第5章
流水线和向量处理机

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

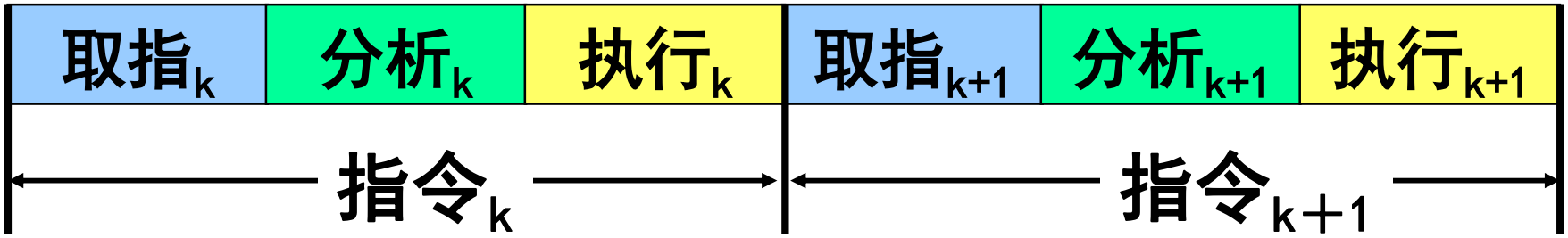
- 为加快机器语言程序的执行，要从系统整体的角度来考虑。可以从两个方面来考虑：
  - 提高每一条指令的执行速度
  - 提高多条指令的执行速度
- 为此，
  - 一方面需要选用更高速的器件、采取更好的算法，提高指令内部各微操作的并行度
  - 另一方面需要通过控制机构采用同时解释两条、多条甚至整段程序的控制方式

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

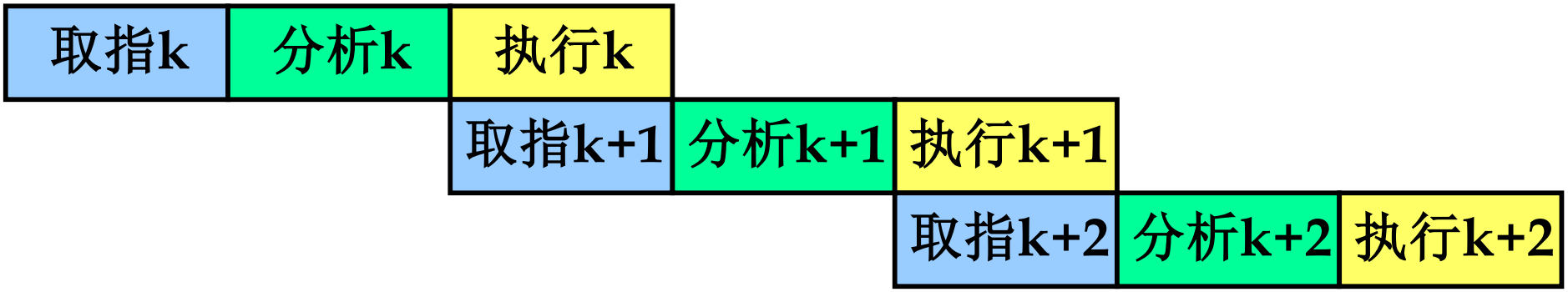
- 执行n条指令所用的时间为：

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

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



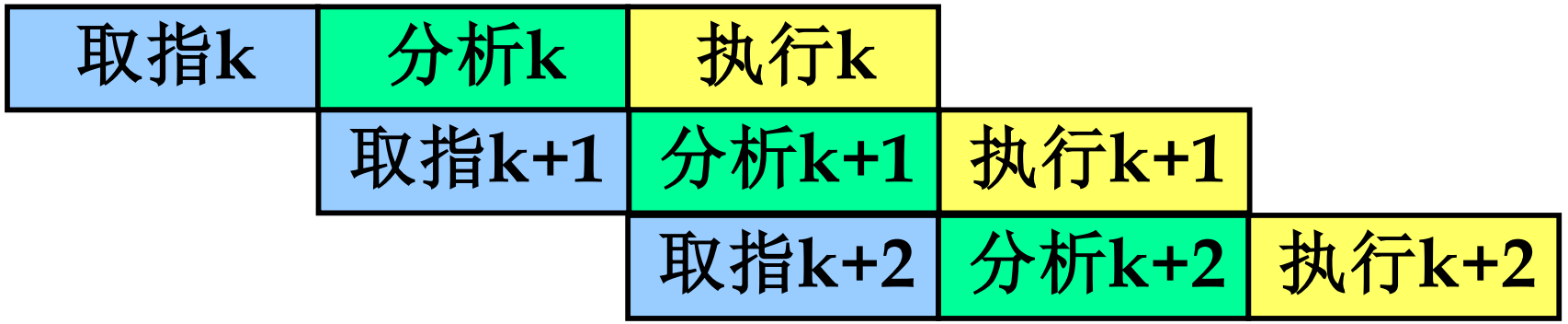
如果三个过程的时间相等，都为t，则执行n条指令的时间为：T=(1+2n)t

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

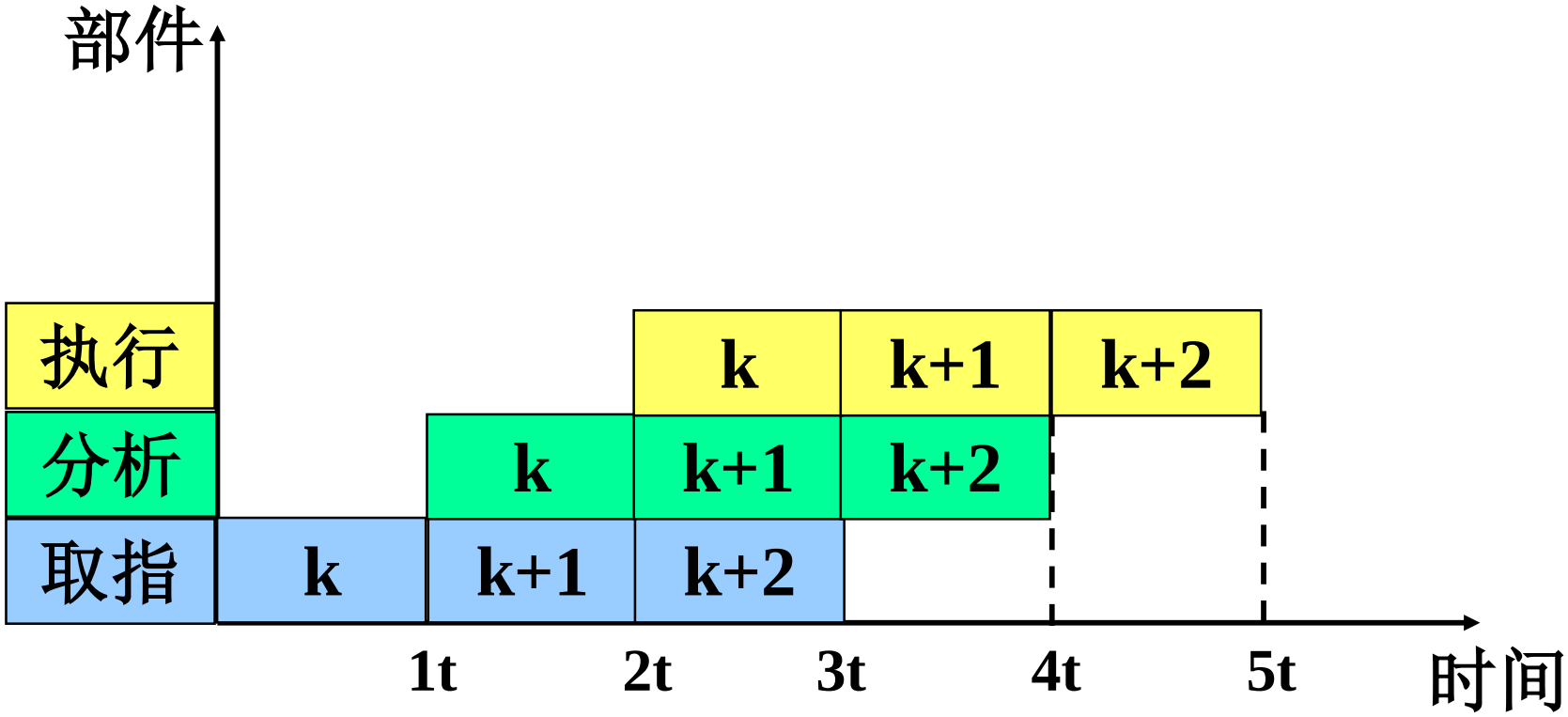
- 每次重叠执行三条指令



- 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

  } “结构相关”或“资源相关”

  - 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）

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$_k$”与“分析$_{k+1}$”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储单元。
  - 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、可同时访问的的存储器，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。

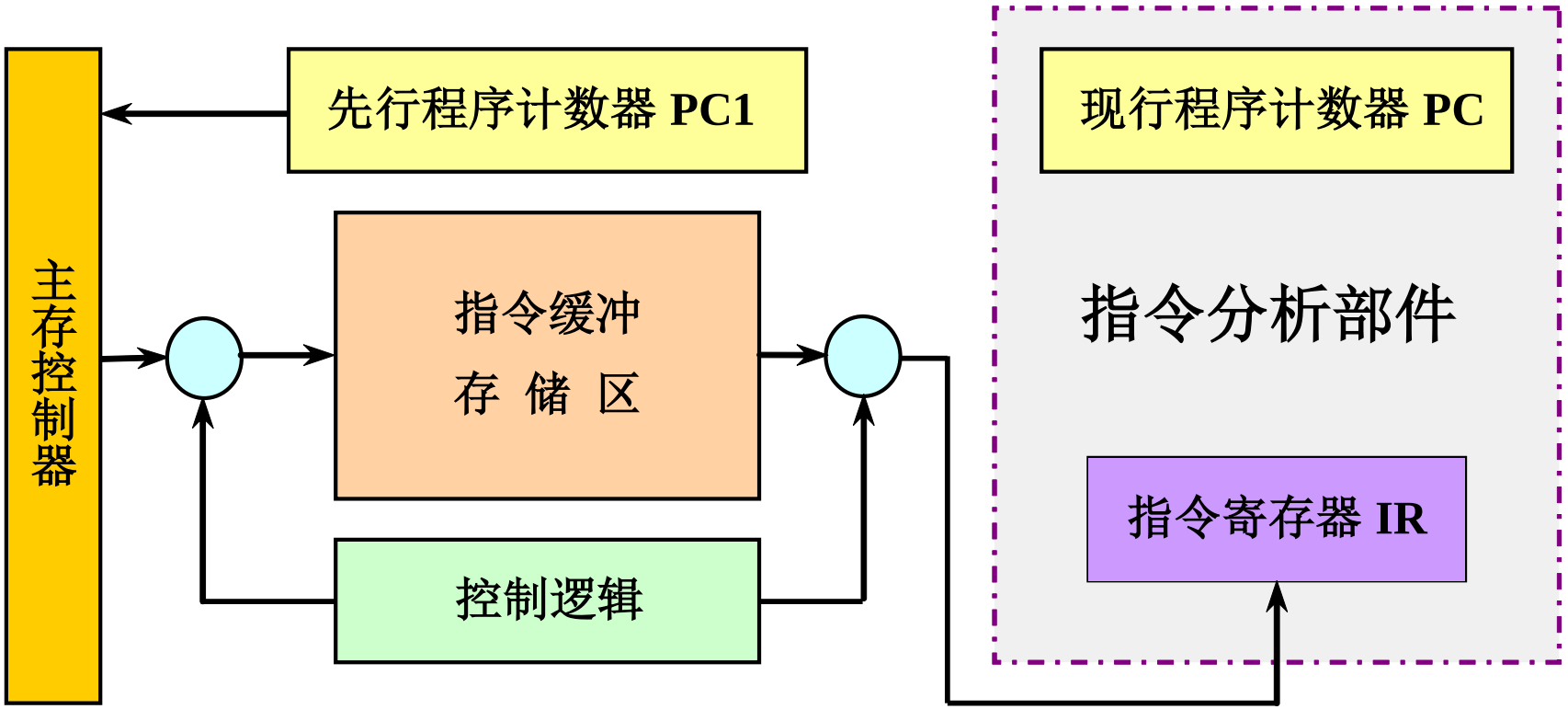
重叠执行对计算机组成的要求

- 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，采用多体交叉存储器，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储体。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

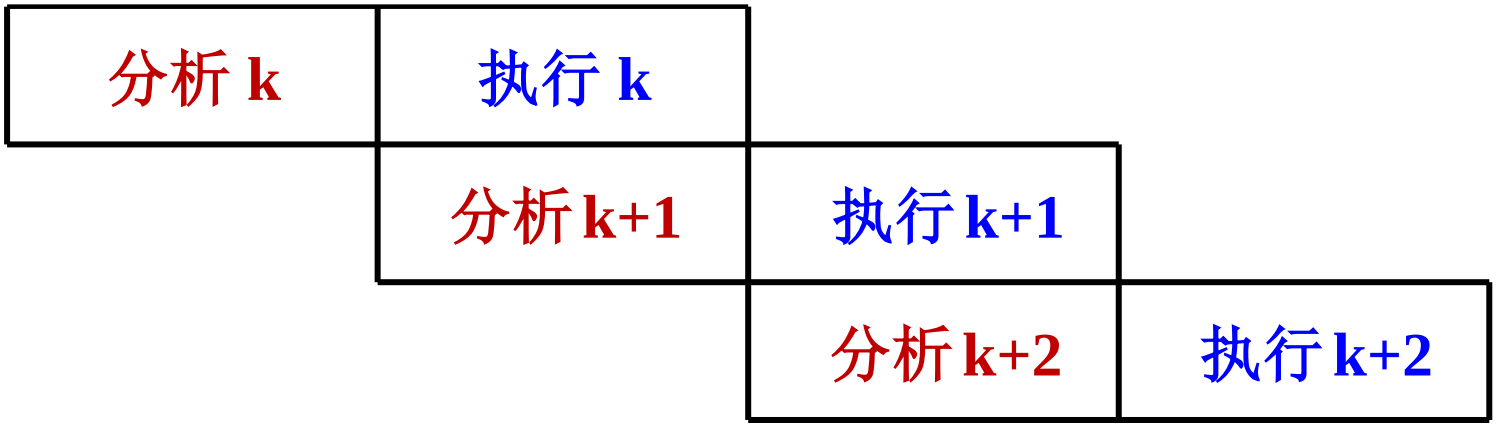
先行指令缓冲站

- **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：

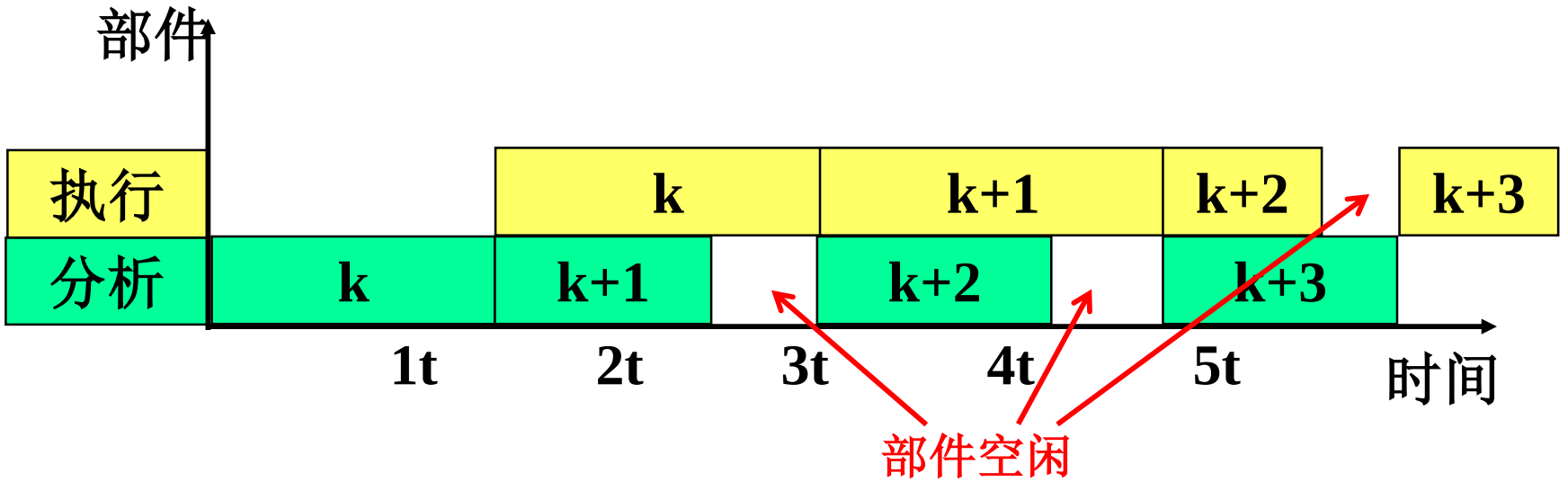
$$T = (1+n)\,t$$

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。

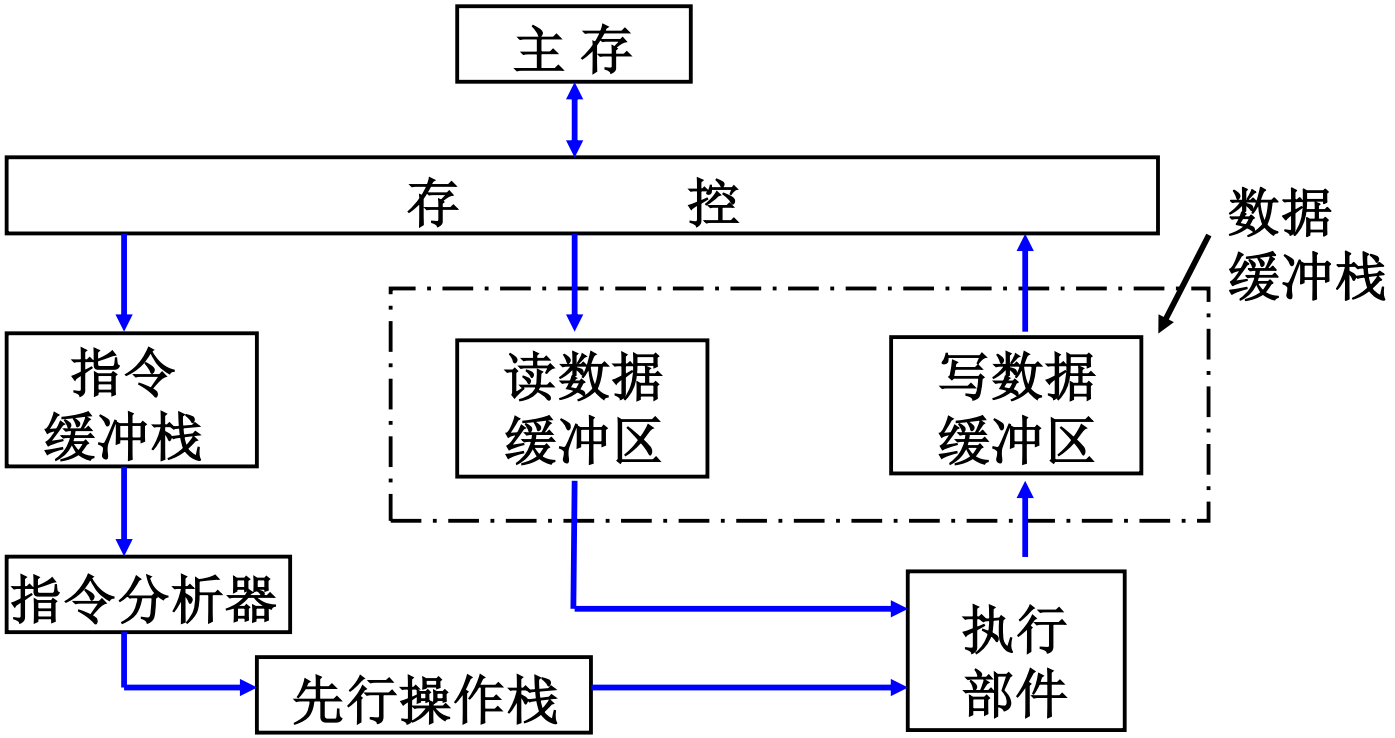


重叠执行对计算机组成的要求

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。

采取先行控制技术

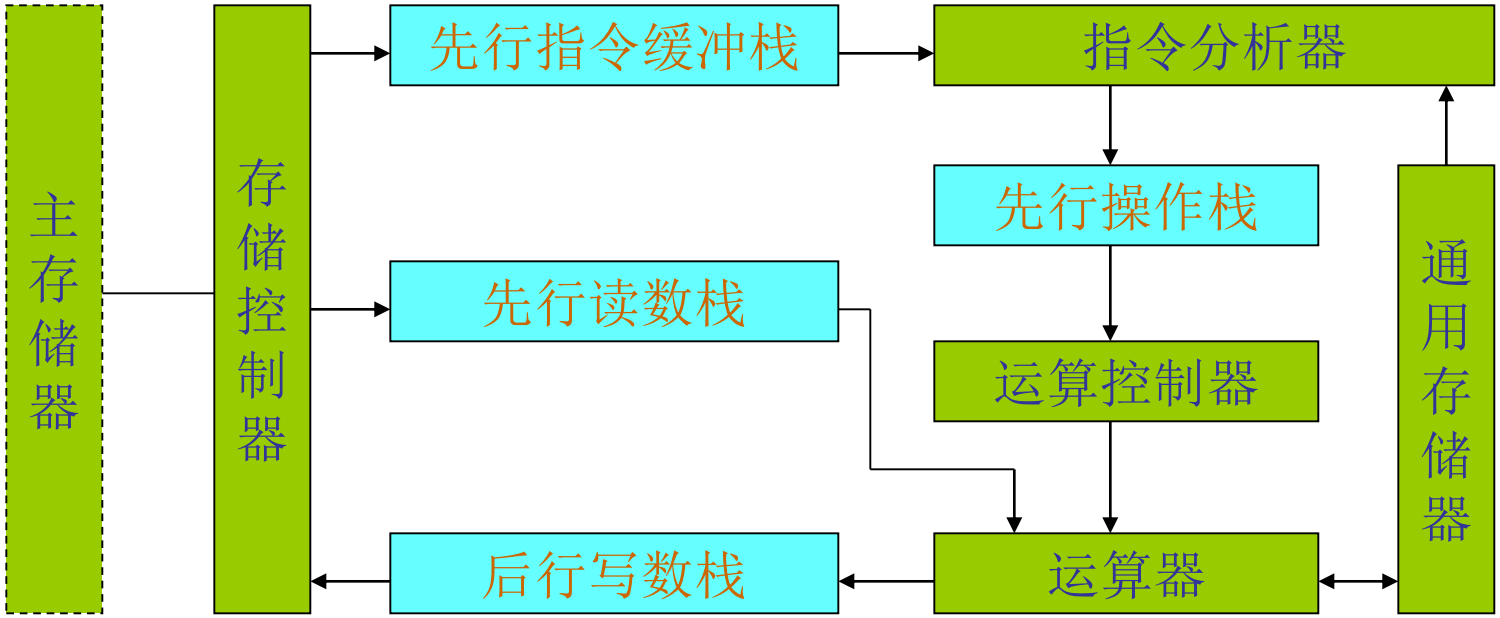


栈的深度（缓冲寄存器数量）要求：$D_{指缓} \geqslant D_{操作} \geqslant D_{读栈} \geqslant D_{写栈}$

采用先行控制技术系统示意图

先行控制技术

先行控制技术=缓冲技术+预处理技术

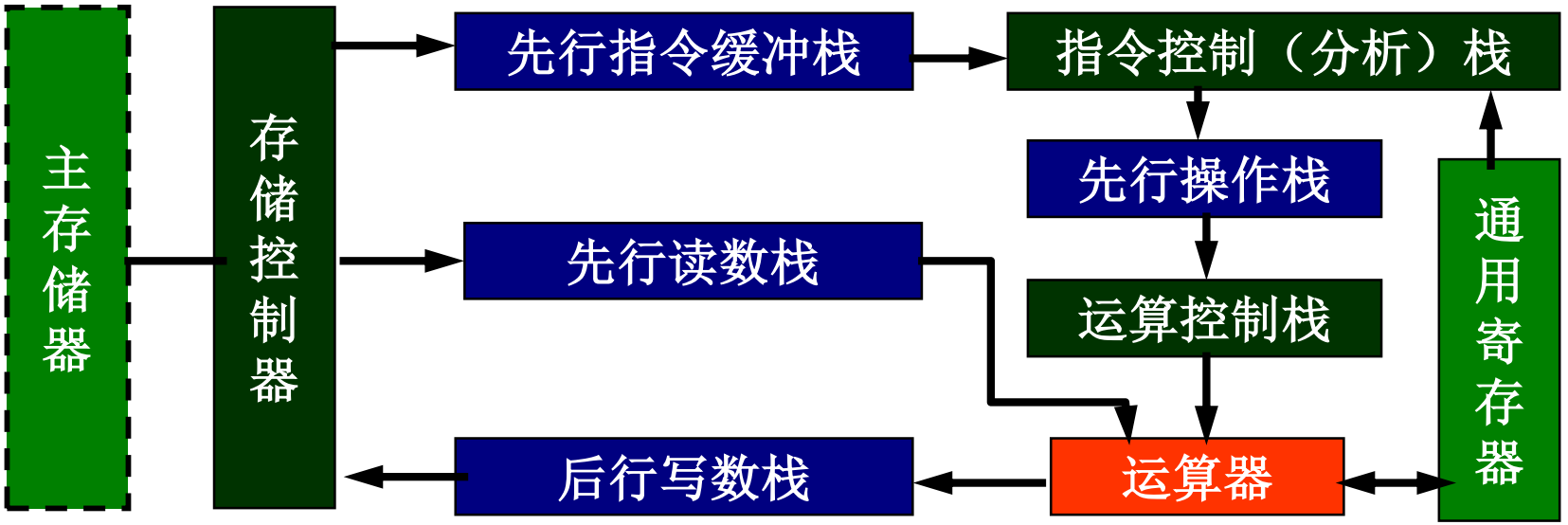


采用先行控制方式的处理机机构

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

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：$D_{指缓} \geqslant D_{操作} \geqslant D_{读栈} \geqslant D_{写栈}$

采用先行控制（Look-ahead，又称预测控制）技术的处理机

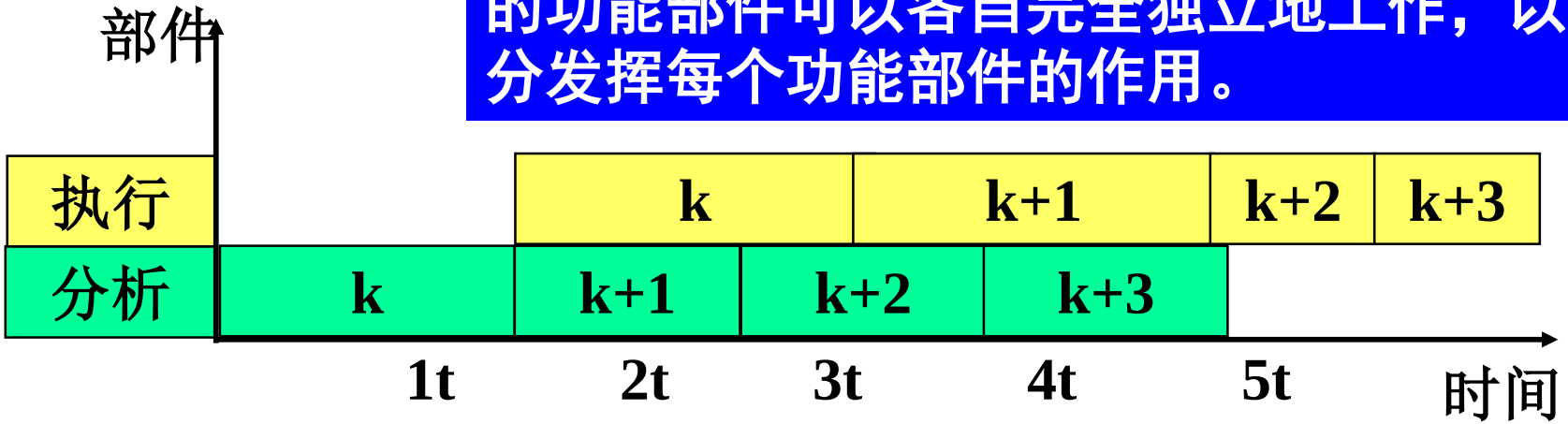
先行控制技术 = 缓冲技术 + 预处理技术

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

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

$$T_{先} = t_{分1} + \sum_{i=1}^{n} t_{执i}$$

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。

采取先行控制技术

先行控制技术的关键是缓冲技术和预处理技术。

缓冲技术是在工作速度不固定的两个功能部件之间设置缓冲栈，用以平滑它们的工作。

在采用了缓冲技术和预处理技术之后，运算器能够专心于数据的运算，从而大幅度提高程序的执行速度。

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 $t_{取指}$、 $t_{分析}$、 $t_{执行}$，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$_K$”与“取指$_{K+1}$”重叠

3)仅“执行$_K$”、“分析$_{K+1}$”、“取指$_{K+2}$”重叠

4)分别$t_{取指}=t_{分析}=2$，$t_{执行}=1$ 及 $t_{取指}=t_{分析}=5$，$t_{执行}=2$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

$100\times(t_{取指}+t_{分析}+t_{执行})$

2)仅“执行$_K$”与“取指$_{K+1}$”重叠

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

3)仅“执行$_K$”、“分析$_{K+1}$”、“取指$_{K+2}$”重叠

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

例题解答

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

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

2)仅“执行$_K$”与“取指$_{K+1}$”重叠

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

3)仅“执行$_K$”、“分析$_{K+1}$”、“取指$_{K+2}$”重叠

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

例题解答

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

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

2)仅“执行$_K$”与“取指$_{K+1}$”重叠

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

3)仅“执行$_K$”、“分析$_{K+1}$”、“取指$_{K+2}$”重叠

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

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

  “结构相关”或“资源相关”
  - 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）

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

5.1.2 相关处理

- **相关（Correlation）：**

　　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：**
  - 数据相关
  - 指令相关

5.1.2 相关处理

- **数据相关：**第k条指令与第$k+1$条指令的数据地址（例如操作数、变址偏移量等）之间出现关联。例如：第k+1条指令的源操作数地址正好是第k条指令存放运算结果的地址。在顺序解释执行时，不会出错，但在重叠方式下，就不对。
  - **主存数相关：**发生在主存空间
  - **寄存器组数相关：**发生在寄存器空间

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！

由于在“执行k”的末尾才形成第k+1条指令，按照一次重叠的时间关系，“分析k+1”所分析的是早已取进指缓的第k+1条指令的旧内容，这就会出错。为了避免出错，第k、k+1条指令就不能同时解释，我们称此时这两条指令之间发生了“指令相关”。 特别是当指令缓冲器可缓冲存放n条指令情况下，执行到第k条指令时，与已预取进指缓的第k+1到第k+n条指令都有可能发生指令相关。指缓容量越大，或者说指令预处理能力愈强的机器发生指令相关的概率就愈高。

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

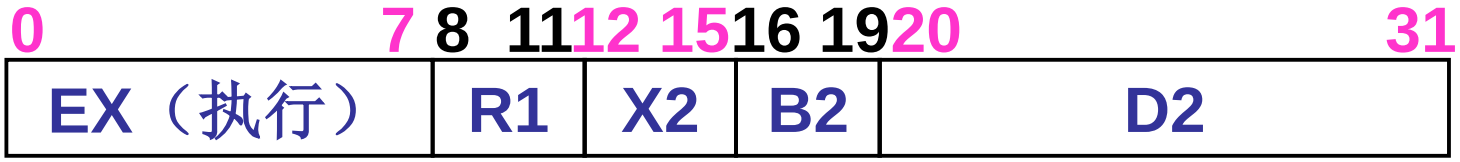
指令相关与处理

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：



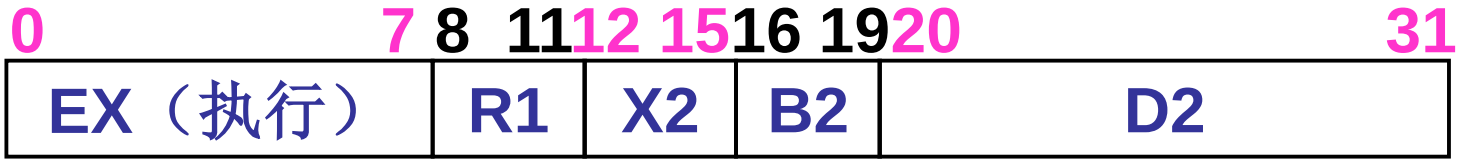
“执行”指令本身并不直接执行，实际指令的指令地址由第二个操作数地址指定：

((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。

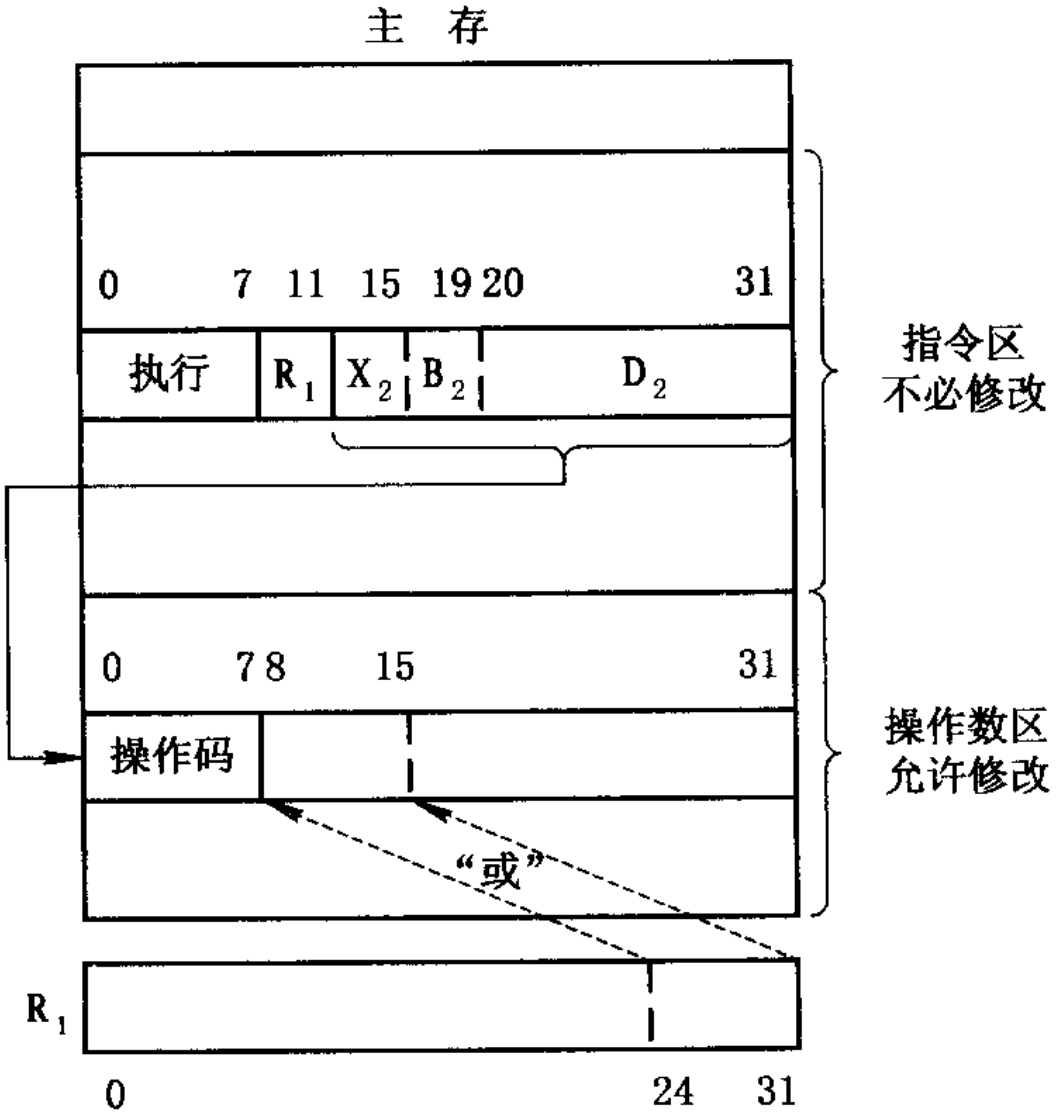
指令相关与处理

- IBM 370 “执行”指令形式：



- 当执行到“执行”指令时，按第二操作数$(X_2)+(B_2)+D_2$地址取出操作数区中单元的内容作为指令来执行。被修改的指令是以“执行”指令的操作数形式出现，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
0 7 11 15 19 20 31
执行
R_1
X_2
B_2
D_2
指令区
不必修改
0 7 8 15 31
操作码
操作数区
允许修改
“或”
R_1
0 24 31

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$_{k+1}$”读出的操作数不是前一条指令“执行$_k$”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

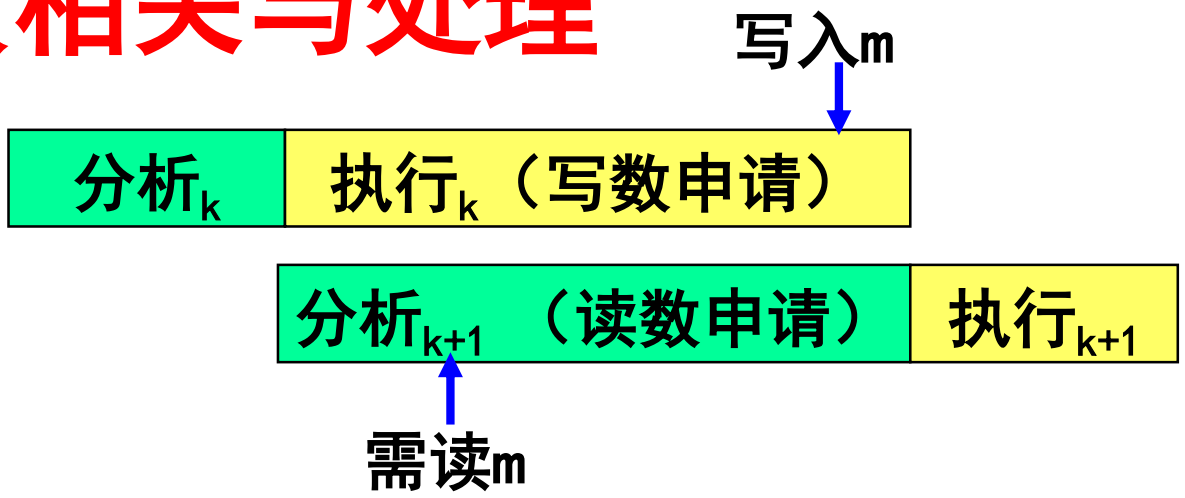
对于A1主存单元来说，存在“先写后读”相关。

主存空间数相关与处理

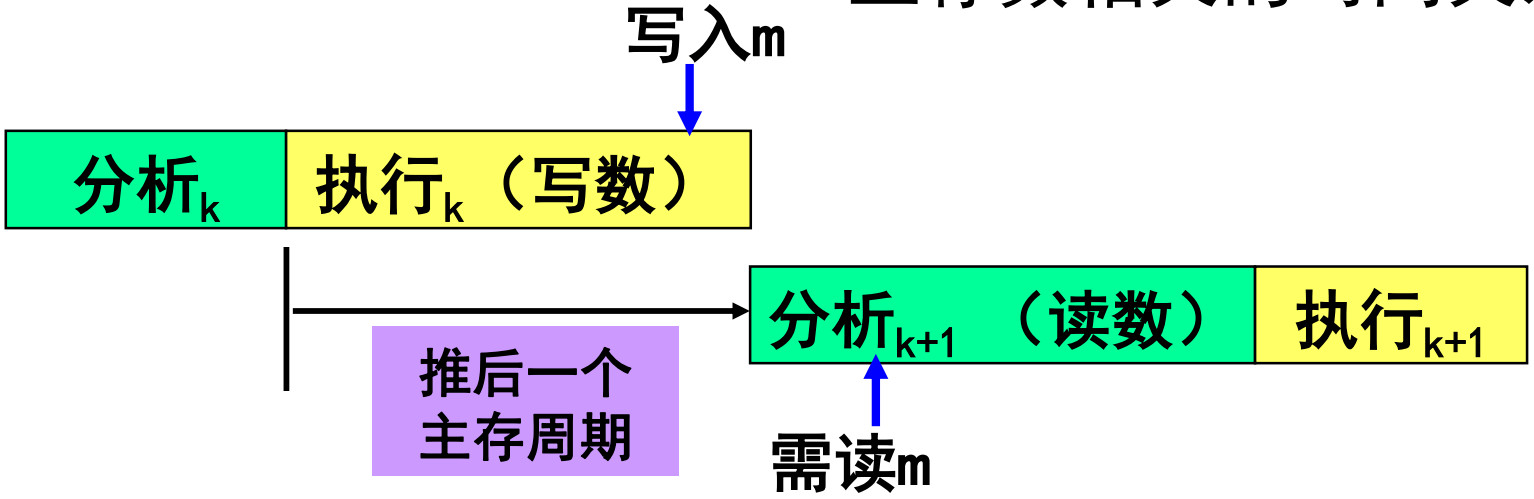
解决方法：

- 推后“分析$_{k+1}$”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



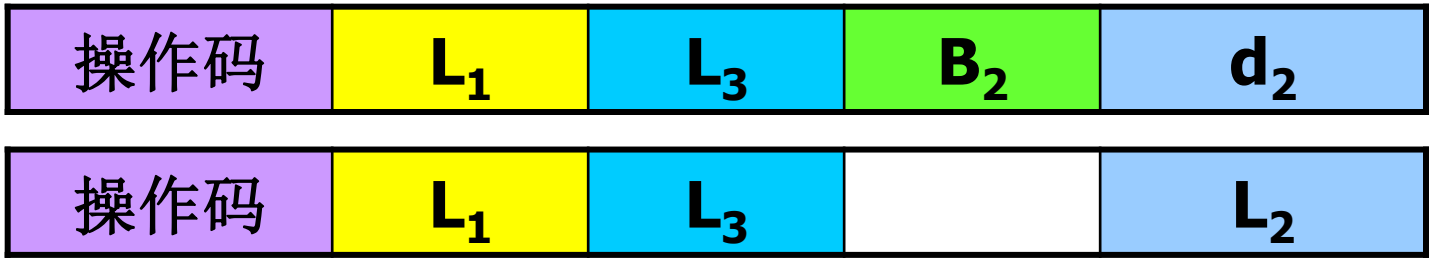
主存数相关的时间关系



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

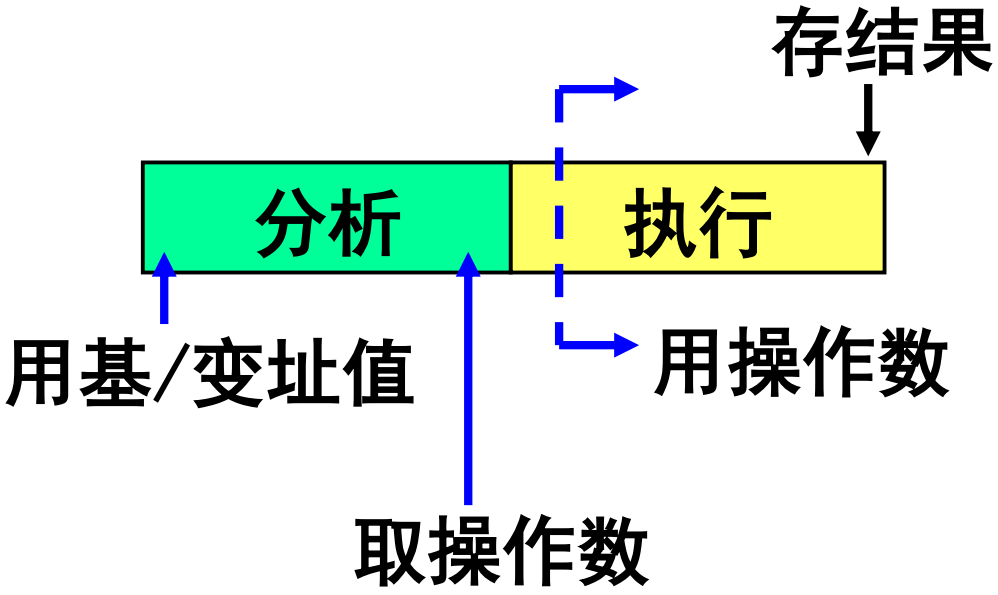
- 通用寄存器可以存放操作数、运算结果，也可以存放操作数物理地址的基址或变址值。
- 假设指令：



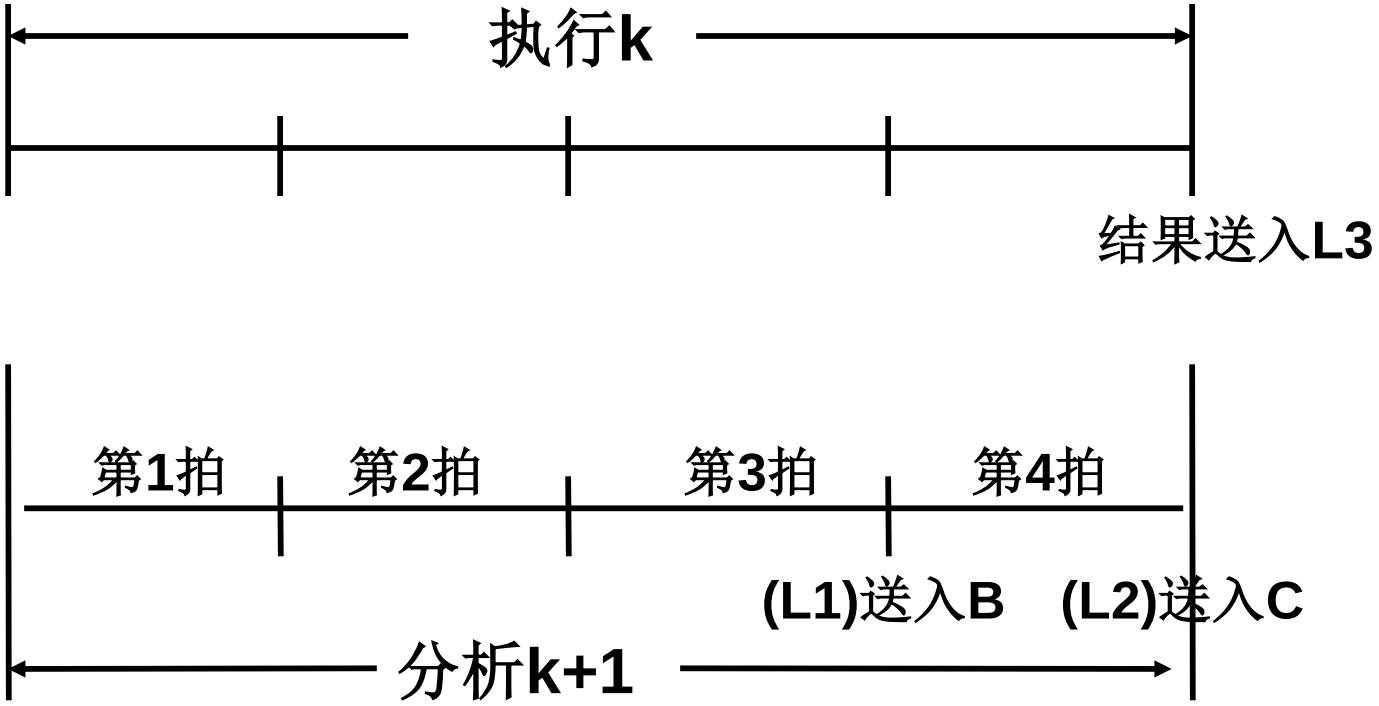
  - L1，L2，L3分别指明存放第1操作数、第2操作数和结果数的通用寄存器号；
  - B2为形成第二操作数地址的基址值所在的通用寄存器号；
  - d2为相对位移量。

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。**
- **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。**

在寄存器-寄存器型（RR型）指令和寄存器-存储器型（RS型）指令的执行过程中可能发生通用寄存器数据相关。看下面两条指令：

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

如果发生：

R1（k）＝R1（k＋1）

称为R1数据相关。如果发生：　R1（k）＝R2（k＋1）

称为R2数据相关。

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)
```

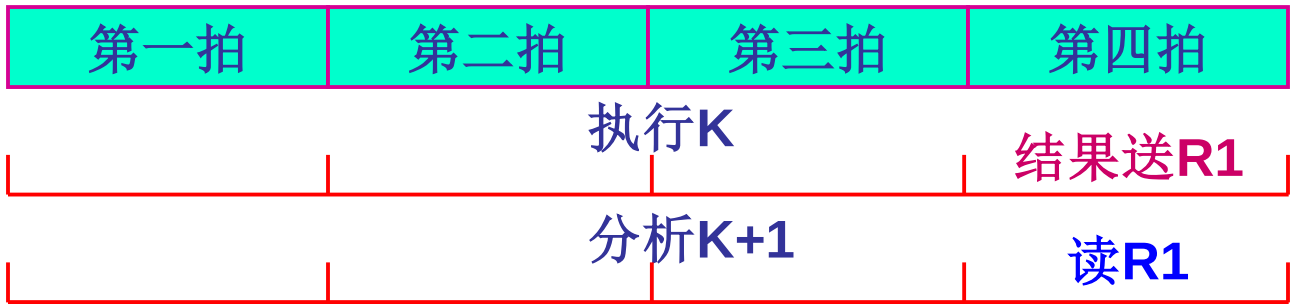
对于R1通用寄存器来说，存在“先写后读”相关。

解决方法：

- ①推后“分析$_{k+1}$”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$_{k+1}$”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



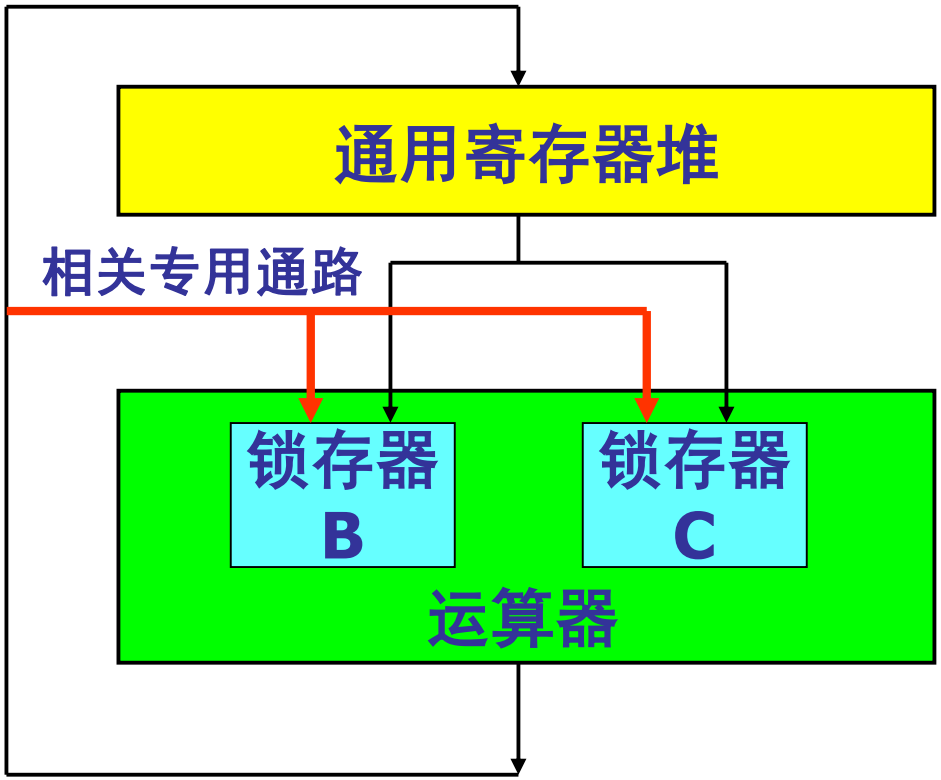
推迟一个节拍情况：

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

设操作数的有效地址

$$(X_d) + (B_2) \cdot (B_2 \neq 0000) + d_2$$

由分析器内的地址加法器形成。由于通常情况下，“分析”周期等于主存周期，所以，从时间关系上要求在“分析”周期的前半段，就能由通用寄存器输出总线取得(B2)，送入地址加法器。由于运算结果是在“执行”周期的末尾才送入通用寄存器组的，它当然不能立即出现在通用寄存器输出总线上。

通用寄存器组基址值/变址值相关处理

也就是说，在“执行k”得到的、送入通用寄存器的运算结果来不及作为“分析k+2”的基址值用，更不用说作为“分析k+1”的基址值用。因此，虽然是一次重叠，但基址值相关(B相关)就不止会出现一次相关，还会出现二次相关。即当出现B(k+1)=L3(k)时，称为发生了B一次相关；而当出现B(k+2)=L3(k)时，称为发生了B二次相关，如图所示：

通用寄存器组基址值/变址值相关处理

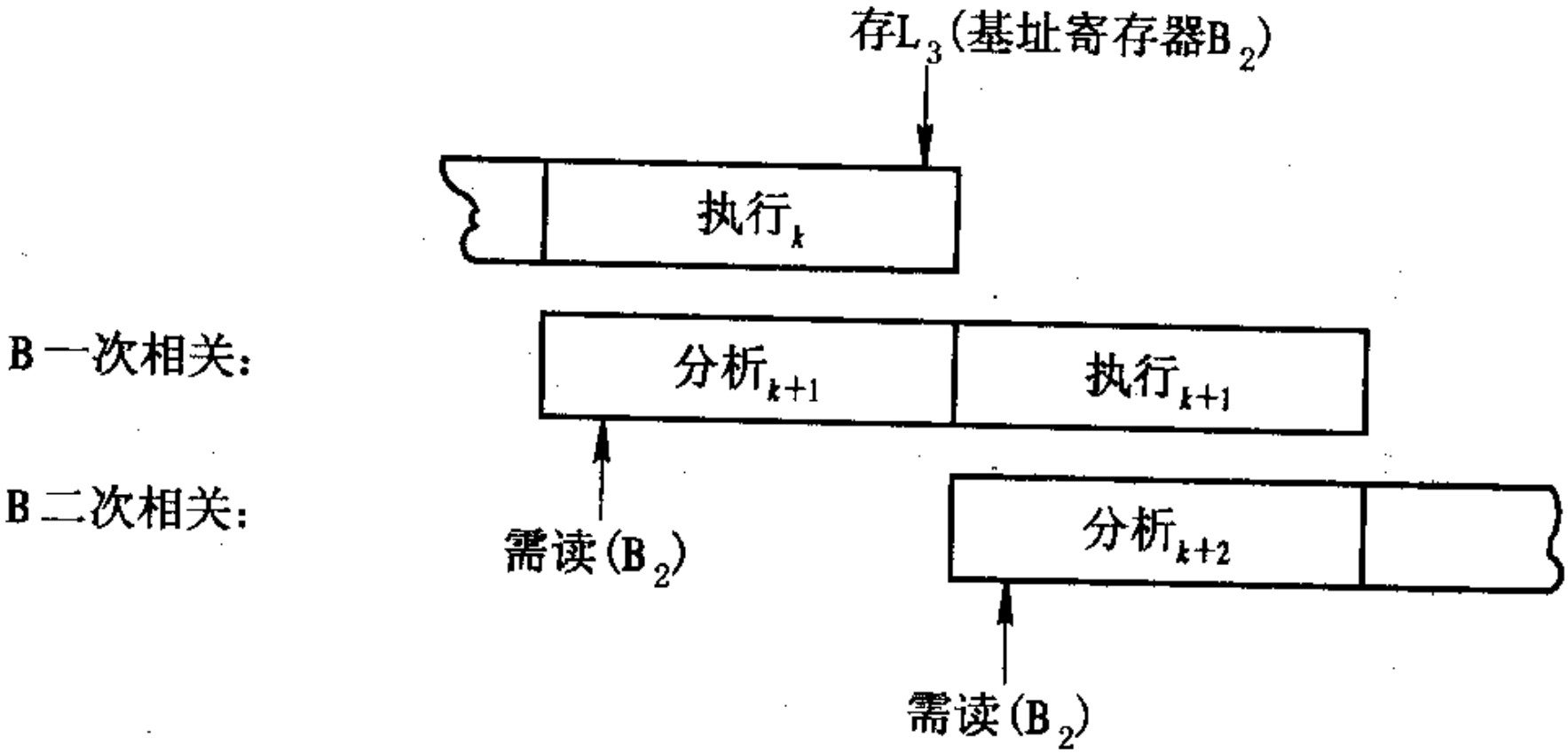
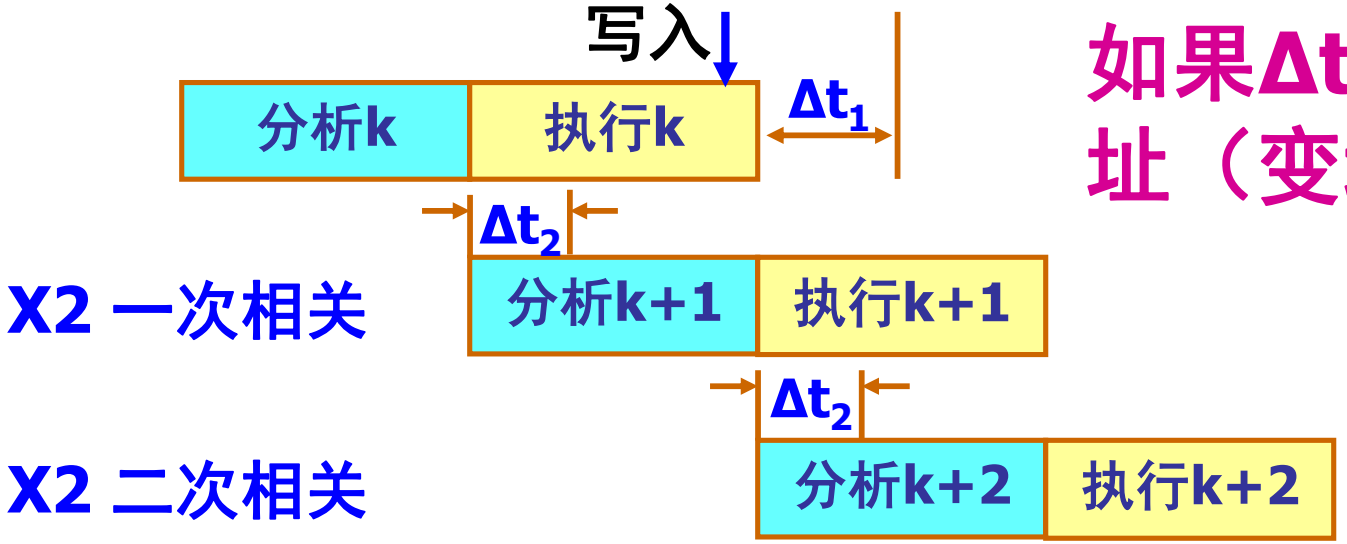


图　B一次相关与二次相关

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

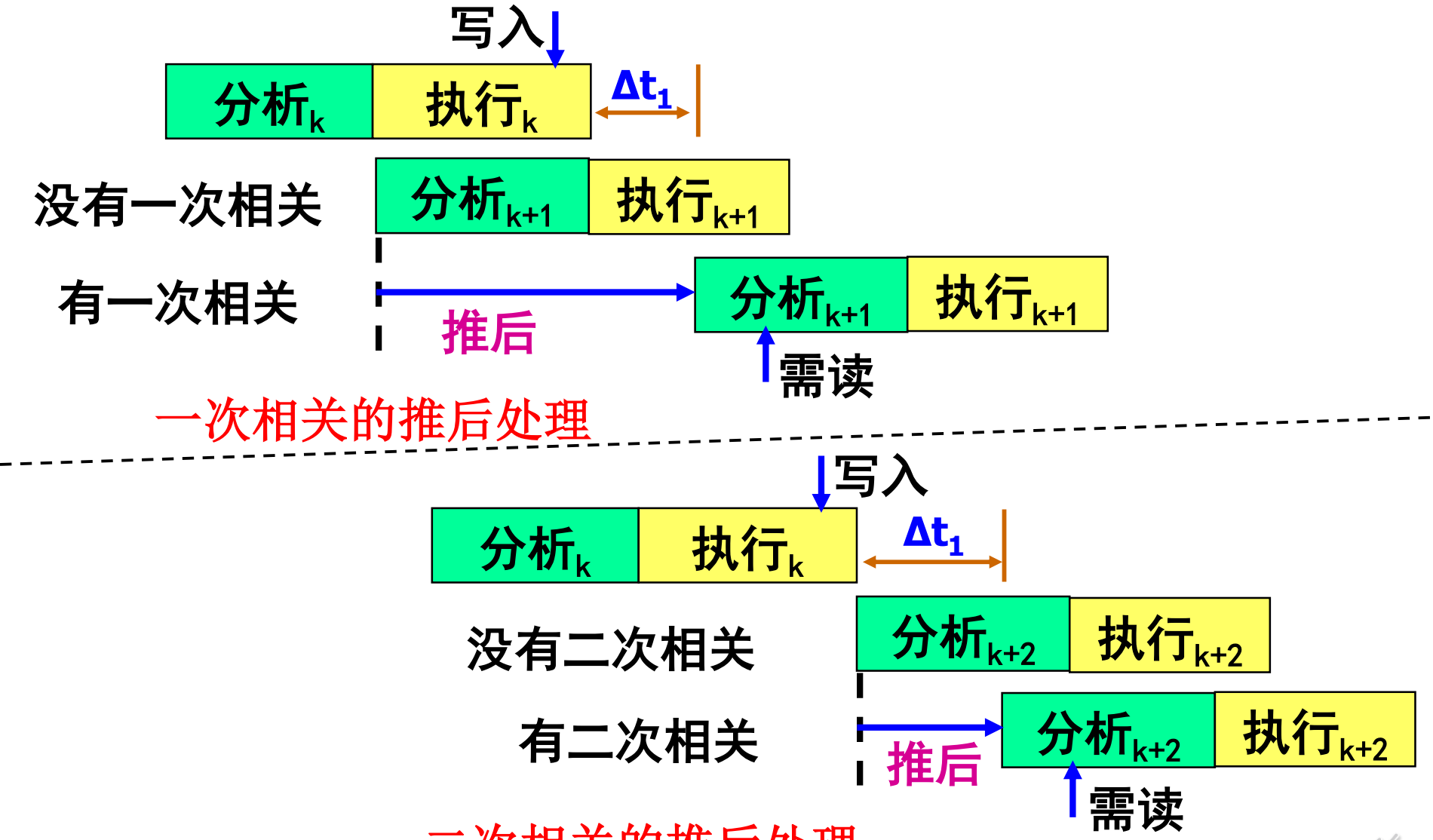
解决方法：

- ①推后“分析$_{k+1}$”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$_{k+1}$”的读（推后读）
  - 对于一次相关，除推后“分析$_{k+1}$”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$_{k+2}$”的起点，使“执行$_k$”送入通用寄存器的结果，在“分析$_{k+2}$”开始时已经出现在通用寄存器的输出总线上即可。

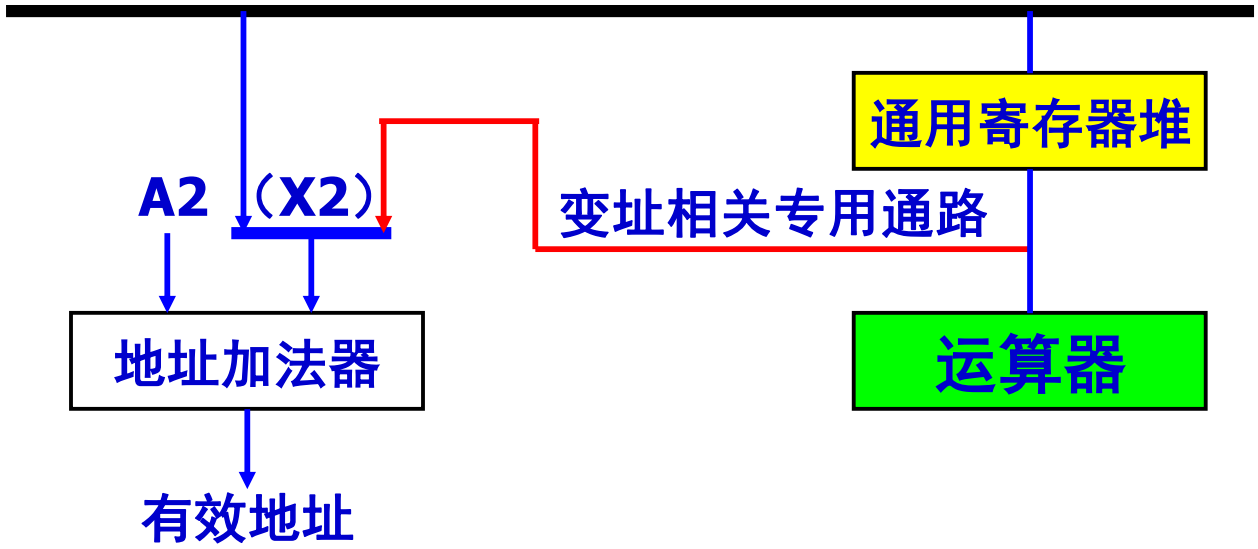
通用寄存器组基址值/变址值相关处理



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$_k$”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$_{k+1}$”推后到接着“执行$_k$”进行就可以了；
- 对于二次相关，不必推后“分析$_{k+2}$”。

控制相关

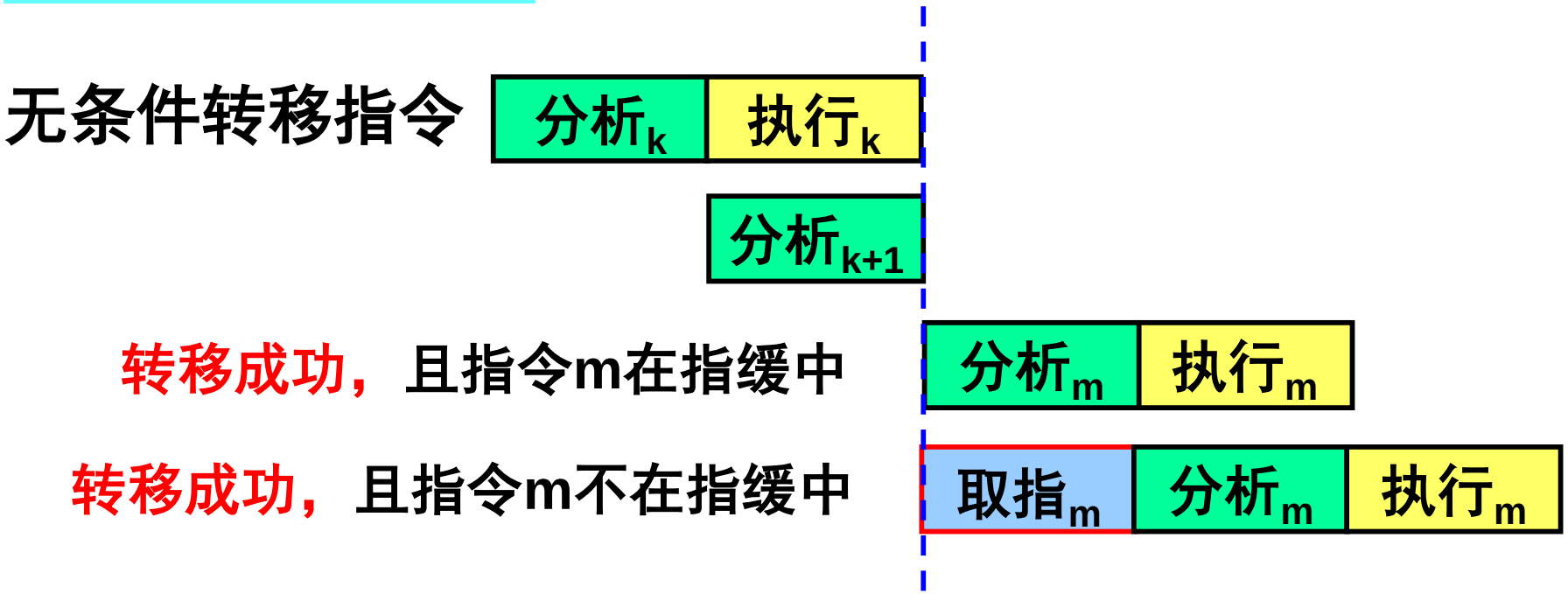
- 无条件转移
- 一般条件转移
- 复合条件转移
- 子程序调用
- 中断 等

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:      ...
```

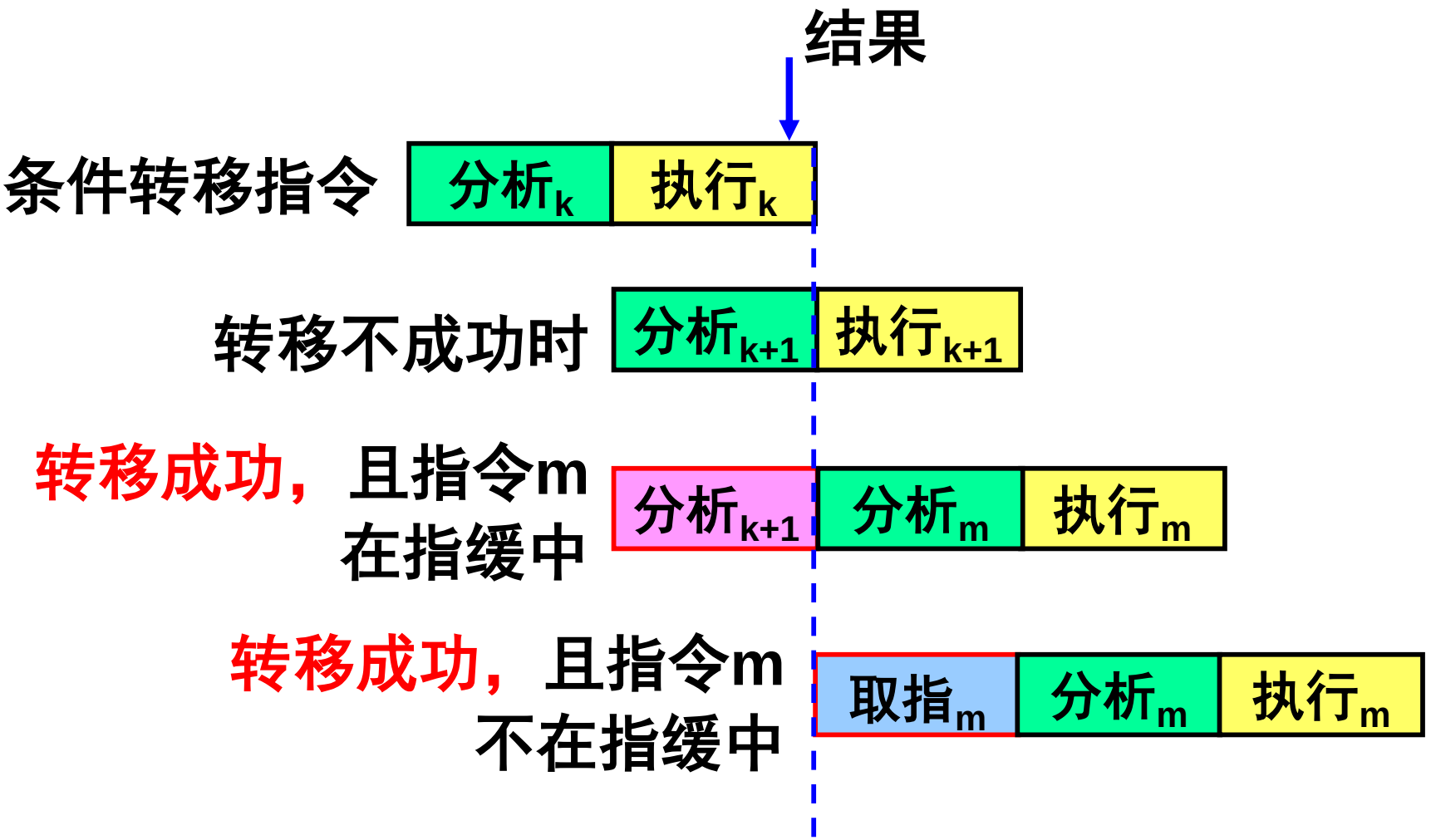
；M → (PC)，　M → (PC1)



指令相关处理：条件转移

- 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。

指令相关处理：条件转移

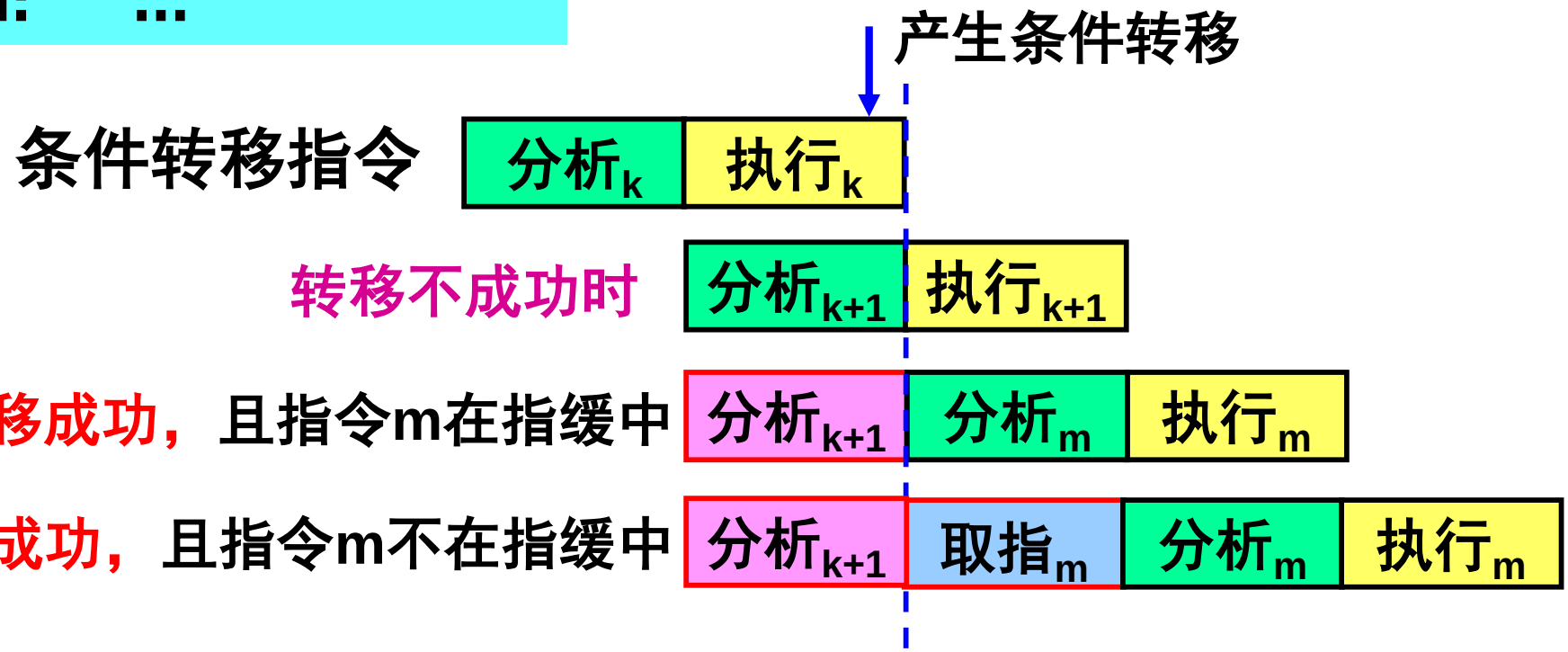

结果
条件转移指令
分析k
执行k
转移不成功时
分析k+1
执行k+1
转移成功，且指令m
在指缓中
分析k+1
分析m
执行m
转移成功，且指令m
不在指缓中
取指m
分析m
执行m

一般条件转移

```
K:      ...
K+1: JMP（CC）    M
...:    ...
M:      ...
```

；置条件码CC

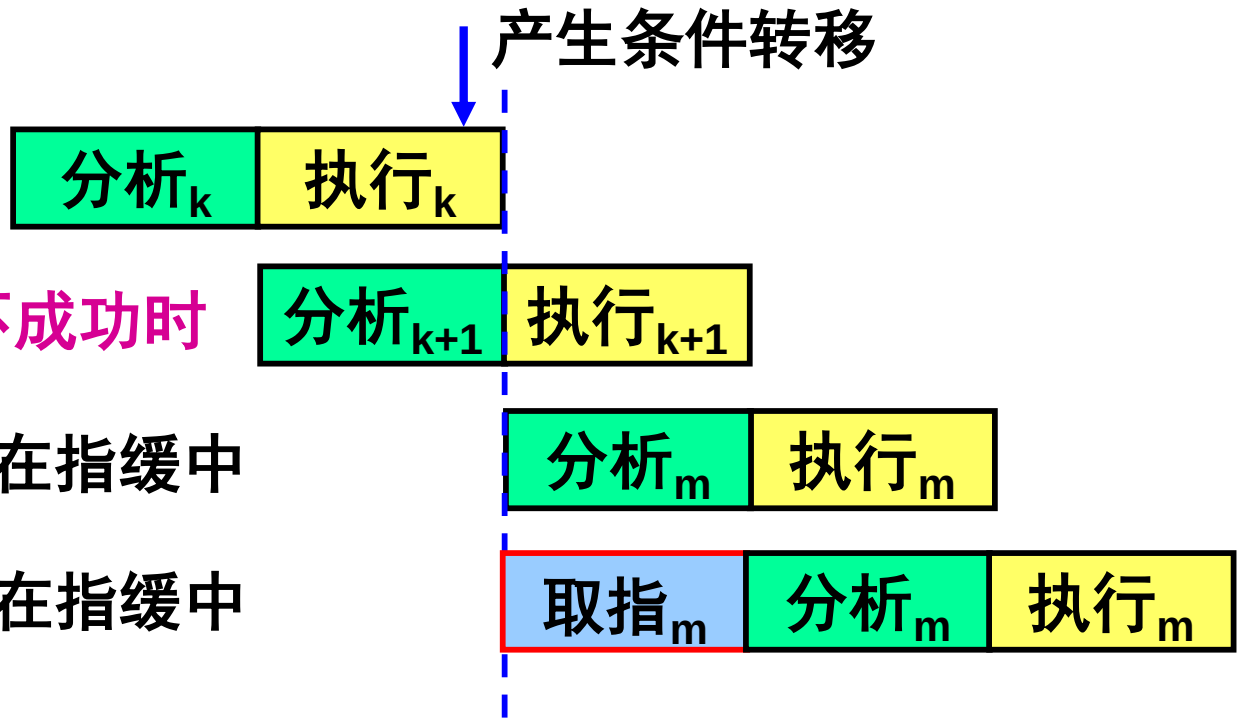
；如果CC为TRUE，转向M；否则继续



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



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。