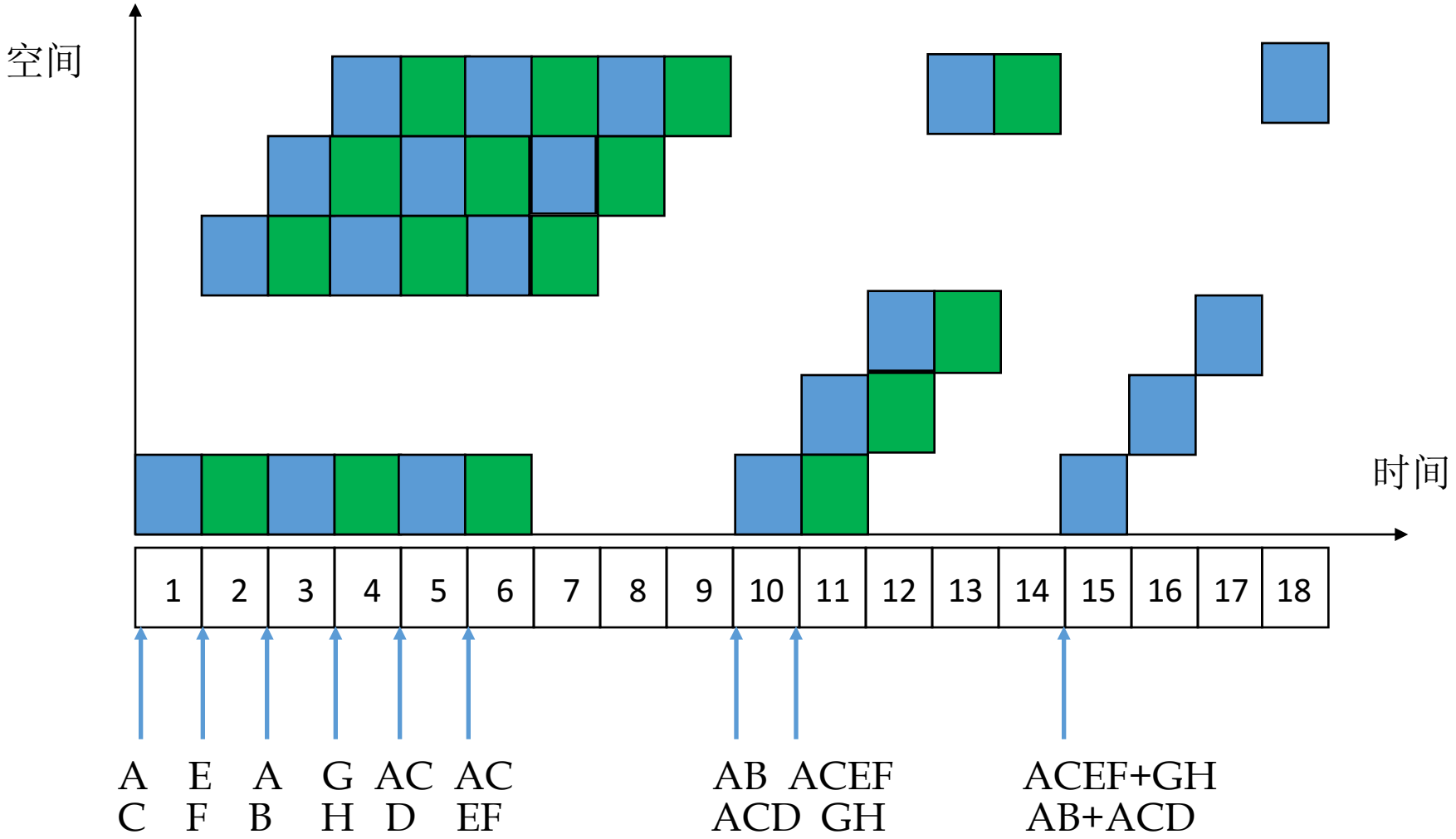
  的运算，请对运算顺序进行交换，画出能获得尽可能高的吞吐率的流水时空图；标出流水线入、出端的操作数变化情况；求出完成全部运算所需时间及此期间整个流水线的效率。如对流水线瓶颈子过程细分，最少需多少时间完成全部运算？若子过程3已无法再细分，只能采用并联方法改进，问流水线的效率为多少？

A*(B+C*(D+E*F))+G*H = A*B + A*C*D + A*C*E*F + G*H

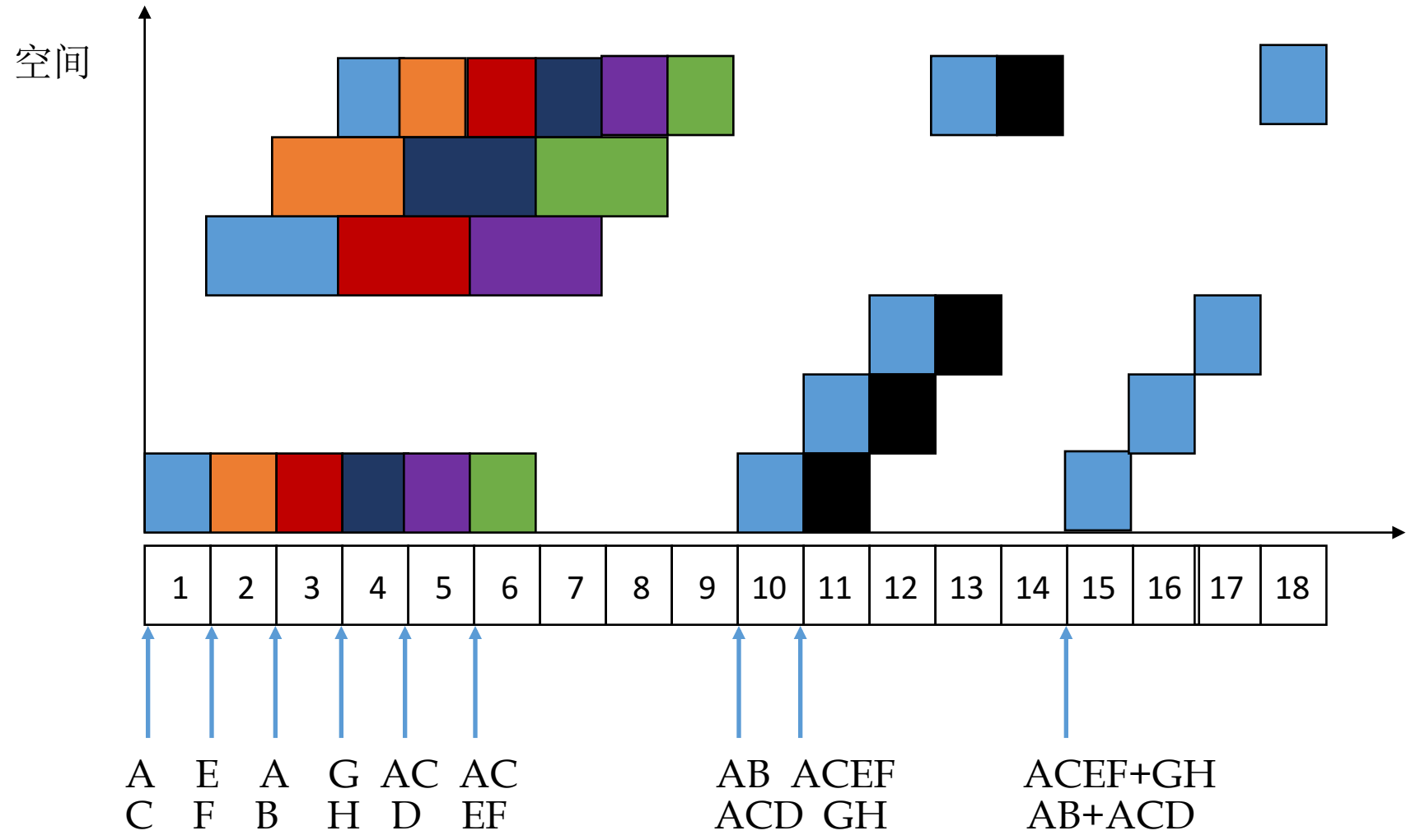


$$\eta = \frac{6\times 4\Delta t + 3\times 4\Delta t}{24\Delta t\times 4} = \frac{3}{8}$$

A*(B+C*(D+E*F))+G*H = A*B + A*C*D + A*C*E*F + G*H



$$\eta = (6*4\Delta t + 3*4\Delta t) \ / \ (6*18\Delta t) = 1/3$$



效率η=（6*4Δt+3*4Δt）/（6*18Δt) =1/3

流水机器的相关处理和控制机构

- 流水线只有连续不断地流动，不出现断流，才能获得高效率
- 断流的原因
  - 编译形成的目标程序不能发挥流水结构的作用
  - 存储系统供不上为连续流动所需要的指令和操作数
  - 出现了相关和中断

- 全局性相关：转移指令和其后的指令之间存在关联
  - 转移条件码由转移条件本身或其前一条指令形成，则只有该指令流出流水线后才能建立转移条件，期间不能继续流入任何指令
- 局部性相关：指令相关、主存操作数相关、通用寄存器组相关、基址值或变址值相关
  - 只影响相关的两条或几条指令，最多会使流水线某些段工作推后，不会改动指缓中预取到的指令，影响是局部的

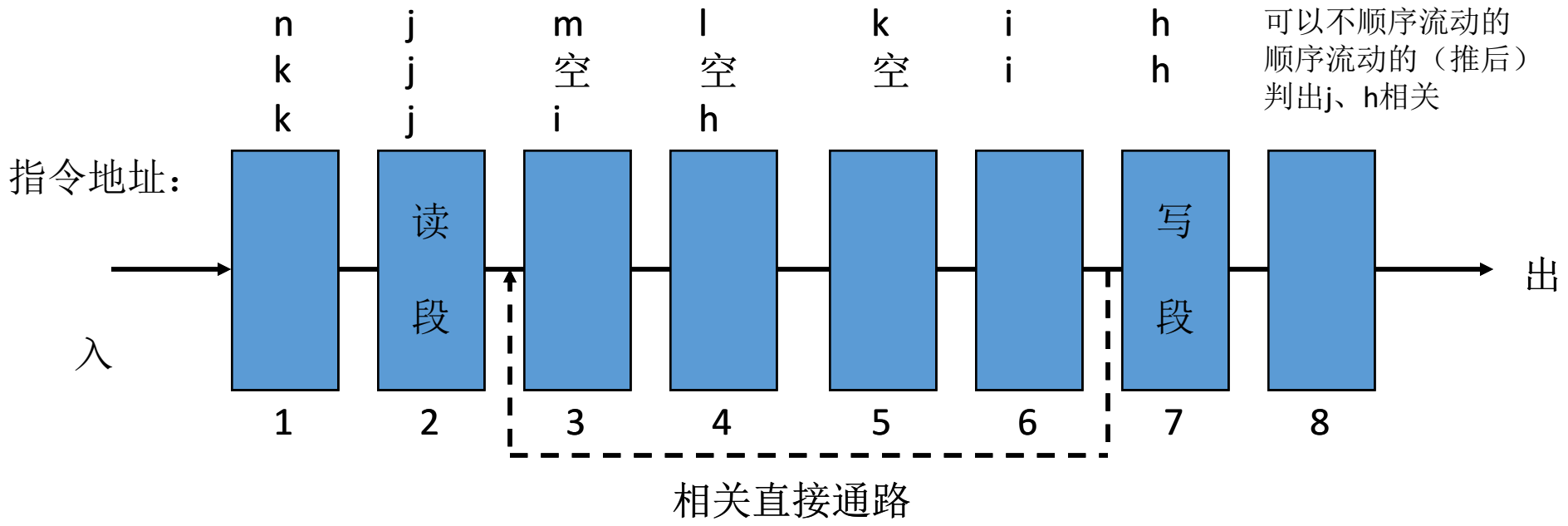
局部性相关的处理

- 原因：在机器同时解释多条指令之间出现了对同一主存单元或寄存器要求“先写后读”而产生的。
- 解决：
  - 推后后续指令对相关单元的读，直至在先的指令写入完成
  - 设置相关直接通路，将运算结果经相关直接通路直接送入所需部件

局部性相关的处理（续）

- 任务在流水线中流动顺序的安排和控制
  - 顺序流动方式（同步流动方式）：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：流出流水线的任务（指令）可以和流入流水线的顺序不同

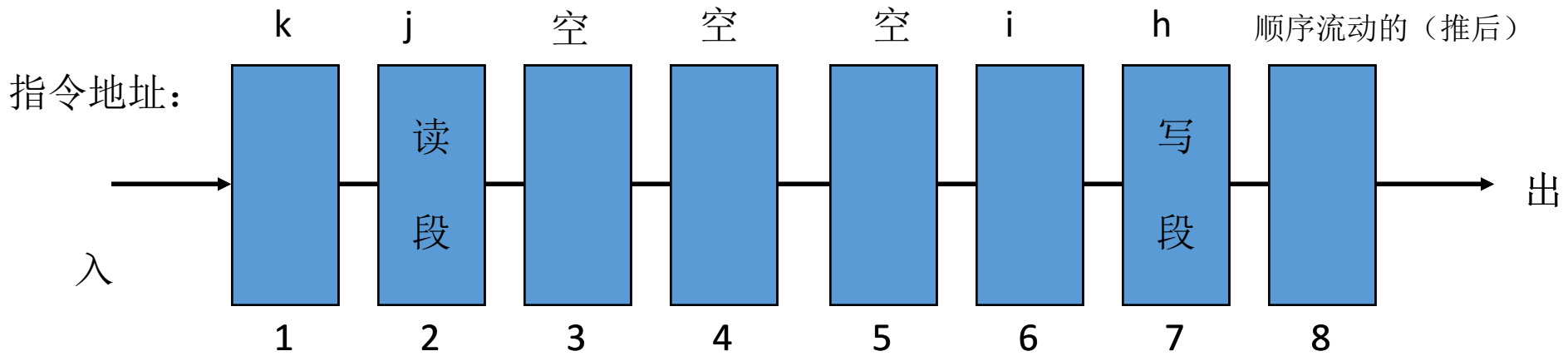
指令j的源操作数地址与指令h的目的操作数地址相同时，h和j就发生**先写后读**的操作数相关。



顺序流动和异步流动

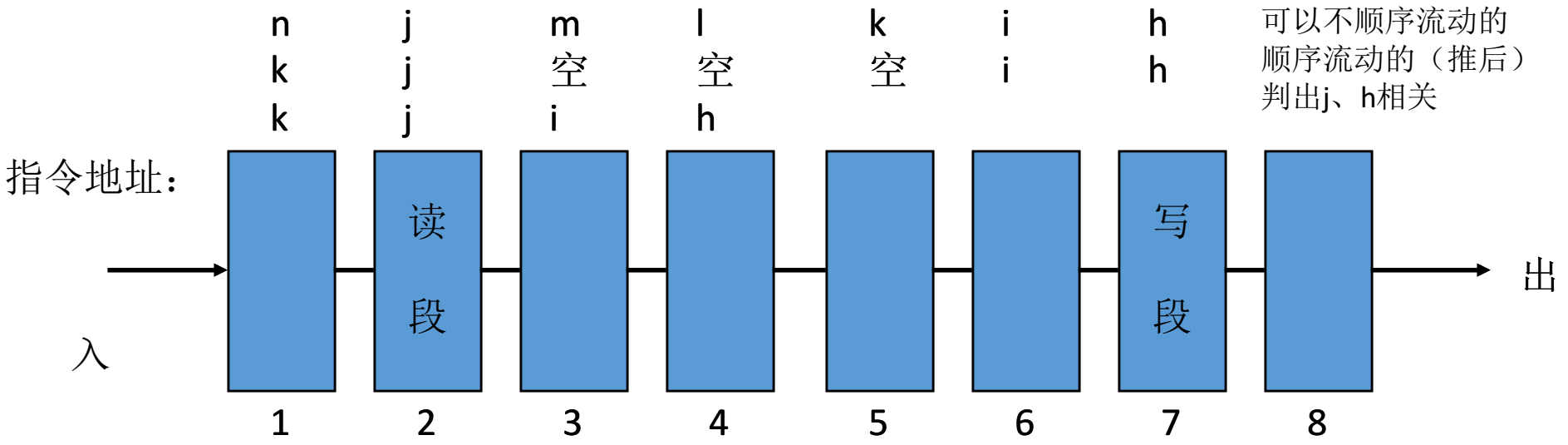
举例：流动顺序的控制

- 8段流水线，第2段为读段，第7段为写段
- 一串指令流入：h, i, j, k, l, m, n
- 当指令j的源操作数地址与指令h的目的操作数相同时，发生先写后读的操作数相关
- 顺序流动时：j读段是停下来等待，直到h到达写段并完成后，才流动。推后读。
  - 优点：控制比较简单
  - 相关后流水线的吞吐率和效率下降



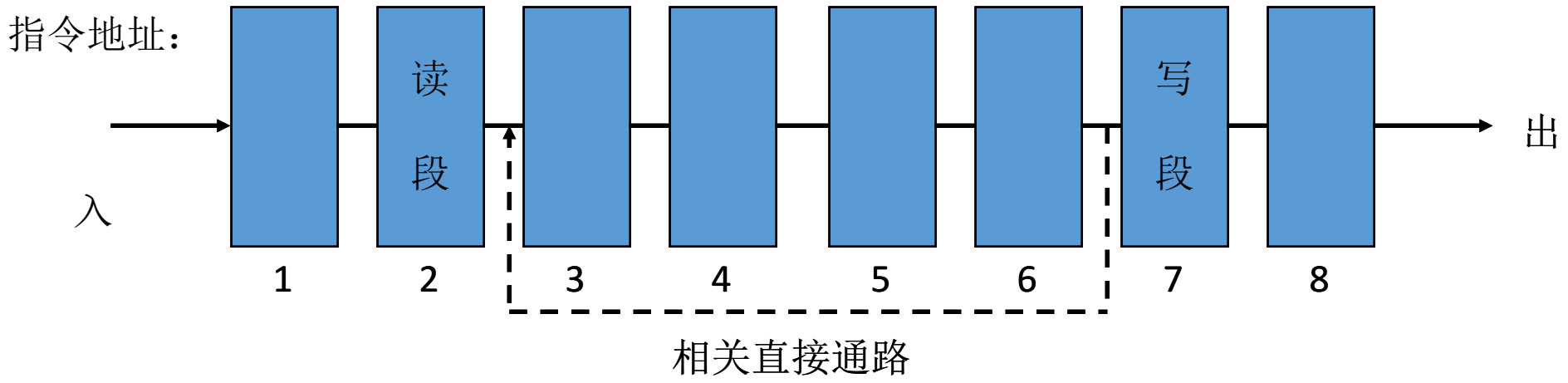
举例：流动顺序的控制（续）

- 异步流动：如果让j之后的指令，如k, l, m, n，只要与j没有相关，就越过j继续向前流动。



- 当采用异步流动方式后，会发生其他相关
  - **写-写相关**：对同一单元，要求在先的指令先写入，在后的指令后写入的关联。例如指令i和k对同一单元进行写入操作，由于指令i有“先写后读”相关，出现指令k先于指令i到达“写段”，使得该单元的内容错为指令i的写入结果
  - **先读后写相关**：对同一单元，要求在先的指令先读出，在后的指令再写入的关联。若指令i的读操作和指令k的写操作对应于同一单元，若指令k越过指令i向前流，且其写操作在指令i的读操作开始前完成，那指令i就会错误地读出指令k的写入结果
- 异步流动时，控制机构应能同时处理好这三种相关：“先写后读”、“写—写”、“先读后写”

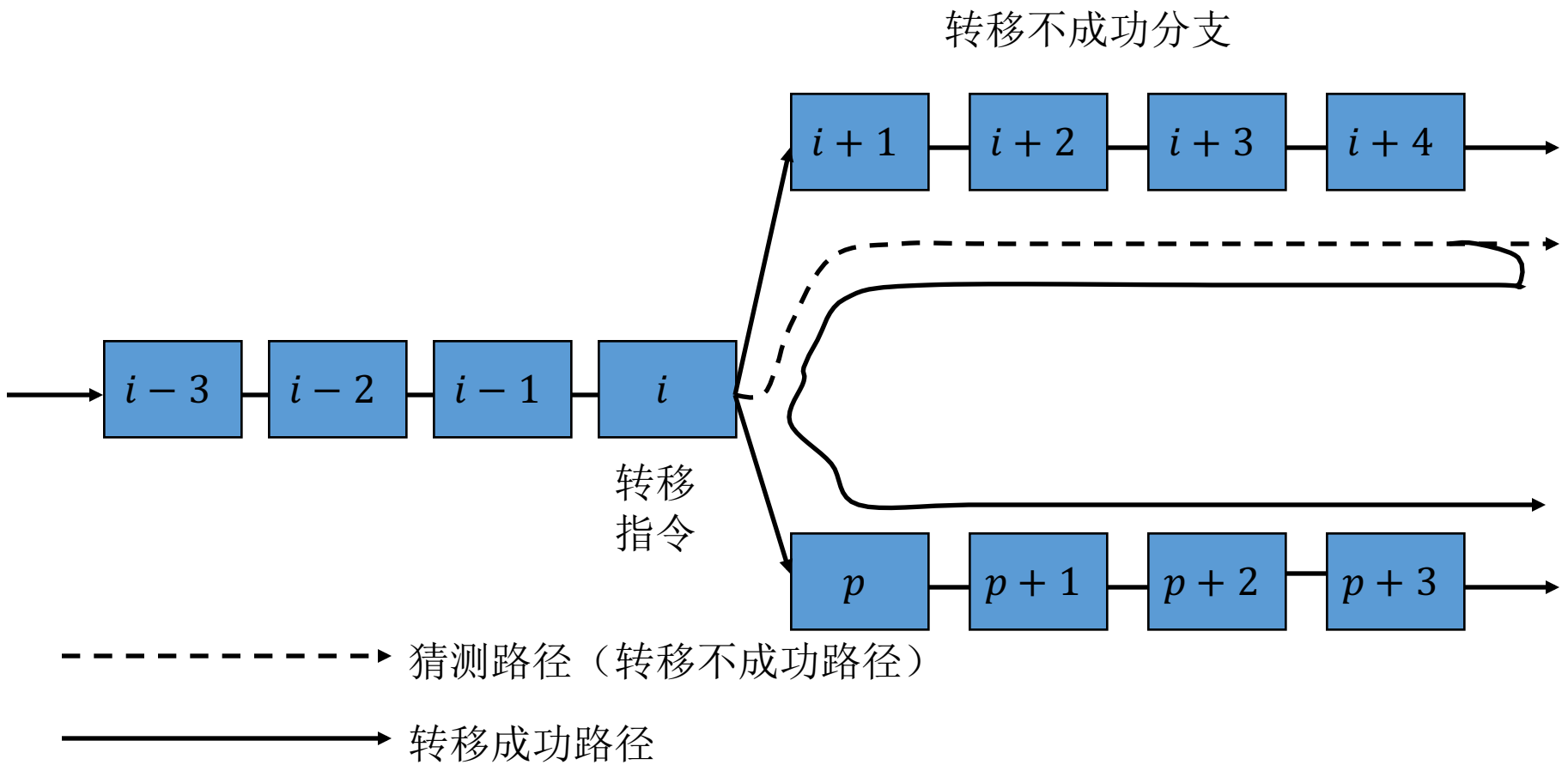
- 设置专用通路
  - 采用分布式控制和管理，并设置公共数据总线，以简化各种相关的判别和实现相关直接通路的连接



全局性相关的处理

- 指的是已进入流水线的转移指令（尤其是条件转移指令）和其后续指令之间的相关。
  - 猜测法
  - 加快和提前形成条件码
  - 采用延迟转移------采用软件进行静态指令调度
  - 加快短循环程序的处理

猜测法


转移不成功分支
$i+1$
$i+2$
$i+3$
$i+4$
$i-3$
$i-2$
$i-1$
i
转移
指令
p
$p+1$
$p+2$
$p+3$
猜测路径（转移不成功路径）
转移成功路径

猜测法（续）

- 在典型的标量类机器指令程序中，条件转移指令占20%，其中转移成功的概率有约占其中的60%
- 如果概率相近，则宜选择转移不成功分支
- 如果转移的两个分支概率不均等，宜猜高概率分支
- 转移概率可以静态地根据转移指令类型或程序执行期间的转移历史状况来预测，也可以由编译程序根据执行过程中转移的历史记录来动态预测未来的转移选择

猜测法（续）

- 猜测法应能保证猜错时可恢复分支点原先的现场
  - 方法1：对指令只译码和准备操作数，在转移条件码出现之前不进行运算
  - 方法2：运算完不送回结果
  - 方法3：把可能被破坏的原始状态存储在后援寄存器中，若猜错，则可以取出后援寄存器中的内容恢复分支点的现场
  - 方法4：设置转移目标指令缓冲器，以便尽快回到原分支处转入另一分支，减少流水线等待时间

加快和提前形成条件码

- 加快单条指令内部条件码的形成，不等指令执行完就提前形成反映运算结果的条件码
  - 例如，判断乘除运算结果正负可在实际运算前完成
- 在一段程序内提前形成条件码
  - 适合循环型程序在判断循环是否继续时的转移情况
  - 例如，为了使“不等于零”条件转移指令的条件码提前形成，可以将减1指令提前到与其不相关的其他指令之前，甚至提前到循环体外开始执行
  - 在硬件上增设为循环减1指令专用的条件码寄存器

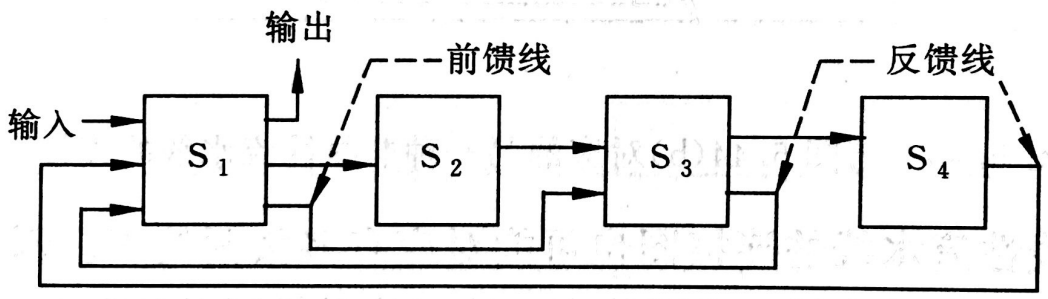
- 采取延迟转移
  - 采用软件方法进行静态指令调度
  - 在编译时，将转移指令与其前面不相关的一条或多条指令交换位置，让成功转移总是延迟到这一条或多条指令执行之后再进行
- 加快短循环程序的处理
  - 将长度小于指缓容量的短循环程序整个一次性放入指缓内，并暂停预取指令，避免循环执行过程中由于指令预取操作将需要循环执行的指令冲掉，减少主存重复读取次数
  - 由于循环分支概率高，让循环出口端的条件转移指令恒猜循环分支，减少因条件分支造成流水线断流的机会

流水机器的中断处理

- 中断会引起流水线的断流，但是其出现概率比条件转移的概率要低。
- 处理中断的主要问题：断点现场的保护和恢复，而不是缩短流水线的断流时间。
- 不精确断点：无论指令i在流水线的哪一段发生中断，都不再允许尚未进入流水线的后续指令再进入，但已在流水线的所有指令仍继续流动到执行完毕，然后才转入中断处理程序。
  - IBM 360/91
  - 不利于编程和程序的排错
- 精确断点：无论指令i是在流水线中的哪一段响应中断，中断现场全都是对应i的，i之后流入流水线内的指令的原有现场都能保存和恢复。
  - 需设置很多后援寄存器
  - 控制逻辑比较复杂

非线性流水线的表示

- 非线性流水线中，一个任务在流水线的各个功能段中不是线性流动的，有些功能段需要反复使用多次
- 一条非线性流水线需要一个各个功能段之间的连接图和一张预约表共同来表示



(a) 非线性流水线的连接图

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

(b) 非线性流水线的预约表

- 一张非线性流水线的预约表可能与多个非线性流水线连接图相对应
  - 在预约表的同一列中可能有多个“X”，即在这个时钟周期中有多个功能段有输出，从而造成下一个功能段的输入可能有多种来源，从而对应有多个流水线的连接图
- 一个非线性流水线的连接图也可能对应多张预约表
  - 在非线性流水线中，有些功能段可能有多个输出端，也有些功能段可能有多个输入端，这些输出端和输入端之间的连接的先后次序形成了多张预约表

流水线调度

- 向一条非线性流水线的输入端连续输入两个任务之间间隔称为非线性流水线的启动距离或等待时间。
- 非线性流水线中的冲突：当以某个启动距离向一条非线性流水线连续输入任务时，可能在某个功能段，或某几个功能段中发生有几个任务同时争用同一个功能段的情况

拍号n

段号k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X_1			X_2			X_3		X_1			X_2			X_3
2		X_1	X_1		X_2	X_2		X_1X_3	X_3		X_2			X_3	
3				X_1			X_2			X_3					
4					X_1	X_1		X_2	X_2		X_3	X_3			
5							X_1	X_1		X_2	X_2		X_3	X_3	

启动距离为3的流水线冲突情况

- 引起非线性流水线功能段冲突的启动距离称为禁止启动距离
- 有些启动距离在非线性流水线的所有功能段在任何时间都不会发生冲突
- 不发生冲突的启动距离不一定仅仅是一个常数，在一般情况下是一个循环序列

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

无冲突调度方法

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

拍号n

段号k	1	2	3	4	5	6	7	8	9
1	*								*
2		*	*					*	
3				*					
4					*	*			
5							*	*	

预约表

延迟禁止表F(Forbidden List)

- {1,5,6,8}，相邻两个任务的间隔拍数不能为1，5，6，8

冲突向量C(Collision Vector)

- 第i位的状态用以表示与当时相隔i拍给流水线送入后继任务是否会发生功能段的使用冲突。如不发生，0，否则，1
- 初始冲突向量 $C = (10110001)$

（1）形成预约表

指令总拍数为n，流水线有k个段，则形成$n\times k$的预约表，段的使用情况用“×”表示

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

（2）由预约表形成禁止表F

F={各段中冲突间隔拍数}

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

1　6　5　8

本例：　$F = \{1，5，6，8\}$

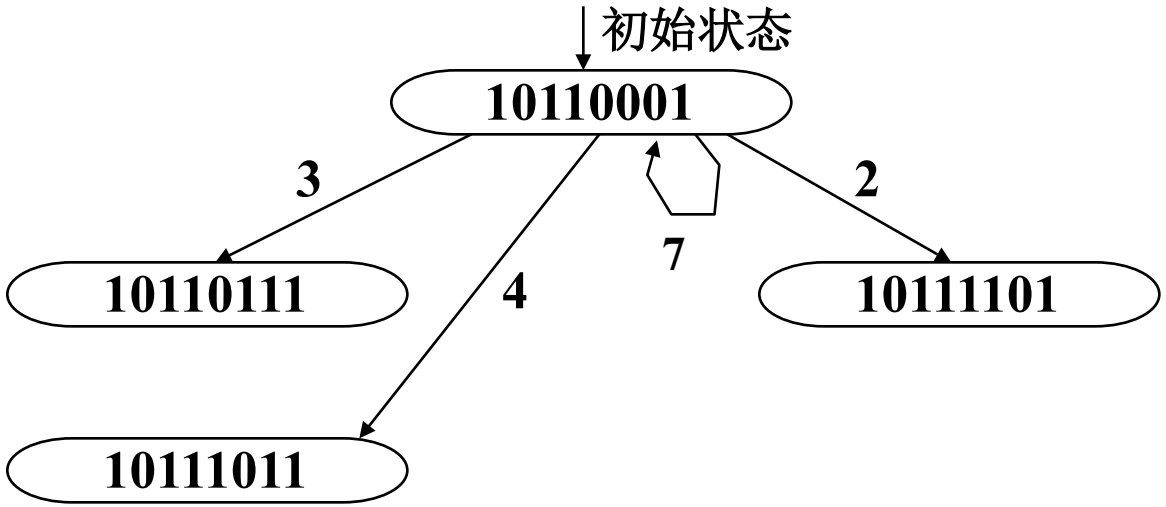
（3）由禁止表F形成初始冲突向量C_0

$C_0 =$ （$c_{n-1} \ldots c_1$），$c_i = 1$冲突，$c_i = 0$不冲突。

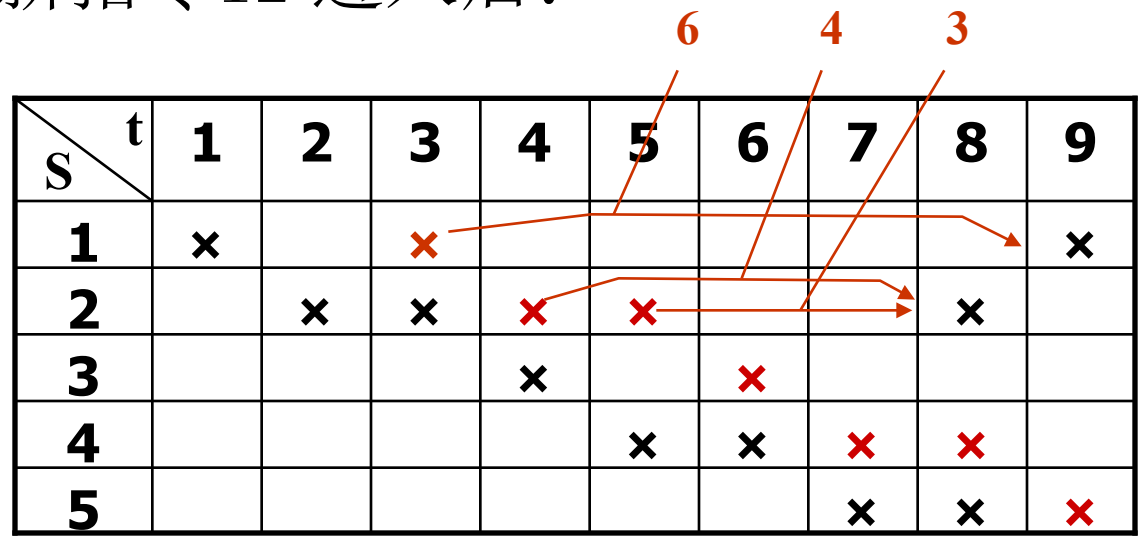
本例：$C_0 =$ （10110001）。

（4）由初始冲突向量C_0形成状态转换图

C_0每过一拍逻辑右移一位，若移出0，则允许后续指令进入流水线，再与C_0按位“或”，形成新的冲突向量C_i；



2拍后，新指令I2 进入后：

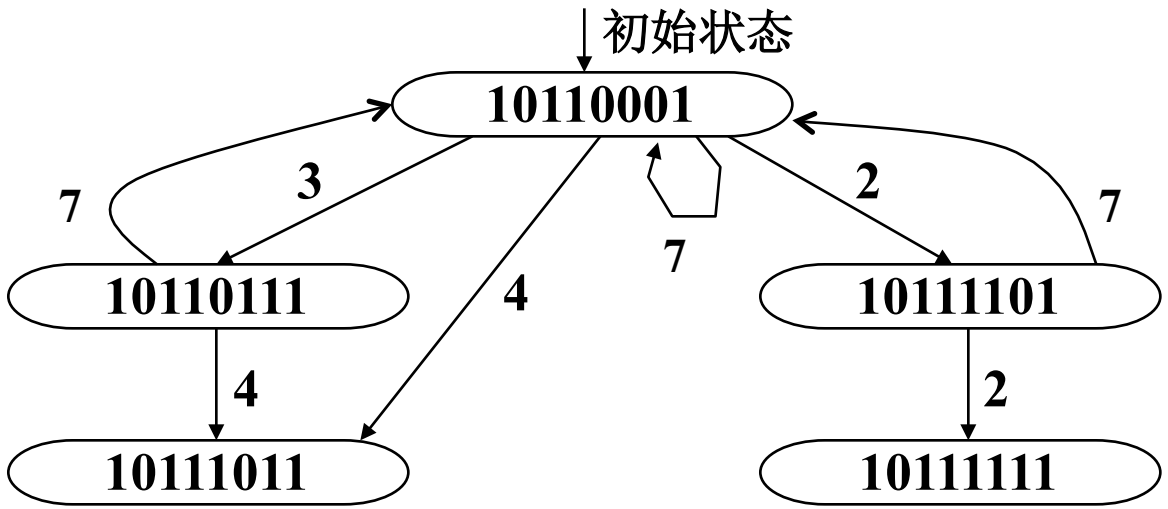


I1与I3的$F=\{3,4,6\}$，I2与I3的$F=\{1,5,6,8\}$,

新$F=\{1,3,4,5,6,8\}$，$C_2=(10111101)$。

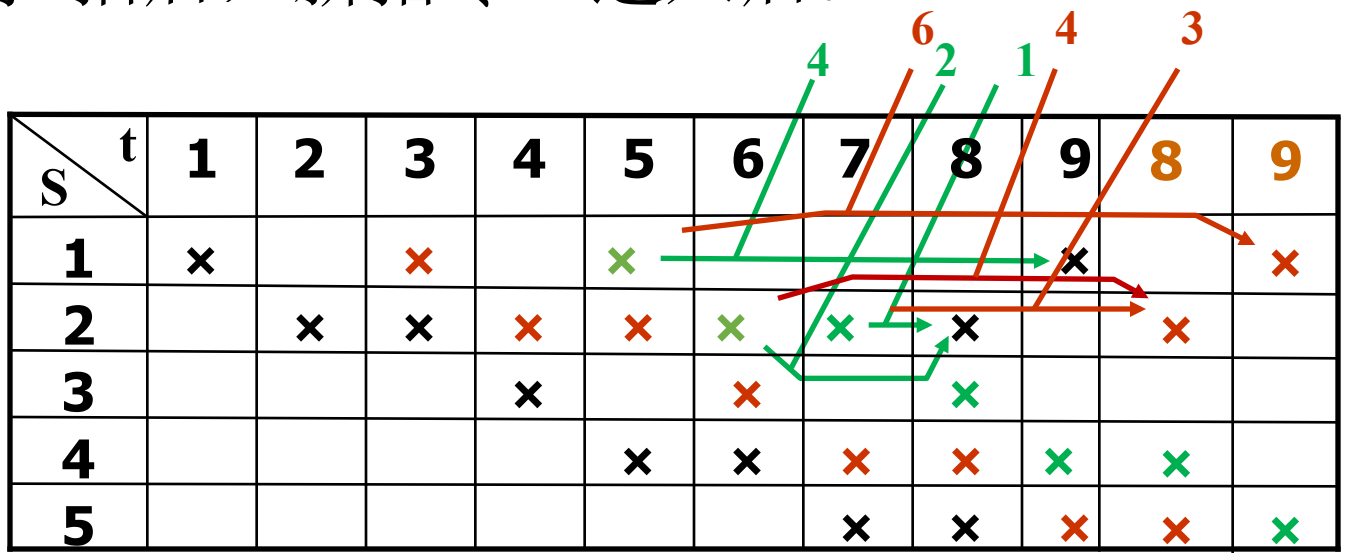
注意：C_i为第i拍后流水线的冲突向量，此时流水线中已有两条指令，C_i用于判断第三条指令的进入。

各C_i再每过一拍逻辑右移一位，若移出0，允许后续指令进入，再与C_0按位“或”，形成新的冲突向量C_{ij}；



注意：C_{ij}为第$i+j$拍后流水线的冲突向量，此时流水线中已有三条指令，C_{ij}用于判断第四条指令的进入。

对C_2，再2拍后，新指令I3进入后：

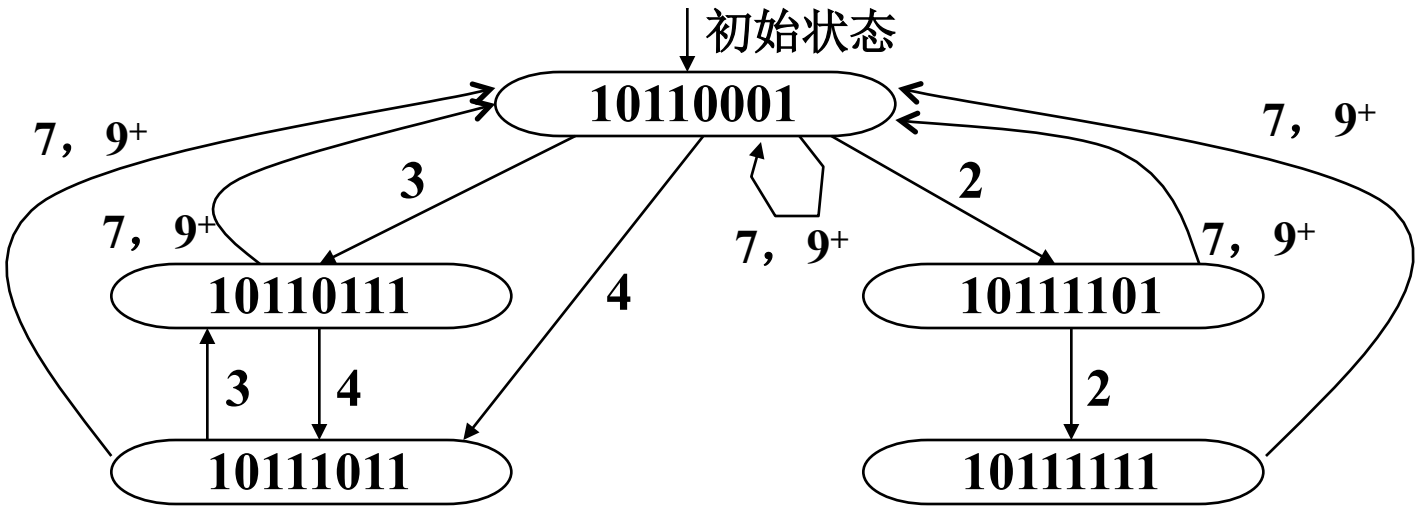


I1与I4的$F=\{1,2,4\}$，I2与I4的$F=\{3,4,6\}$，I3与I4的$F=\{1,5,6,8\}$，

新$F=\{1,2,3,4,5,6,8\}$，$C_{22}=(10111111)$。

注意：C_{ij}为第$i+j$拍后流水线的冲突向量，此时流水线中已有三条指令，C_{ij}用于判断第四条指令的进入。

重复上一步骤，直到不再生成新的冲突向量为止。



（5）找出最加调度方案

从各个闭合回路中找出平均间隔最小的一个。

（5）找出最加调度方案

- 在状态图中找出不发生功能段冲突的启动循环
- 实际上，不发生冲突的启动循环可能有无穷多个
- 由于非线性流水线调度的目标是找出平均启动距离最小的启动循环，因此只需要找出简单循环即可
  - 简单循环：状态图中各种冲突向量只经过一次的启动循环
  - 简单循环的个数是有限的

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		

按(3,4)进行调度

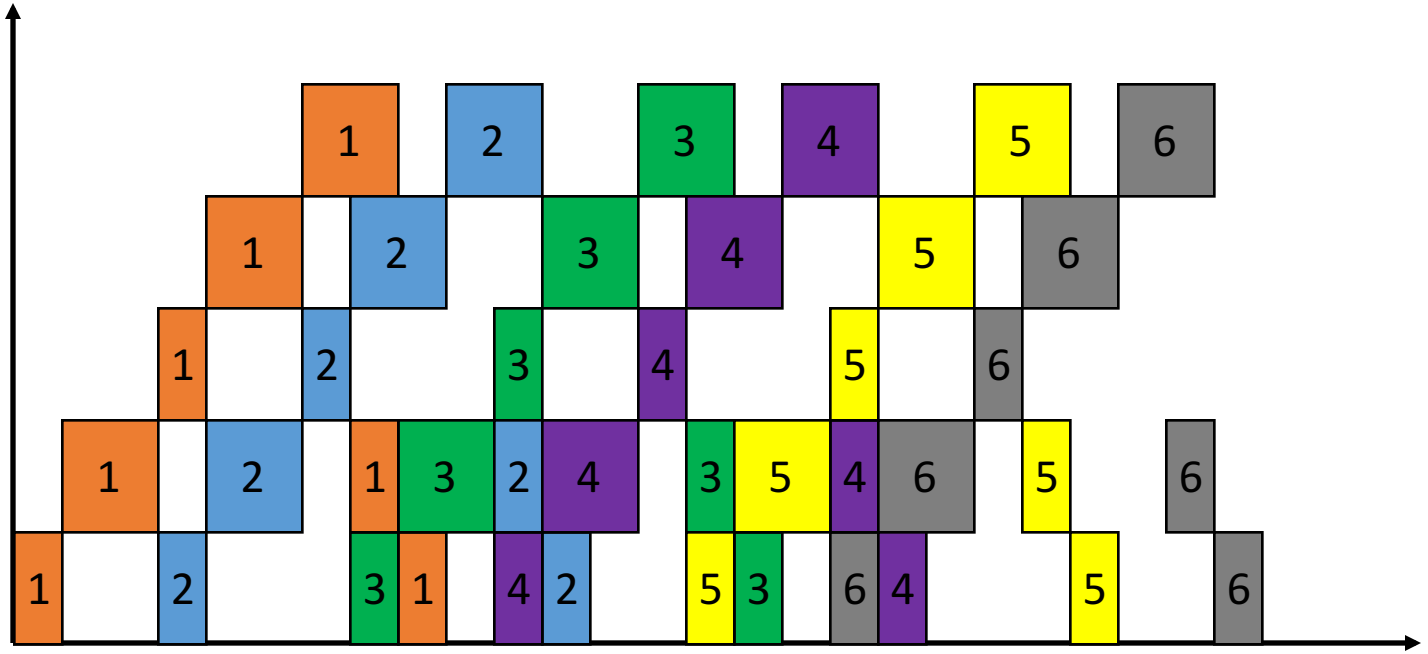
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X_1			X_2				X_3	X_1			X_2			
2		X_1	X_1		X_2	X_2		X_1	X_3	X_3	X_2			X_3	
3				X_1			X_2				X_3				
4					X_1	X_1		X_2	X_2			X_3	X_3		
5							X_1	X_1		X_2	X_2			X_3	X_3

按(3,4)进行调度

$$T_P = 6/26$$

$$S_p = (6 * 10)/26 = 30/13$$

$$e = (6 * 10)/(26 * 5) = 60/130 = 6/13$$



按(4, 3)进行调度

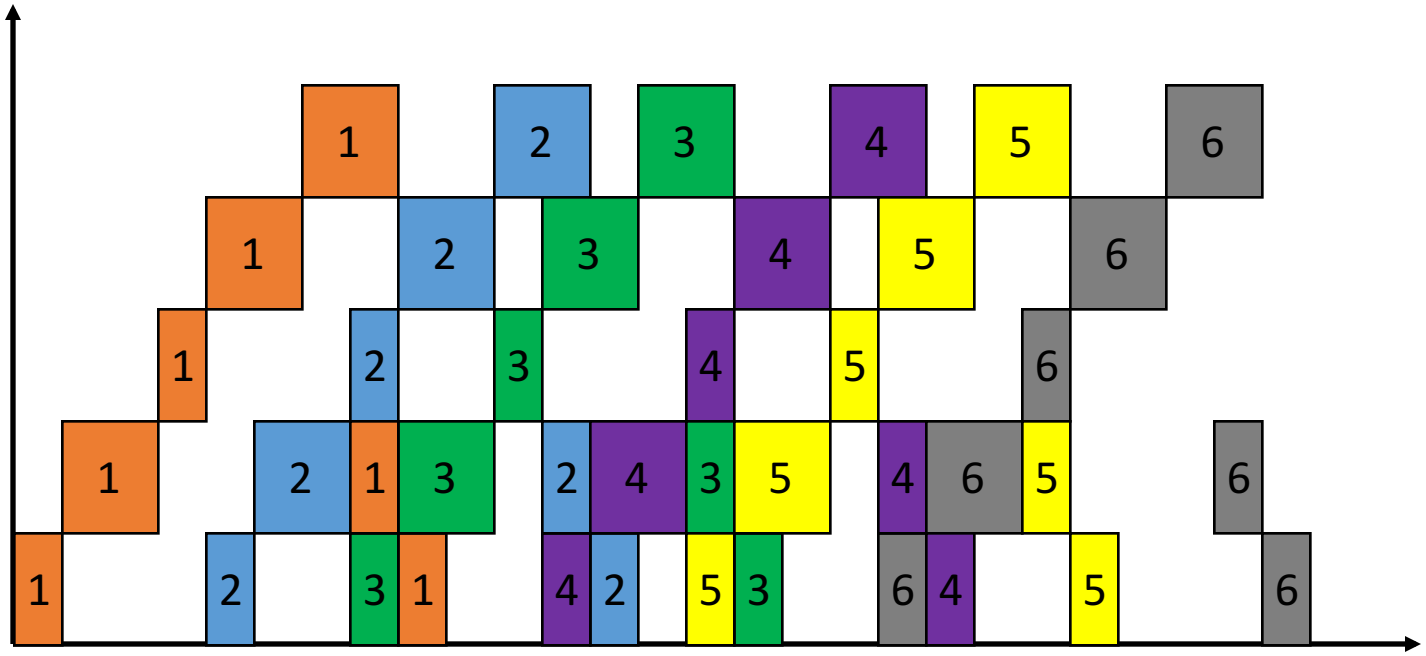
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X_1				X_2			X_3	X_1				X_2		
2		X_1	X_1			X_2	X_2	X_1	X_3	X_3					
3				X_1				X_2			X_3				
4					X_1	X_1			X_2	X_2		X_3	X_3		
5							X_1	X_1			X_2	X_2		X_3	X_3

按(4,3)进行调度

$$T_p = 6/27$$

$$S_p = (6 * 10)/27 = 20/9$$

$$e = (6 * 10)/(27 * 5) = 12/27$$

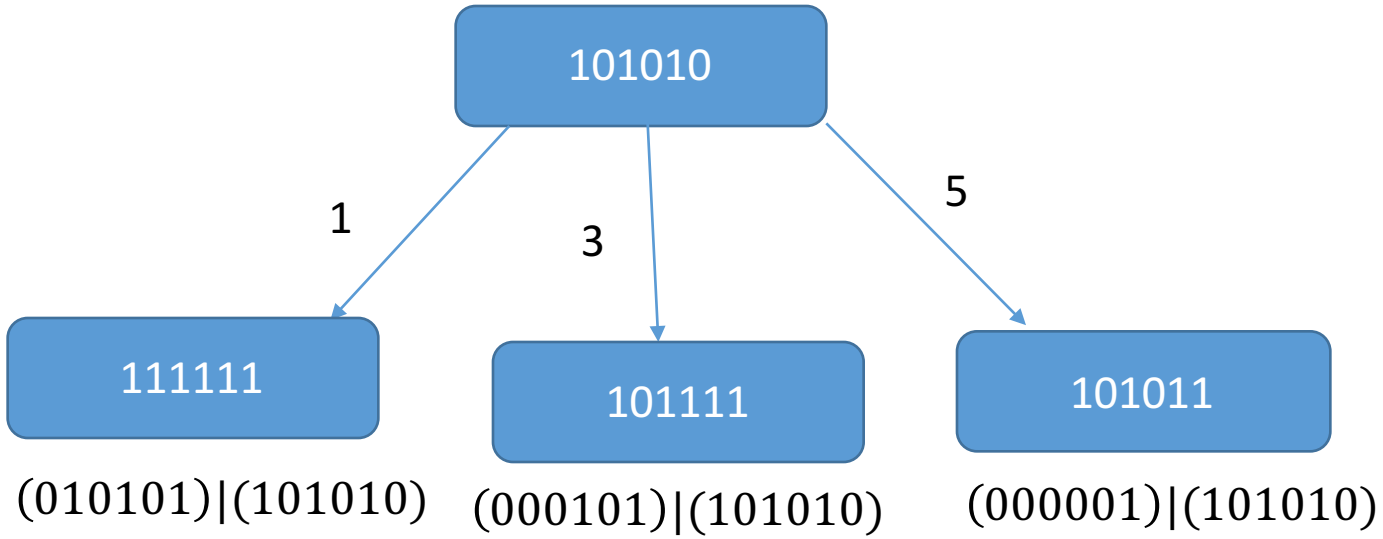


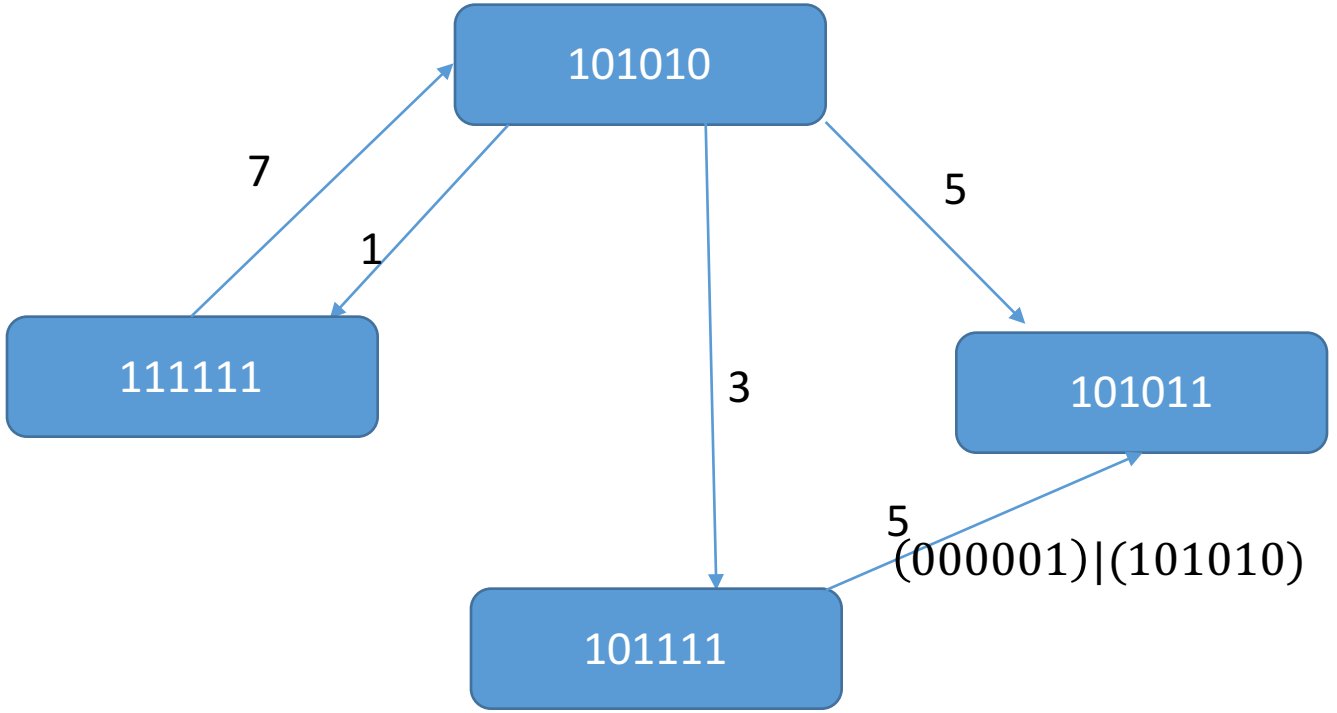
举例

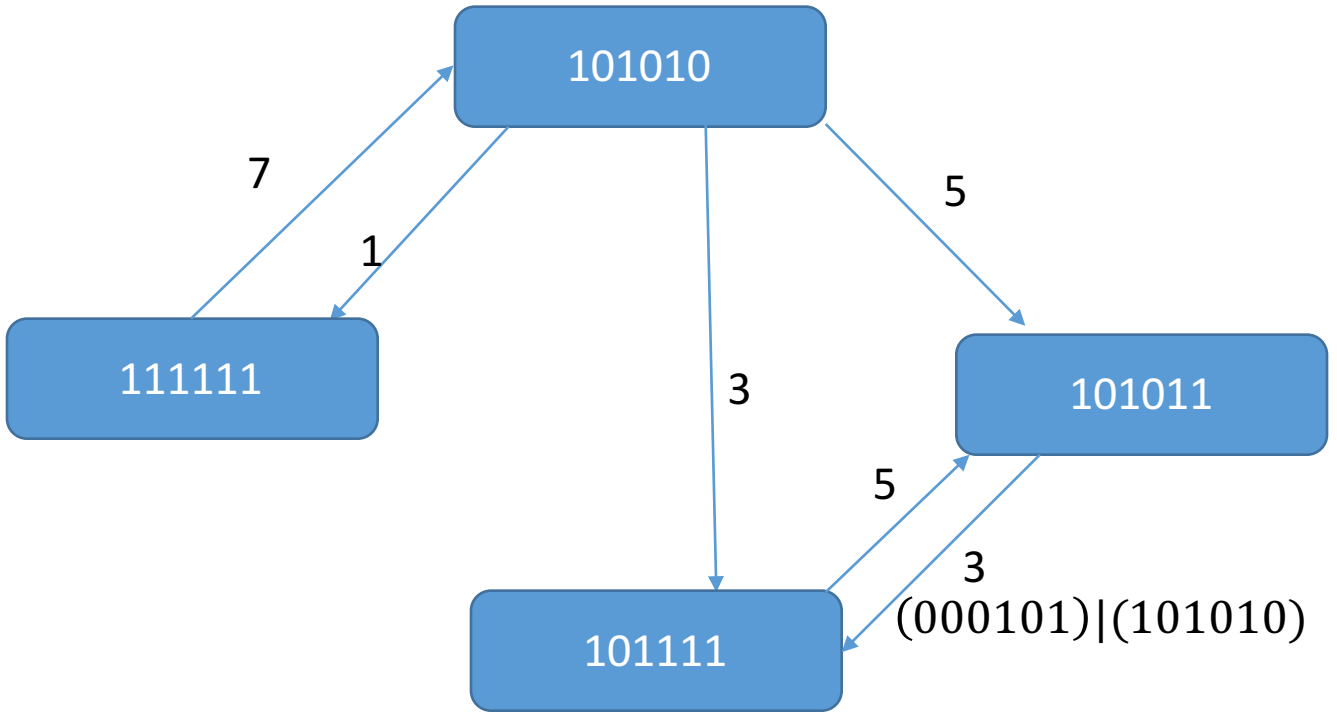
- 在一个4段流水线处理机上需要经过7拍才能完成一个任务

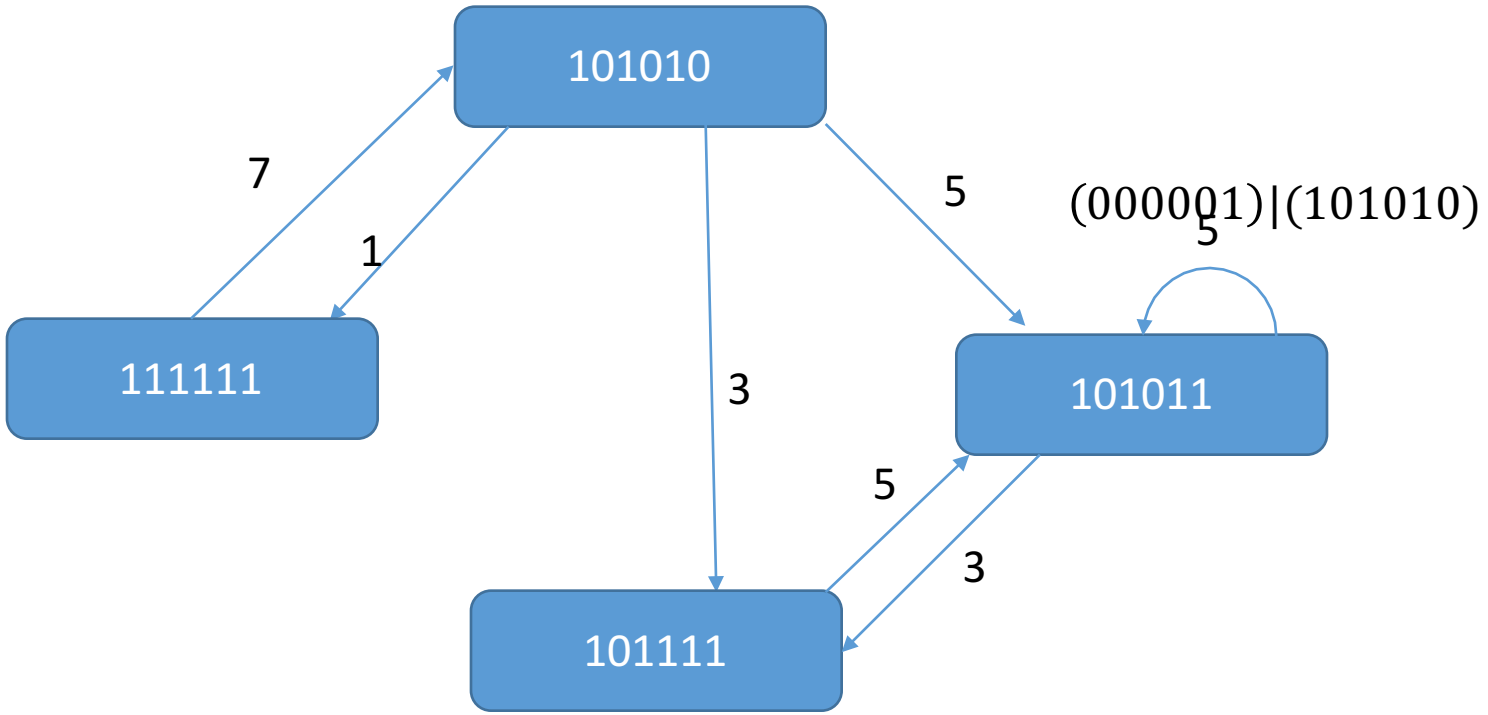
	1	2	3	4	5	6	7
s1	X				X		X
s2		X		X			
s3			X				
s4				X		X	

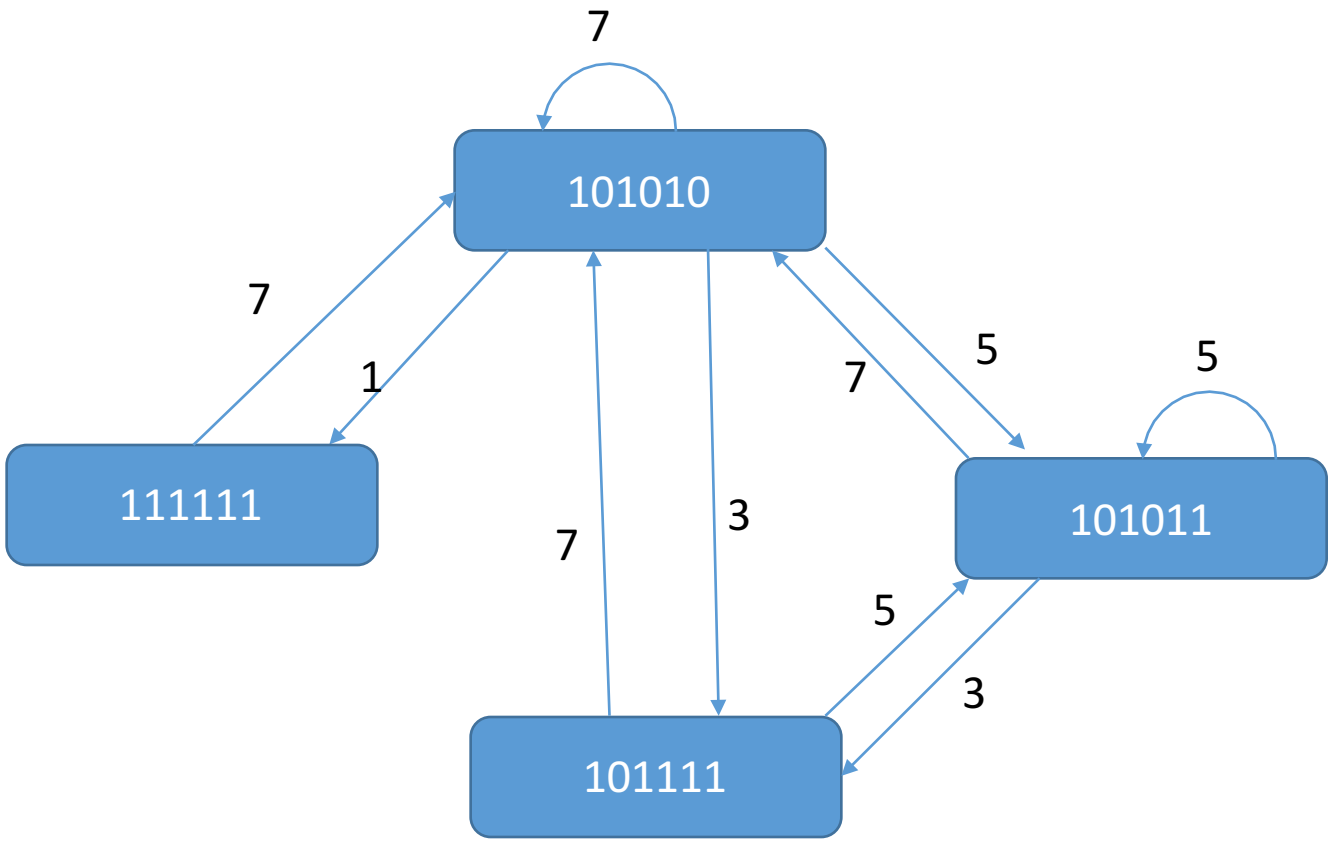
- 延迟禁止表 F={2, 4, 6, 7}
- 初始冲突向量 C=(101010)




101010
7
1
5
111111
3
101011
5
(000001)|(101010)
101111


101010
7
1
5
111111
3
101011
5
3
(000101)|(101010)
101111


101010
7
1
5
(000001)|(101010)
5
111111
3
101011
5
3
101111



调度方案	平均延迟
(1, 7)	4
(3, 5)	4
(5, 3)	4
(5)	5

- 输入6个任务
  - (1, 7): 完成时间＝1+7+1+7+1+7＝24
  - (3, 5): 完成时间＝3+5+3+5+3+7＝26
  - (5, 3): 完成时间＝5+3+5+3+5+7＝28

多功能流水线

- 二功能动态流水线

拍号 n

段号 k	1	2	3	4	5
1	A	B		A	B
2		A		B	
3	B		AB		A

- 交叉冲突向量（Cross-collision Vector）

$$V_{AB} = (1011), V_{BA} = (1010), V_{AA} = (0110), V_{BB} = (0110)$$

- V_{AA}和V_{BB}分别表示都按A功能和B功能流水时，后继任务流入流水线的冲突向量
- V_{AB}表示先前按B功能流水流入的任务与后继按A功能流水流入的任务之间的冲突向量
- V_{BA}表示先前按A功能流水流入的任务与后继按B功能流水流入的任务之间的冲突向量

- 冲突矩阵

$$M_A = \begin{bmatrix} 0110\ (AA) \\ 1010\ (BA) \end{bmatrix}, M_B = \begin{bmatrix} 1011(AB) \\ 0110(BB) \end{bmatrix}$$

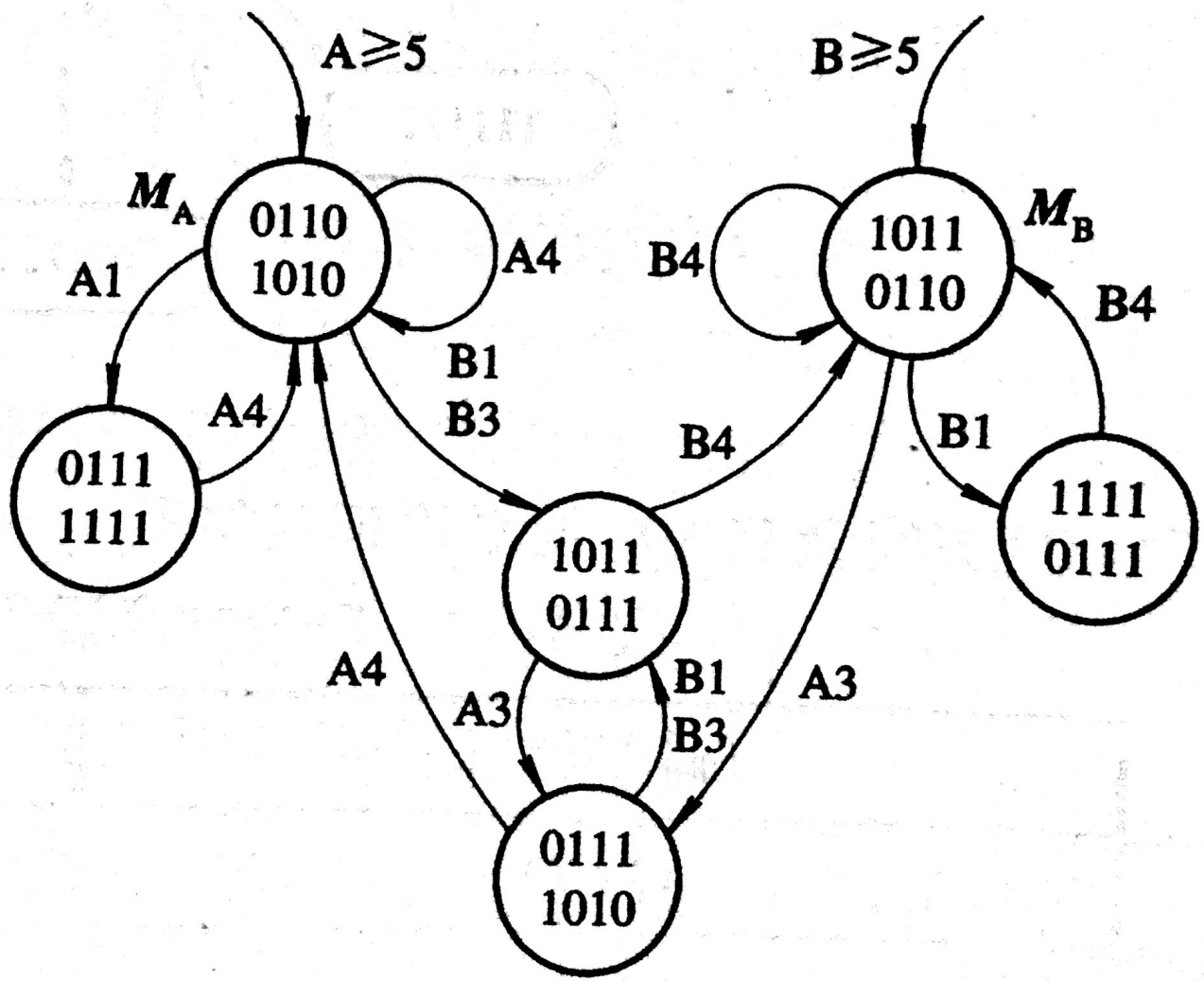
- M_A表示按A功能流水刚流入一个任务后与其他功能流水流入后继任务的所有禁止间隔拍数，M_B表示按B功能流水刚流入一个任务后与其他功能流水流入后继任务的所有禁止间隔拍数

- 按A功能流入一个任务后，根据$V_{AA}=(0110)$，可以隔1拍或者4拍后流入一个A功能任务

$$M'_A=\begin{bmatrix}0011\ (AA)\\0101\ (BA)\end{bmatrix}\ OR\quad M_A=\begin{bmatrix}0110\ (AA)\\1010\ (BA)\end{bmatrix}=\begin{bmatrix}0111\ (AA)\\1111(BA)\end{bmatrix}$$

- 根据$V_{BA}=(1010)$，可在第一拍或者第三拍进行B功能的新任务送入

$$M'_A=\begin{bmatrix}0011\ (AA)\\0101(BA)\end{bmatrix}\ OR\quad M_B=\begin{bmatrix}1011(AB)\\0110(BB)\end{bmatrix}=\begin{bmatrix}1011(AB)\\0111(BB)\end{bmatrix}$$


A≥5
B≥5
M_A
M_B
0110
1010
1011
0110
A1
A4
B4
B4
B1
B3
A4
B4
B1
0111
1111
1011
0111
1111
0111
A4
A3
B1
B3
A3
0111
1010

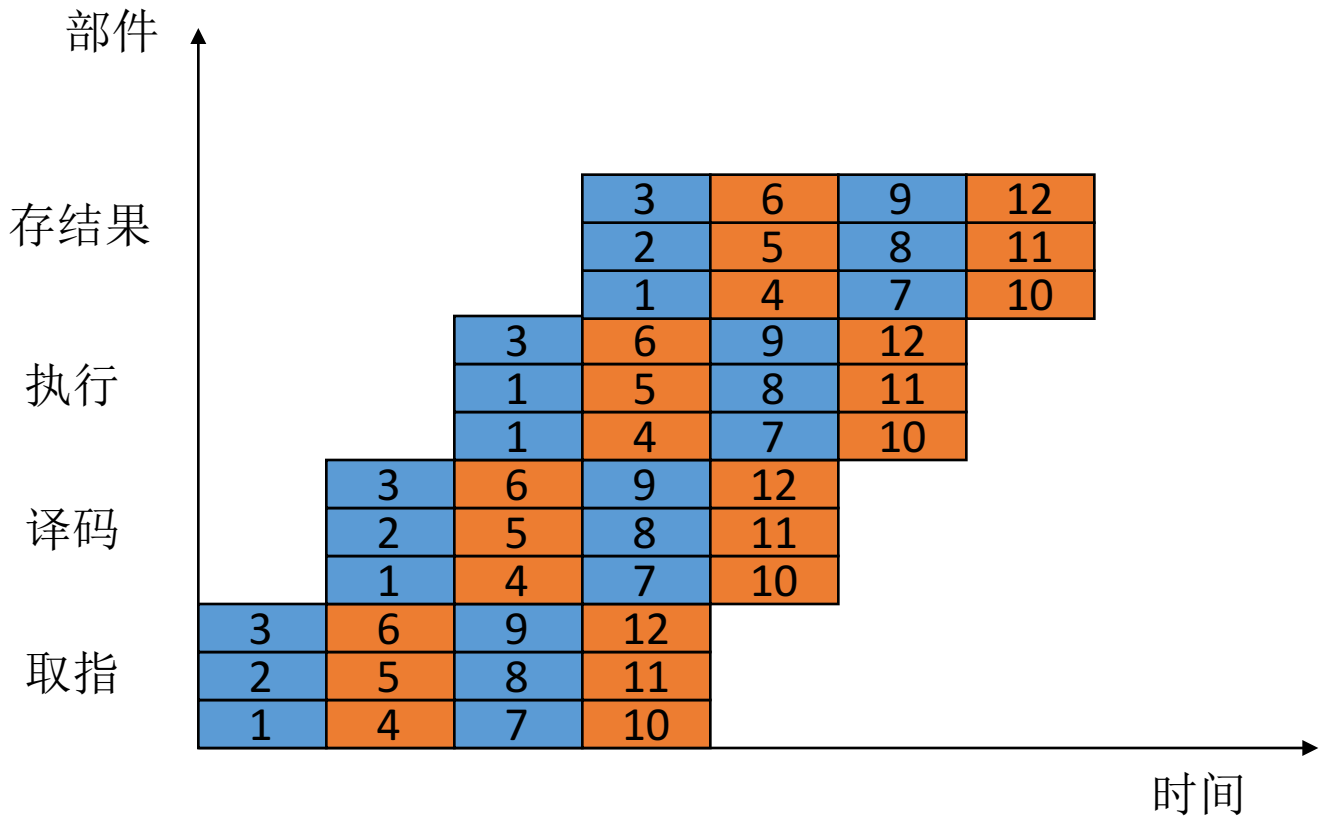
指令级高度并行的超级计算机

- 超标量处理机
- 超长指令字处理机
- 超流水线处理机

超标量处理机

- 采用多指令流水线（度=m）
- 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。
- 适合于求解稀疏向量、矩阵
- IBM RS/6000、DEC 21064、Intel i960CA、Tandem Cyclone（飓风）等

超标量处理机（续）

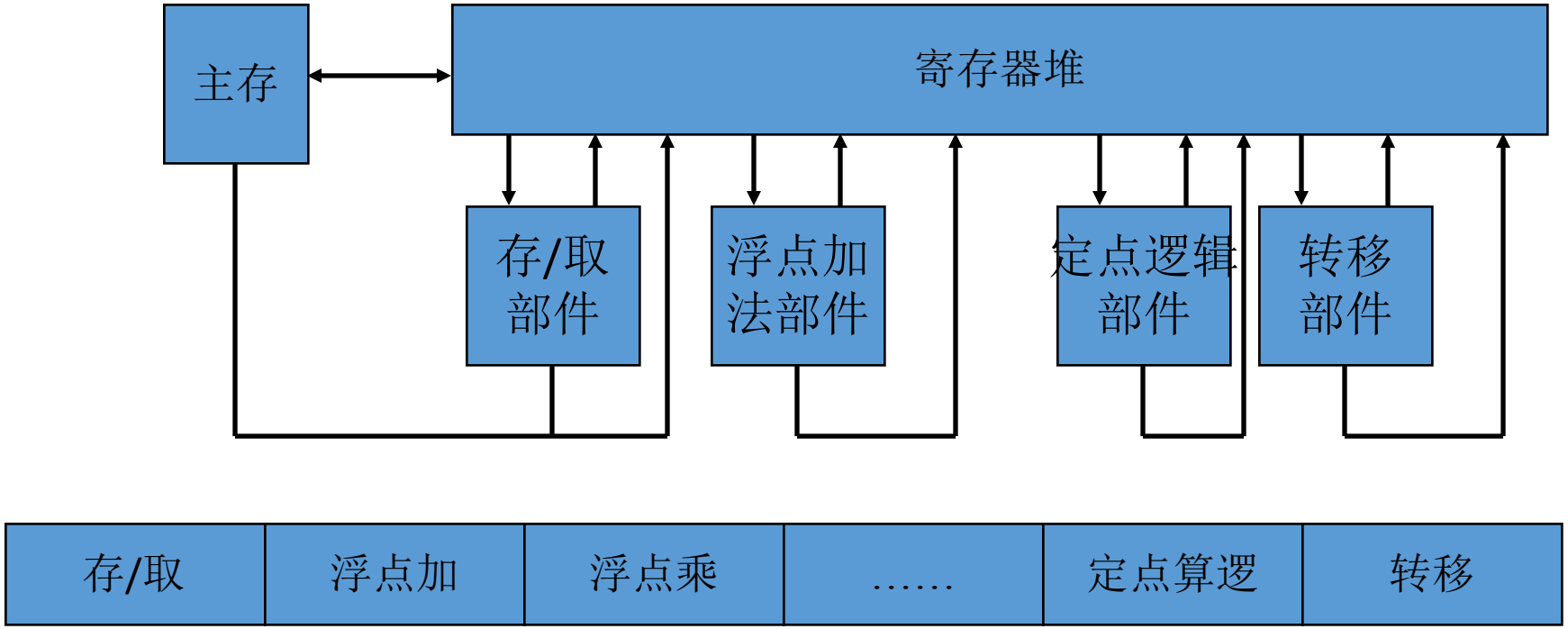


度m=3的超标量处理机时空图

超长指令字处理机(VLIW)

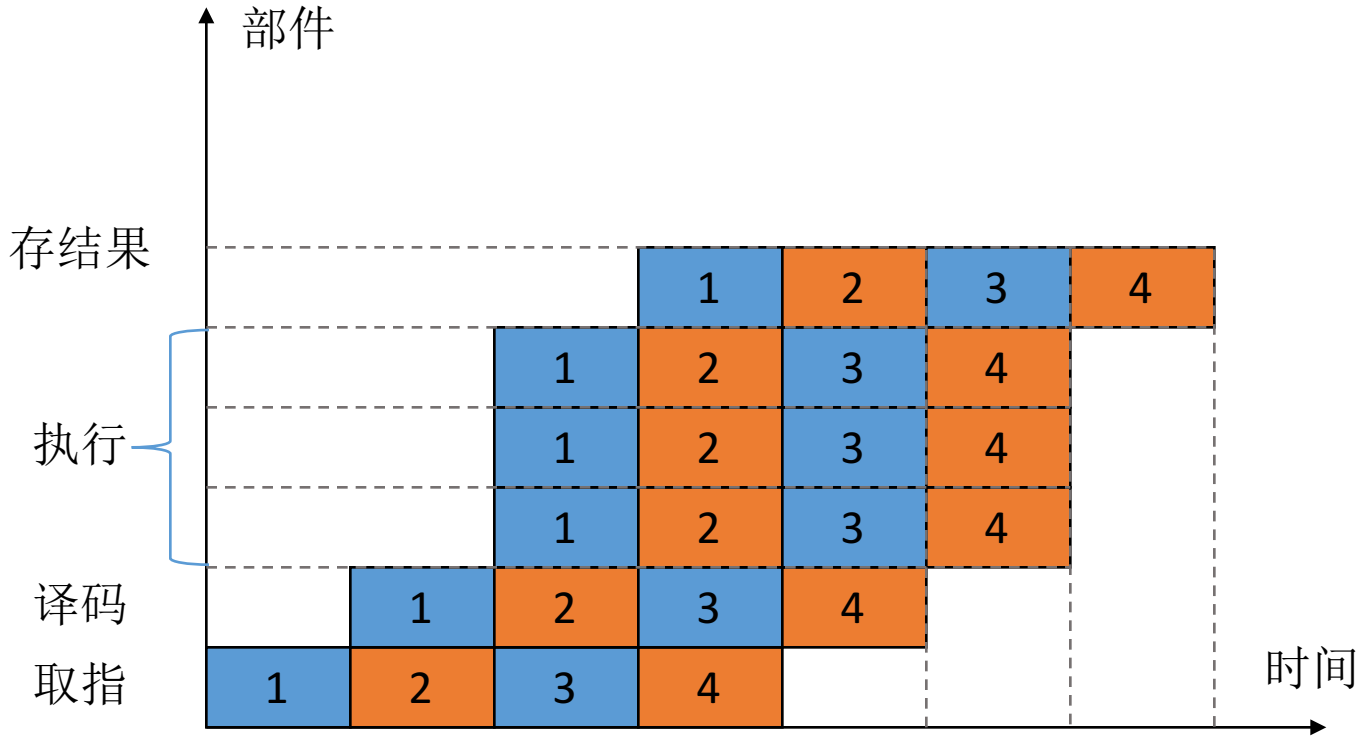
- VLIW (Very Long Instruction Word)
- 是将水平型微码和超标量处理两者结合的结构
- 指令字长可达数百位，多个功能部件并发工作，共享大容量寄存器堆。
- 在编译过程中，由编译器找出指令间的潜在并行性，将多个功能并行执行的不相关或无关操作先行压缩组合在一起，形成一条有多个操作段的超长指令，可以在执行时不在对指令间并行性进行检测，而直接由控制机器中多个相互独立的功能部件并行操作
- 是一种单指令多操作码多数据的系统结构

超长指令字处理机（续）



典型的VLIW处理机组成和指令格式

超长指令字处理机（续）



度$m = 3$的执行时空图

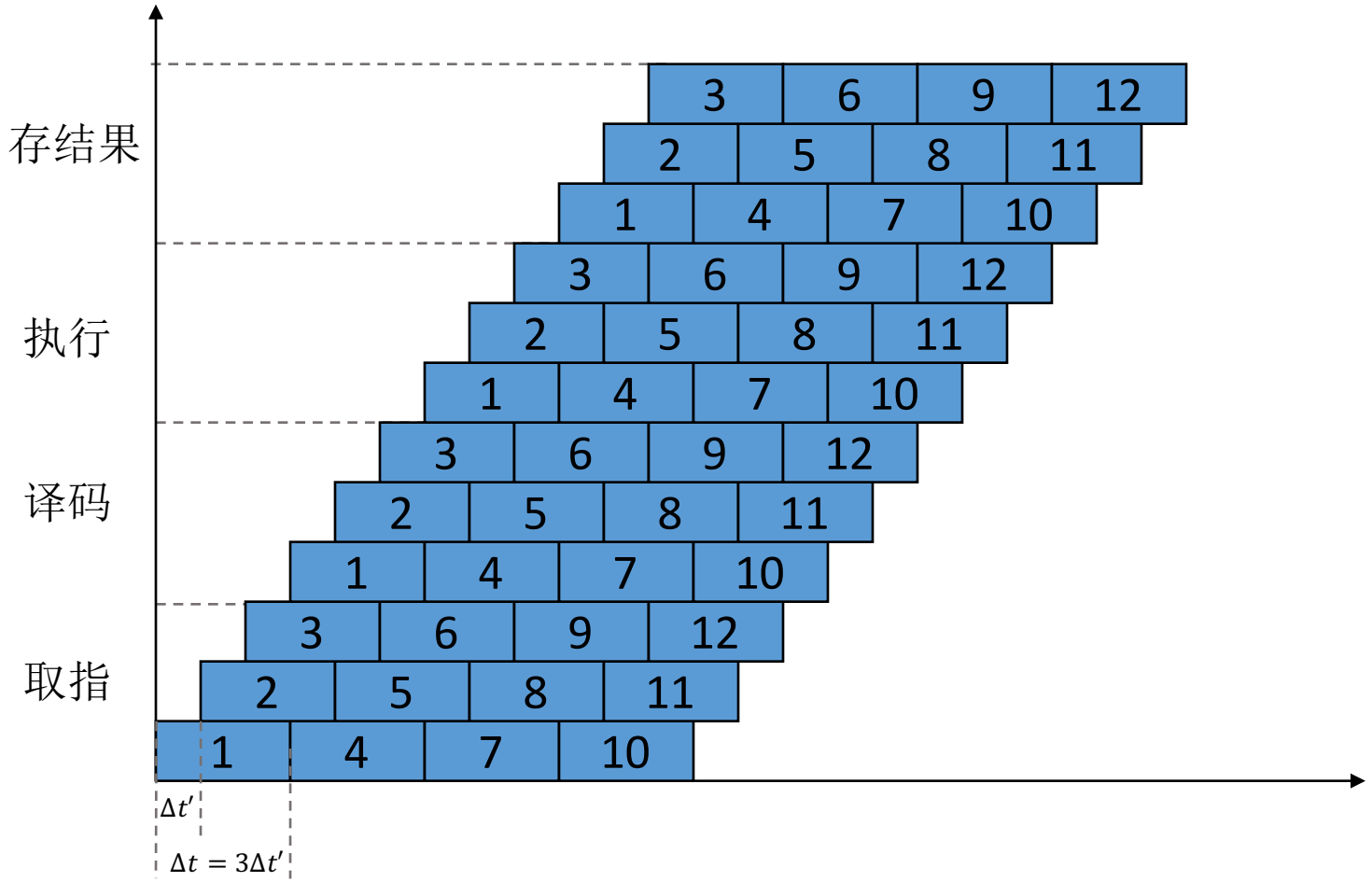
超长指令字处理机（续）

- 每条指令所需要的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，从而可使指令级并行性较高
- 性能取决于代码压缩效率，其结构的目标码与一般计算机不兼容
- 指令字很长且格式固定，容易造成存储空间的浪费
- 需要同时设计编译程序和系统结构，两者关系紧密，兼容性不高

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。
- 提高处理机性能的不同方法：
  - 超标量处理机是通过增加硬件资源为代价来换取处理机性能的。
  - 超流水线处理机则通过各硬件部件充分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

超流水线处理机（续）



每个时钟周期分时发送3条指令的超流水线

- 一台有k段流水线的m度超流水线处理机，执行完N条指令的时间为

$$\left(k+\frac{N-1}{m}\right)\Delta t$$

- 相对常规流水线处理机加速比为

$$S_p=\frac{(k+N-1)\Delta \mathrm{t}}{\left(k+\frac{N-1}{m}\right)\Delta \mathrm{t}}=\frac{m(k+N-1)}{mk+N-1}$$

超标量超流水线处理机

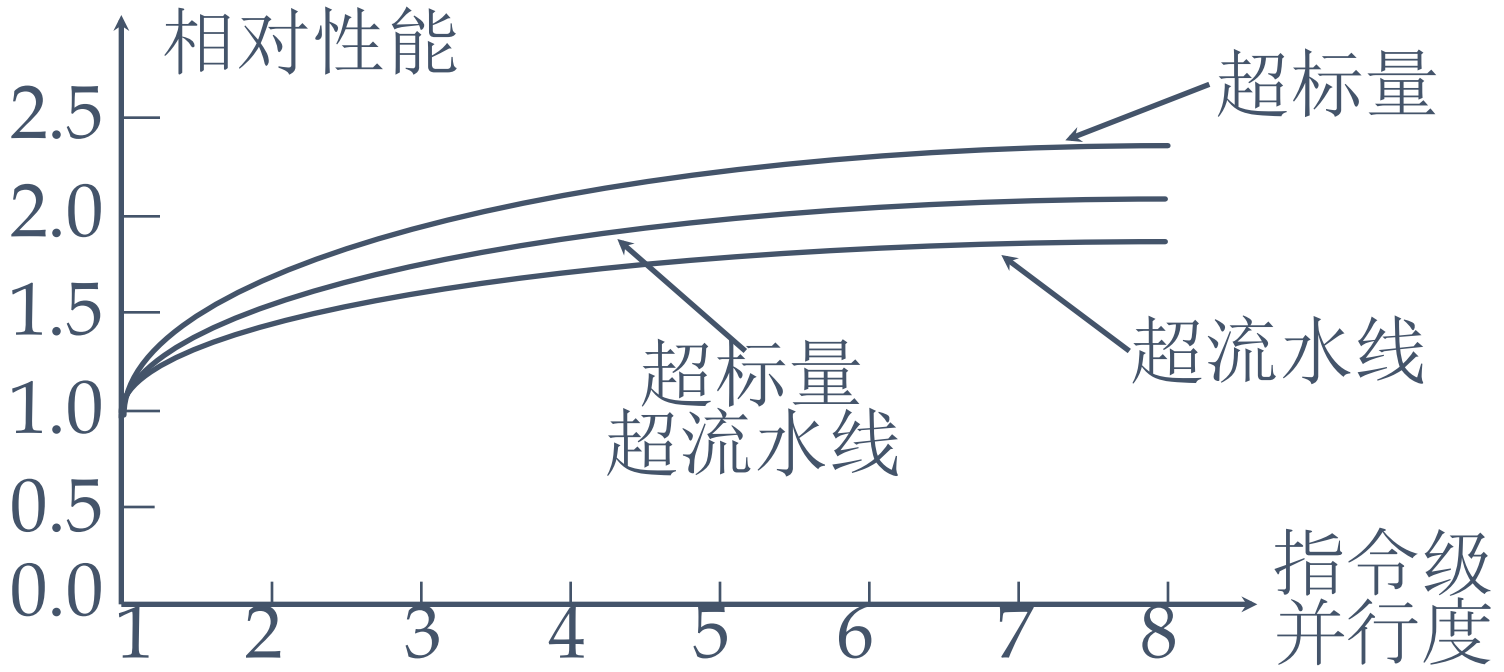
- 把超标量与超流水线技术结合在一起，就成为超标量超流水线处理机

- 指令执行时序
  - 超标量超流水线处理机在一个$\Delta t' = \Delta t/n$时钟周期内分时发射k条指令，相当于每个时钟周期Δt总共发射指令nk条。

每时钟周期发射3次，每次3条指令

三种指令级并行处理机性能比较

- 超标量处理机、超流水线处理机和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性能曲线。



结论

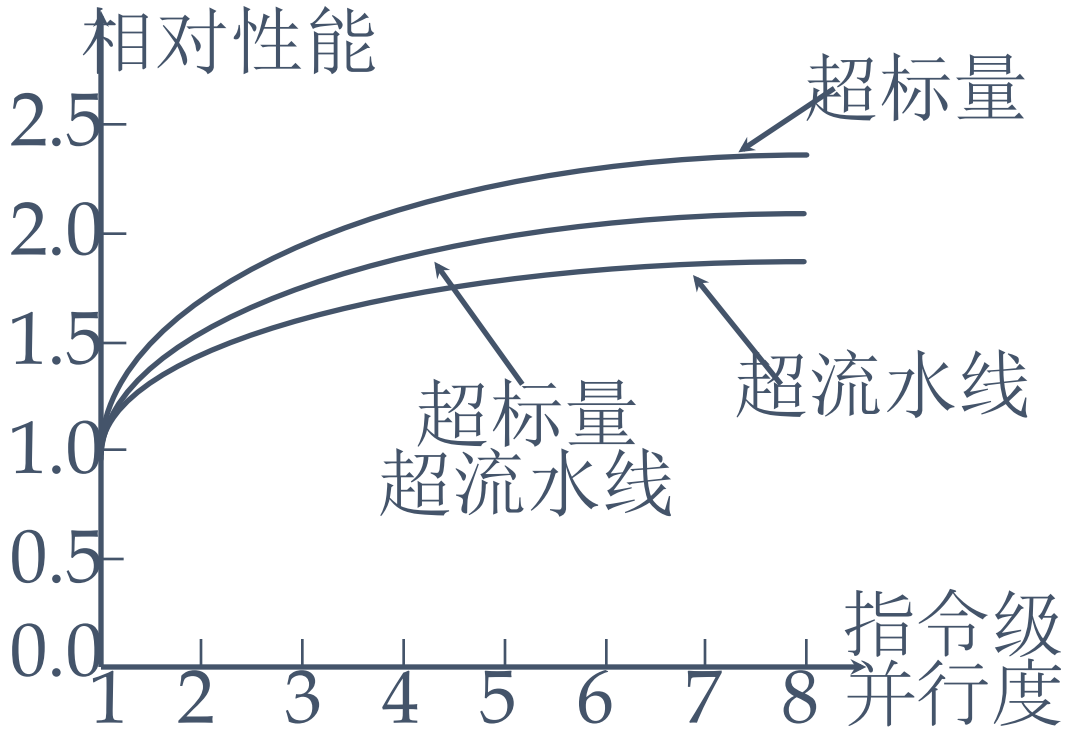
- 三种处理机的性能关系超标量处理机的相对性能最高，其次是超标量超流水线处理机，超流水线处理机的相对性能最低，主要原因如下：
  - 超标量处理机在每个时钟周期的一开始就同时发射多条指令，而超流水线处理机则要把一个时钟周期平均分成多个流水线周期，每个流水线周期发射一条指令；因此，超流水线处理机的启动延迟比超标量处理机大。

结论（续）

- 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。
- 在指令执行过程中的每一个功能段，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；因此，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。

结论（续）

- 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。



结论（续）

- 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。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。

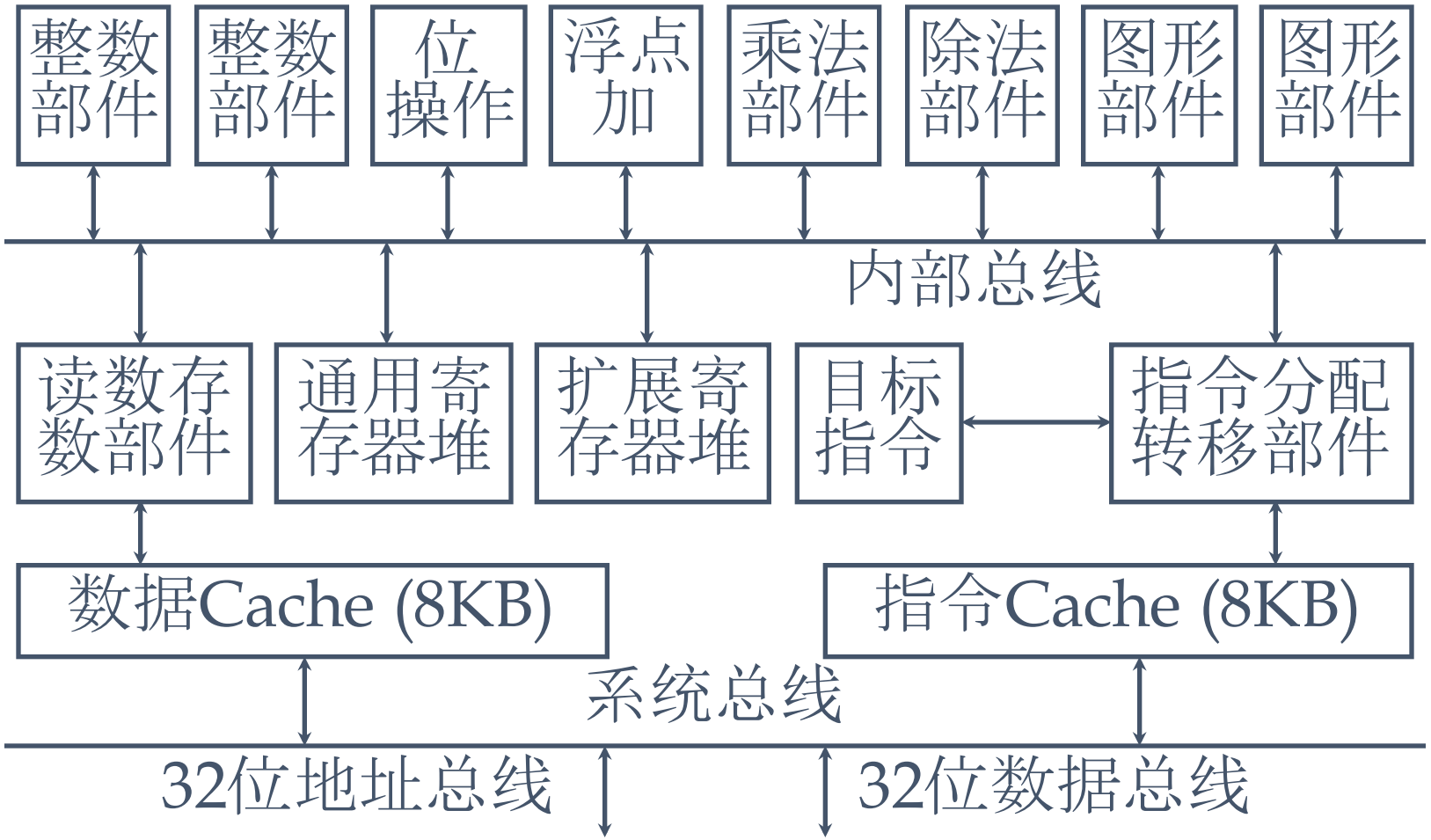
超标量处理机基本结构

- 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
- 超标量处理机典型结构：
  - 多条指令流水线
  - 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
  - 超标量处理机的指令级并行度大于1

举例： 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中，各为8KB，采用两路组相联方式。
- 转移目标指令Cache，在有两路分支时，存放其中一路分支上的指令

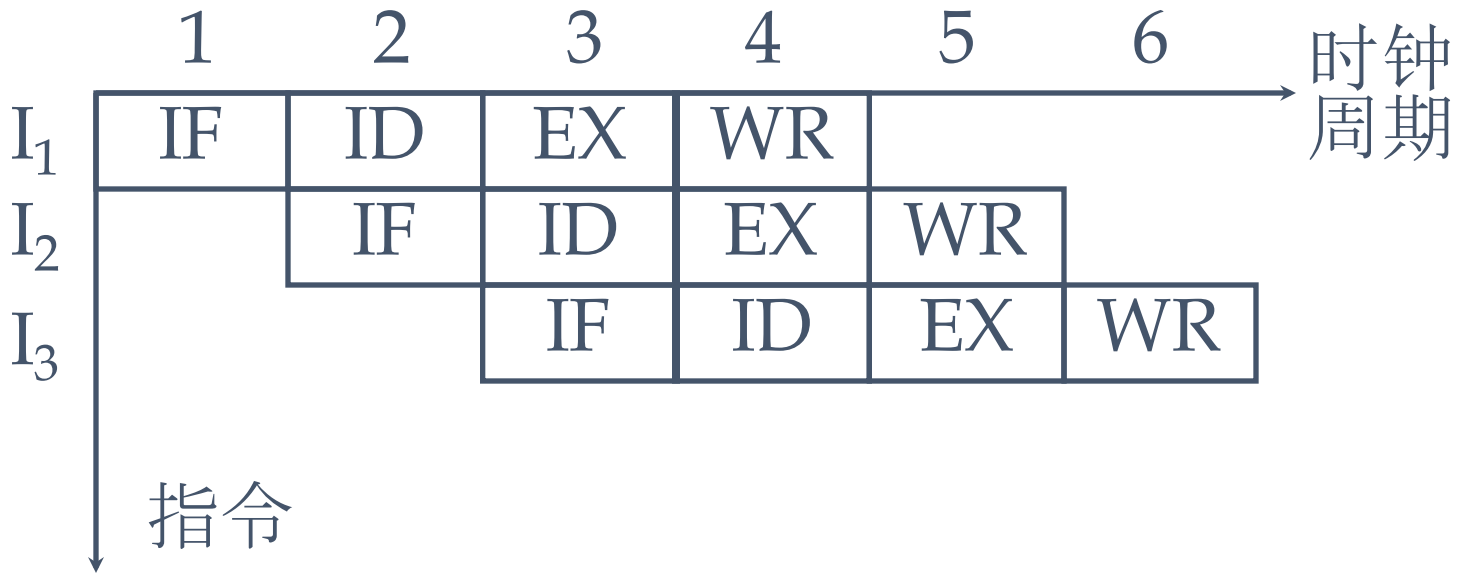
超标量处理机MC88110的结构

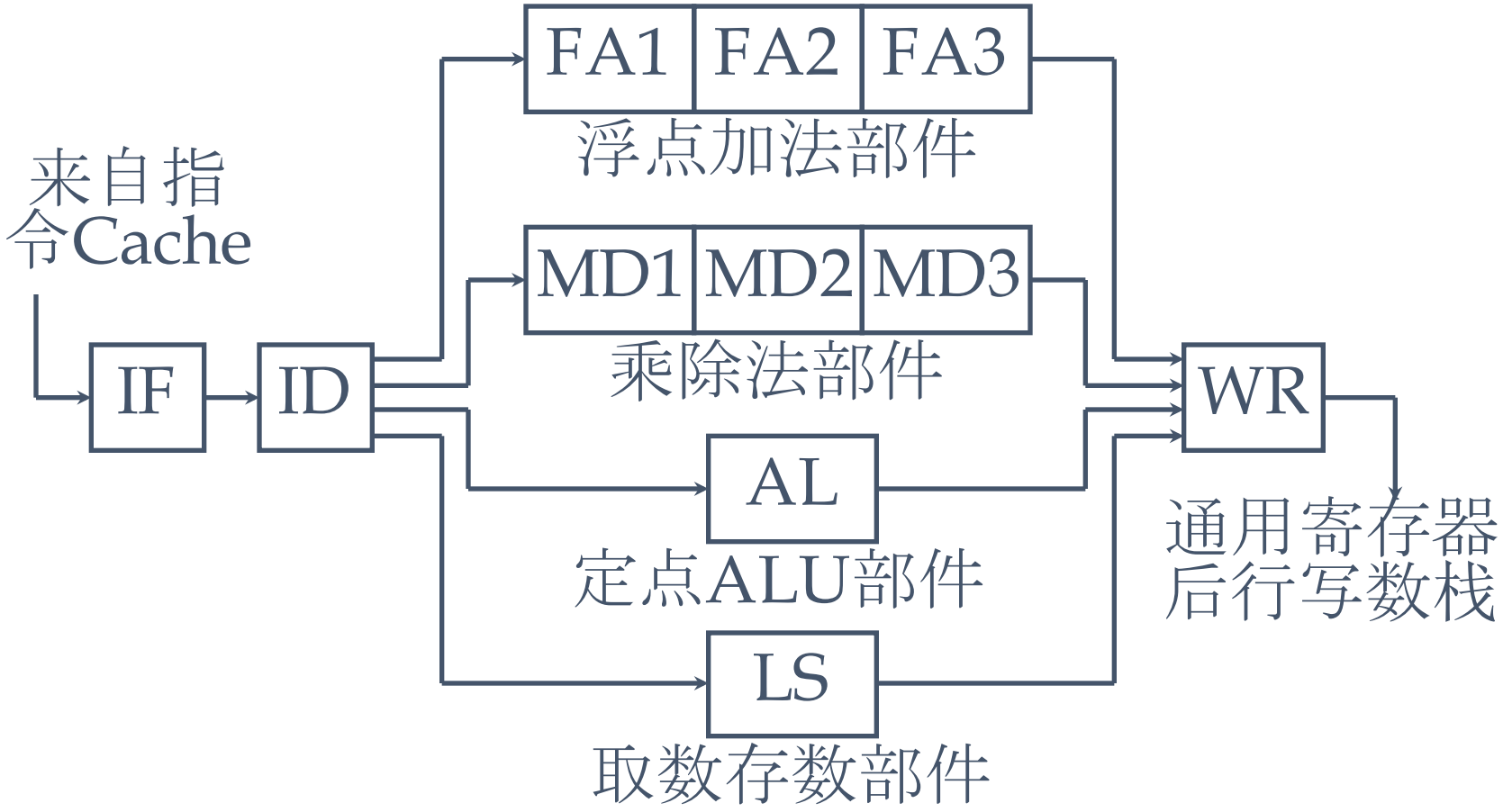


单发射与多发射

- 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
  - 设计目标是每个时钟周期平均执行一条指令，ILP的期望值1

单发射处理机的指令流水线时空图

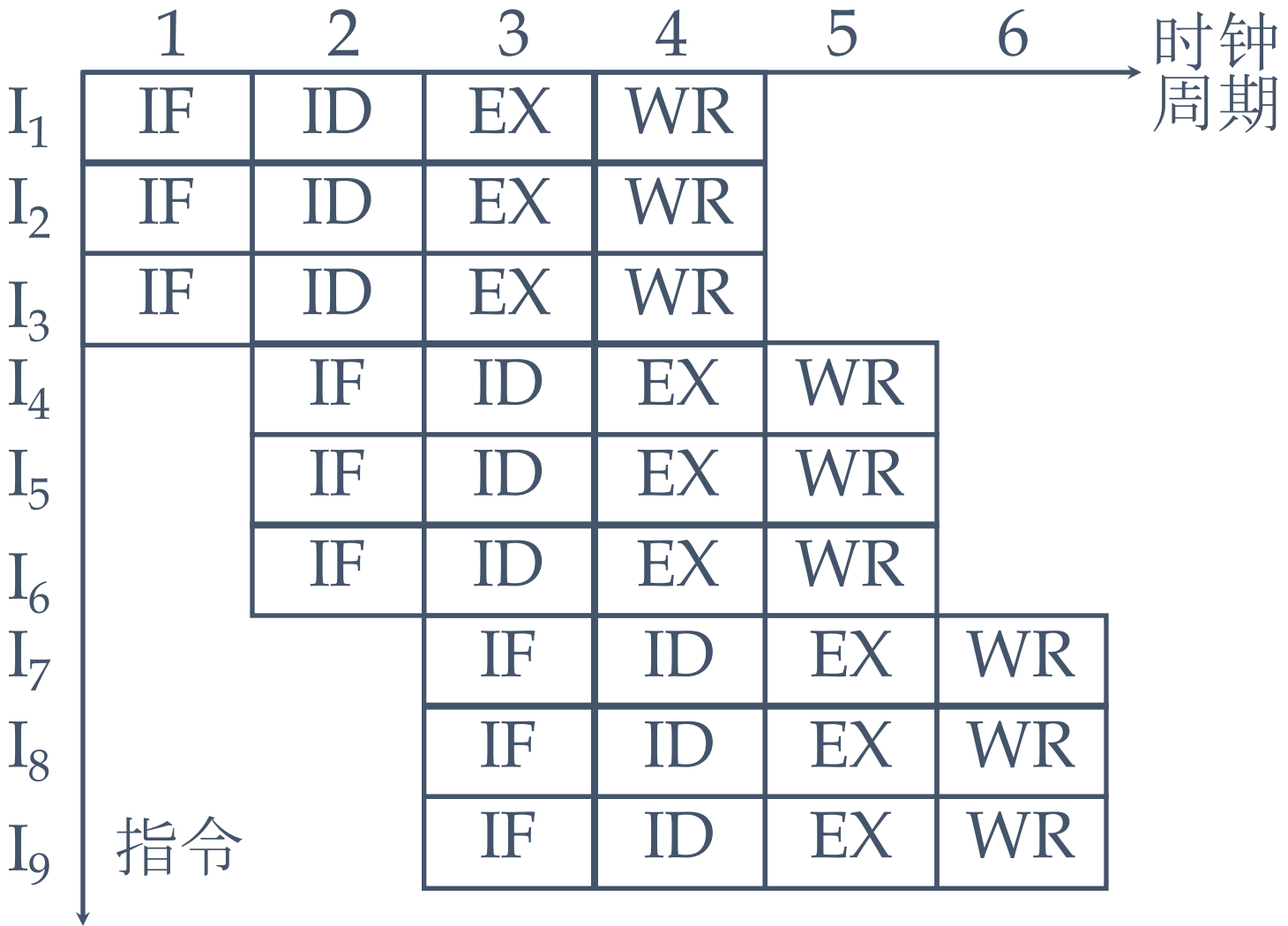


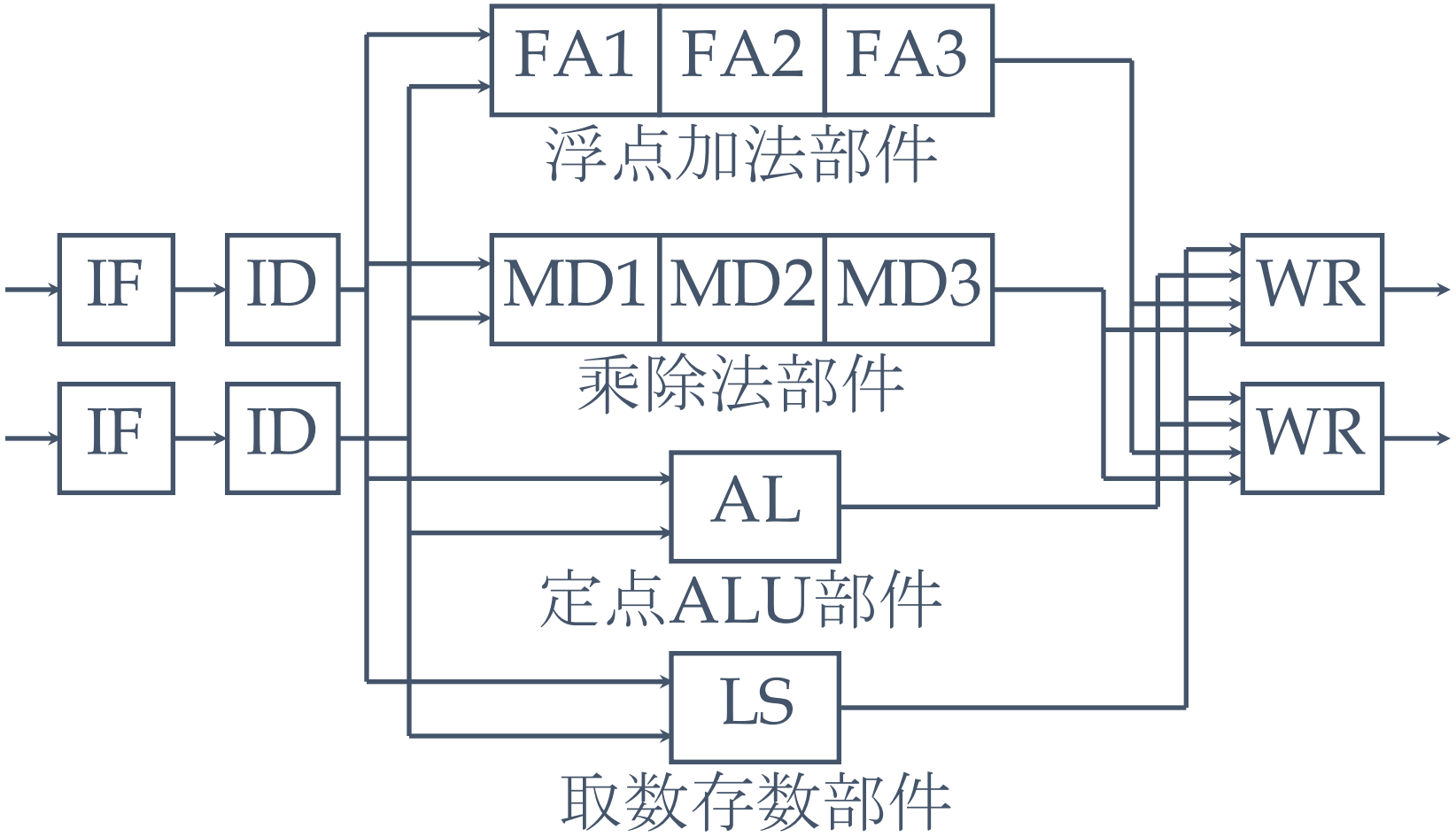

来自指
令Cache
IF
ID
FA1
FA2
FA3
浮点加法部件
MD1
MD2
MD3
乘除法部件
AL
定点ALU部件
LS
取数存数部件
WR
通用寄存器
后行写数栈

单发射与多发射（续）

- 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
  - 设计目标是每个时钟周期平均执行多条指令，ILP的期望值大于1

多发射处理机的指令流水线时空图

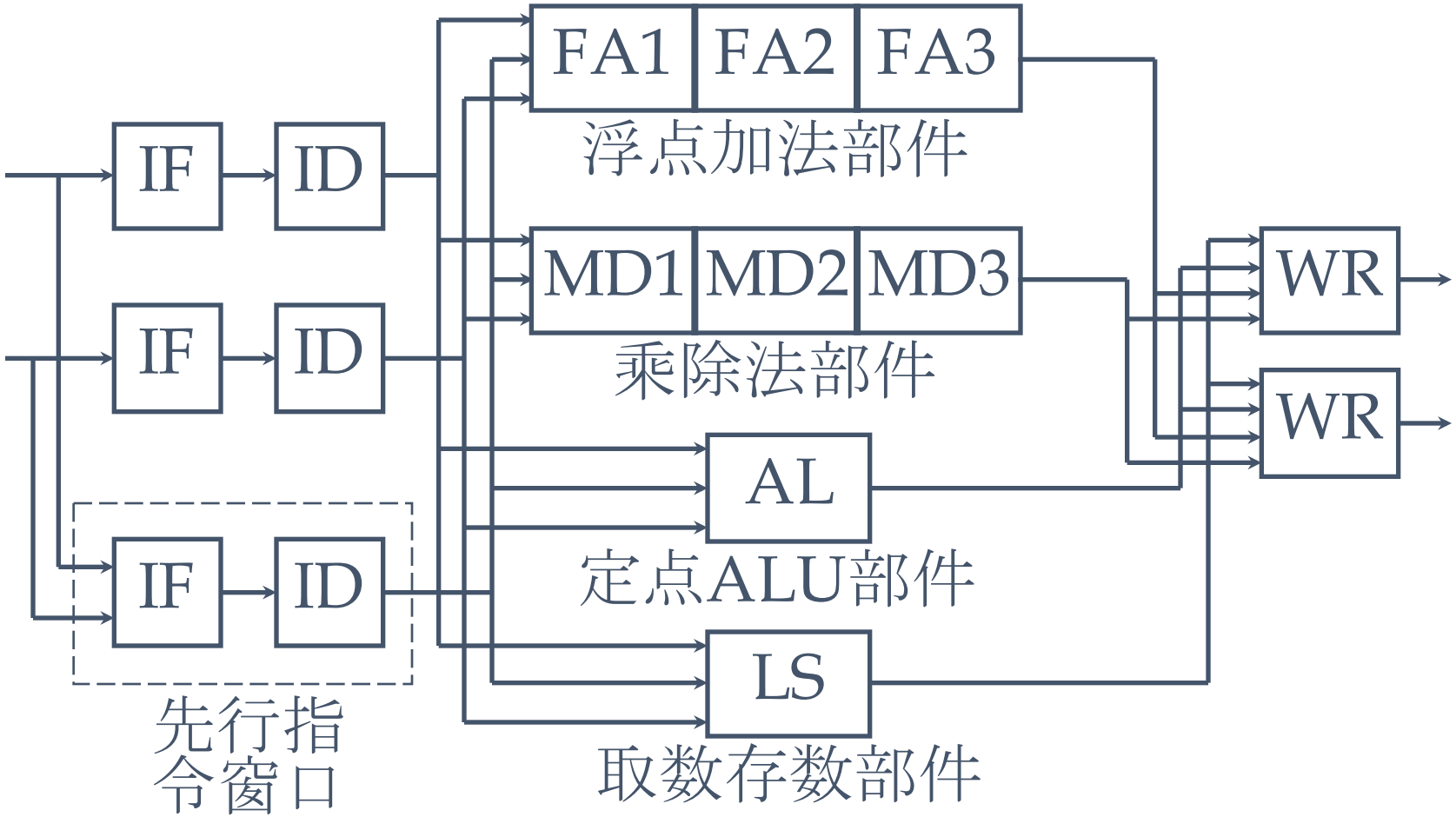



FA1
FA2
FA3
浮点加法部件
IF
ID
MD1
MD2
MD3
乘除法部件
WR
IF
ID
WR
AL
定点ALU部件
LS
取数存数部件

- 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
- 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

举例

- Intel公司的i860、i960、Pentium处理机， Motolora公司的MC88110处理机，IBM公司的Power 6000处理机等每个周期都发射两条指令
- TI公司生产的SuperSPARC处理机以及Intel的Pentium III处理机等每个周期发射三条指令
- 操作部件的个数多于每个周期发射的指令条数。4个至16个操作部件
- 超标量处理机的指令级并行度：1＜ILP＜m；m为每个周期发射的指令条数。


FA1
FA2
FA3
浮点加法部件
IF
ID
IF
ID
MD1
MD2
MD3
乘除法部件
WR
WR
AL
定点ALU部件
IF
ID
LS
先行指
令窗口
取数存数部件

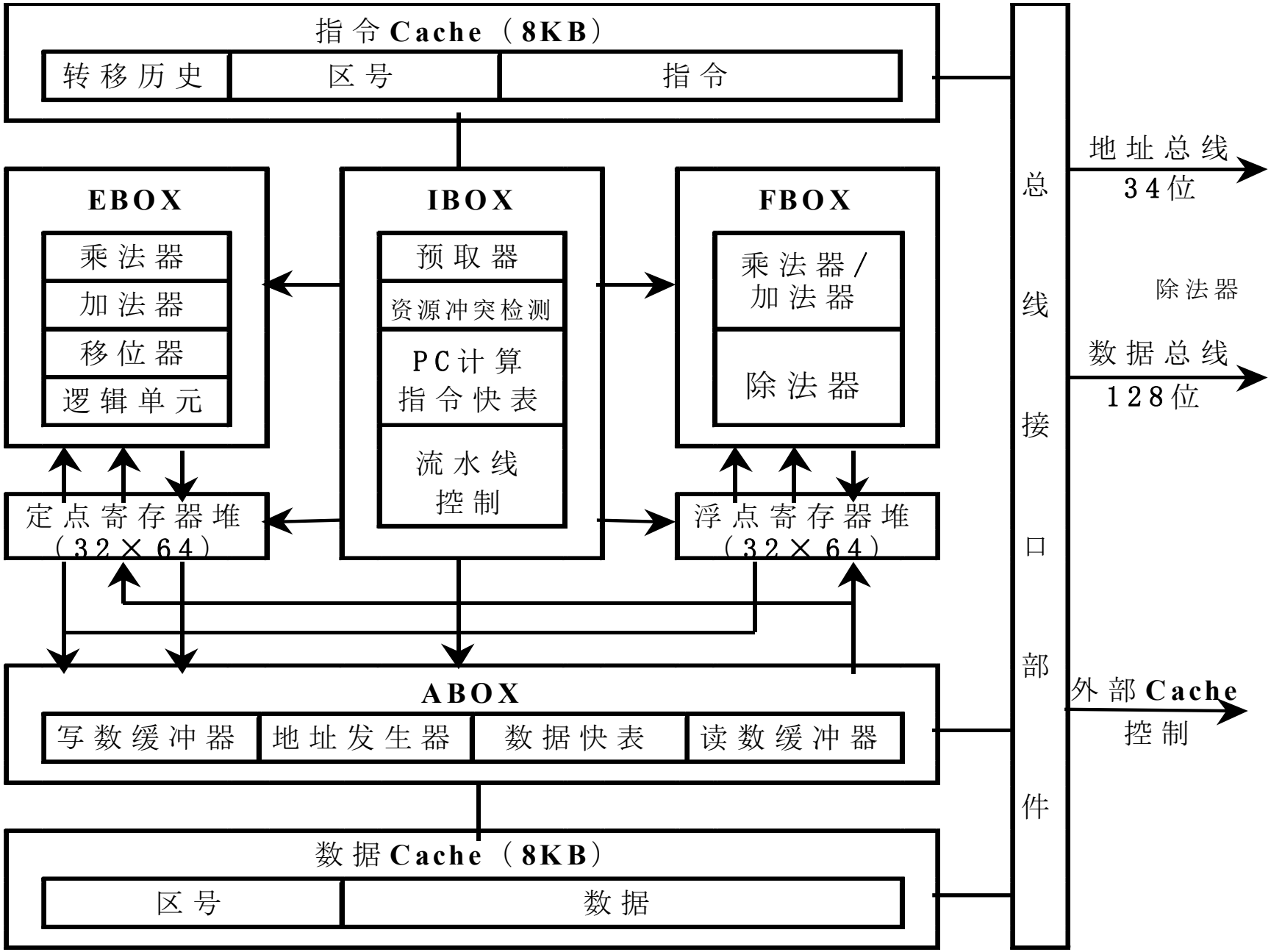
- 超标量处理机：
  - Intel公司的i860, i960, Pentium处理机
  - Motolora公司的MC88110
  - IBM公司的Power 6000
  - SUN公司的SuperSPARC等
- 超流水线处理机：
  - SGI公司的MIPS R4000, R5000, R10000等
- 超标量超流水线处理机：
  - DEC公司的Alpha等

举例： DEC公司的Alpha

- DEC公司的Alpha处理机采用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。

举例： DEC公司的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。 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。



Alpha 21064 处理机结构