《计组II》

第六章——流水线(2)

李瑞　副教授

蒋志平　讲师

计算机科学与技术学院

温故
&
知新

温故 —关于流水线(1)

- 什么是流水线?
- 为什么用流水线?
- 时空图
- 流水线的分类：
- 单功能 v.s 多功能
- 静态 v.s 动态
- 级别：部件级 v.s 指令级 v.s 宏
- 线性 v.s 非线性
- 顺序 v.s 乱序

温故 —关于流水线(2)

- 流水线的性能指标与计算
  - 总执行时间T，非流水 v.s 流水
  - 最大吞吐率(TP_{max})
  - 吞吐率(TP)
  - 加速比S
  - 效率E
- 流水线的瓶颈在哪里?
- 如何消减瓶颈?
- 流水线的特点

温故

&

知新

- 请先思考几个问题：

1. 流水线的深度如何权衡？ 过深， 过浅有何利弊?
2. 什么情况下，流水线能最大效率工作?
3. 什么情况下，流水线的效率最差?
4. 能否给出几种导致流水线效率降低的例子?
5. 如何改善流水线性能?

流水线性能优化策略　1. 增加指令流水线深度

- **Intel x86处理器的指令流水线：**
  - **8086：2级**
  - **80386：4级**
  - **80486：5级**
  - **Pentium：整数运算5级、浮点运算6级**
  - **Pentium Pro/II/III：至少10级**
  - **Pentium 4：**
    - **Willamette：20级**
    - **Northwood：20级**
    - **Prescott：31级**
  - **Core微构架的双核处理器：14级**

来自指令Cache

32字节通道

取指令

IFU1 指令流缓冲器（保存1行）

下一指令指针

16字节

IFU2 指令长度译码器

动态分支预测器

16字节

IFU3 译码器排列段

16字节

多于4个微操作的指令

MIS

ID1 译码器0（复杂）4 | 译码器1（简单）1 | 译码器2（简单）1

微指令定序器

指令译码

6×118位

ID2 已译码的指令队列

3×118位

静态分支预测

3×118位

寄存器重命名

RAT 寄存器分配器 RAT

IFU (Instruction Fetch Unit)
ID (Instruction Decode)
RAT (Register Allocator)
ROB (Reorder Buffer)
RS (Reorder Buffer)
DIS (Dispatcher)

EX (Execute Stage)
IEU (Integer Execution Unit)
FPU (Float point Unit)
JEU (Jump Execution Unit)
RU (Retire Unit)

图7.14 Intel PentiumII 的指令流水线结构

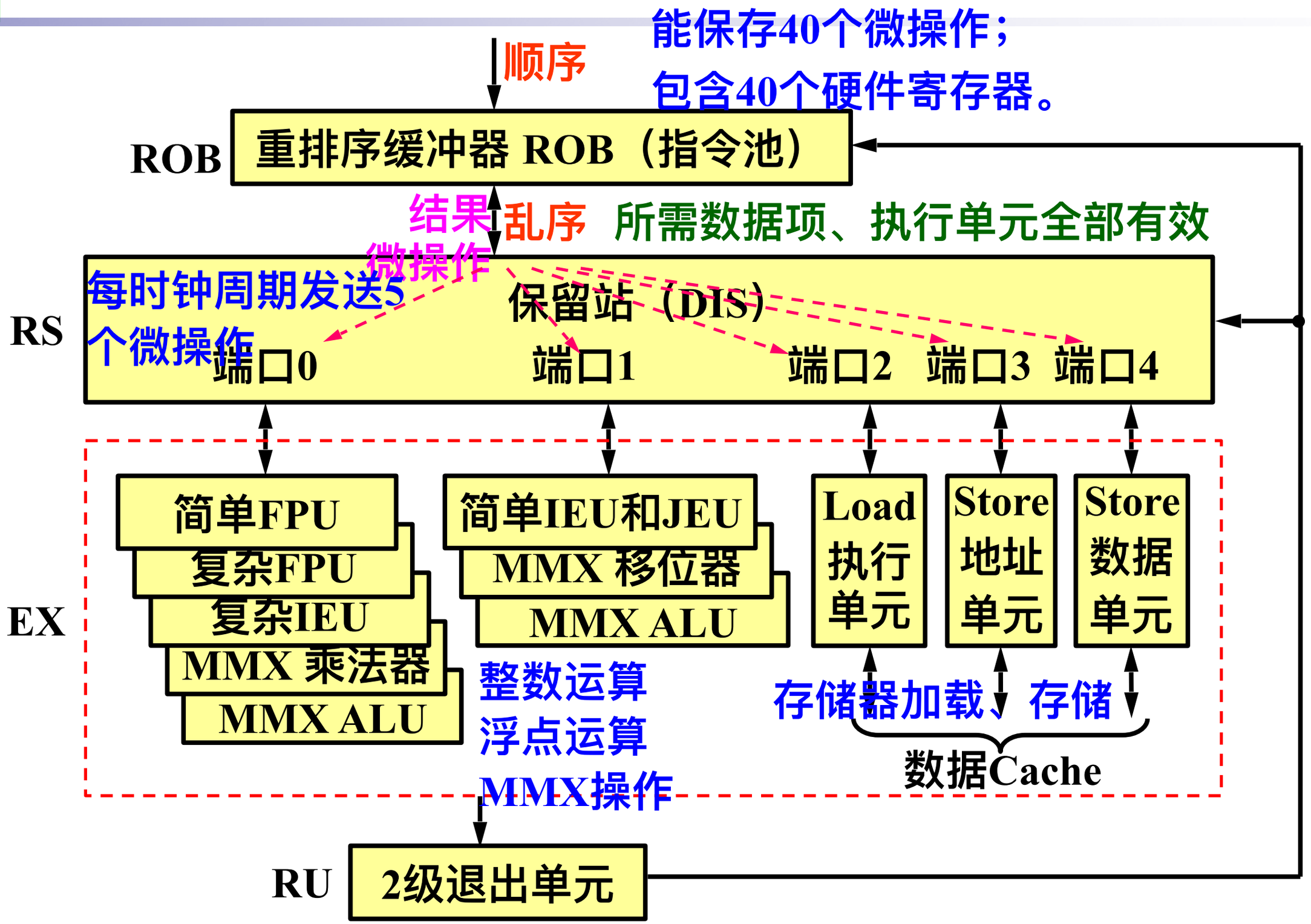


图7.14 Intel PentiumII 的指令流水线结构(续)

7.3.2 指令流水线策略　2. 增加指令流水线条数

- 增加指令流水线的深度的局限：
  - 指令执行过程的细化是有限度的
  - 随着流水线深度的增加，流水线段之间的缓冲器增多，延迟加大，使流水线的性能提高受到阻碍
- 多指令流水线结构
  - Intel Pentium 处理器：
    - U指令流水线（主流水线）
    - V指令流水线（副流水线）
  - IBM PowerPC 601
  - 多核CPU
  - AMD Ryzen CPU每个核心都有10条流水线

7.3.2 指令流水线策略　2. 增加指令流水线条数

- 多指令流水线结构
  - Intel Pentium 处理器：
    - U指令流水线（主流水线）
    - V指令流水线（副流水线）

```
MOV  AX,5
INC  BX
ADD  AX,BX
XOR  CX,CX
MOV  DX,8
INC  DX
```

U、V指令流水线并行执行

U、V指令流水线并行执行

不能同时送到U、V指令流水线中去执行

第六代微处理器P6：寄存器重命名映射技术

6.3 流水线中的相关

6.3.1 概述

1. 什么是相关?

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，后一条指令不能在原先指定的时钟周期开始执行。

- 消除相关的基本方法——暂停

暂停流水线中某条指令及其后面所有指令的执行，该指令之前的所有指令继续执行。

2. 相关的类型

- **结构相关**：当指令在重叠执行过程中，硬件资源满足不了指令重叠执行的要求，发生资源冲突时将产生“**结构相关**”。
- **数据相关**：因一条指令需要用到前面指令的结果，而无法与产生结果的指令重叠执行时，就发生了“**数据相关**”。
- **控制相关**：当流水线遇到分支指令和其它会改变PC值的指令时就发生“**控制相关**”。

6.3.2 结构相关

当指令在重叠执行过程中，硬件资源满足不了指令重叠执行的要求，发生资源冲突时将产生“结构相关”

➢ 在流水线机器中，为了使各种指令组合能顺利地重叠执行，需要把功能部件流水化，并把资源重复设置。

- **导致结构相关的常见原因：**
  - **功能部件不是全流水**
  - **重复设置的资源数量不足**

- 实例：当数据和指令存在同一存储器中时，访存指令会引起存储器访问冲突。

	时钟周期								
指令编号	1	2	3	4	5	6	7	8	9
指令i	IF	ID	EX	MEM					
指令i+1		IF	ID	EX	MEM				
指令i+2			IF	ID	EX	MEM			
指令i+3				IF	ID	EX	MEM		
指令i+4					IF	ID	EX	MEM	

IF (Instruction Fetch Unit)
ID (Instruction Decode)
EEX (Instruction Decode)
MEM (Memory Operation)

解决方法：

- 插入暂停周期 （时空图）
- 设置相互独立的指令存储器和数据存储器
  或设置相互独立的指令Cache和数据Cache

引入暂停后的流水线时空图

	时钟周期								
指令编号	1	2	3	4	5	6	7	8	9
指令i	IF	ID	EX	MEM					
指令i+1		IF	ID	EX	MEM				
指令i+2			IF	ID	EX	MEM			
指令i+3				stall	IF	ID	EX	MEM	
指令i+4						IF	ID	EX	MEM

产生“流水线气泡”或“气泡”

- 避免结构相关的方法：
  - 所有功能单元完全流水化
  - 设置足够多的硬件资源

  但是，硬件代价很大！
- 有些设计方案允许结构相关存在
  - 降低成本
  - 减少功能单元的延迟

6.3.3 数据相关

当一条指令的结果还未有效生成，该结果就被作为后续指令的操作数时，就发生了“数据相关”

产生原因： 当指令在流水线中重叠执行时，流水线有可能改变指令读/写操作数的顺序，使之不同于它们在非流水实现时的顺序，这将导致数据相关。

数据相关的分类：

两条指令 *i* 和 *j*，都会访问同一寄存器*R*，假设*i* 先进入流水线，则它们对*R*有四种不同的访问顺序：

(1) 写后读(*RAW*) —— *i* 写*j* 读

- 如果*j* 在*i* 完成写之前从*R* 中读出数据，将得到错误的结果！

➢ 最常见的数据相关，严重制约了CPU的性能，是程序最重要的特征之一！

(2) 写后写(*WAW*) —— *i* 写*j* 写

- 如果*j* 在*i* 之前完成写操作，*R*中将保存错误的结果！

➢ 当流水线中有多个段可以写回，而且当流水线暂停某条指令的执行时，其后的指令可以继续前进时，可能引起这种类型的相关。

(3) 读后写(WAR) —— i 读j 写

➢ 如果j 先将数据写入R，i 将读出错误的结果！

➢ 这种相关很少发生。当有些指令在流水段后半部分读源操作数，另一些指令在流水线前半部分写结果，可能引起这种类型的相关。

(4) 读后读(RAR) —— i 读j 读

- 不引起数据相关！

【例】4级指令流水线，各级分别为取指IF、读数RD、执行EX和写结果WB。简单指令（第1、2、4、6条指令）在EX1段执行，需1个时钟周期；复杂指令（第3、5条指令）在EX2段执行，需3个时钟周期。

指令	1	2	3	4	5	6	7	8	9	10	相关类型
1. R1+R2→R0	IF	RD	EX 1	WB							RAW
2. R0-R3→R4		IF	RD	EX 1	WB						
3. R0→10H(R5)			IF	RD	EX 2-1	EX 2-2	EX 2-3	WB			WAR
4. R6→R0				IF	RD	EX 1	WB				
5. R1 × R2→R0					IF	RD	EX 2 1	EX 2 2	EX 2 3	WB	WAW
6. R1+R2→R0						IF	RD	EX	WB		

相关造成指令2、3和6的结果错误。

解决方法：向流水线中插入暂停周期

- ✓ 采用**直通**（forwarding）**技术**

  控制逻辑将前面指令的结果从其产生的地方直接连通到当前指令所处的位置。

- ✓ 增加**专用硬件**

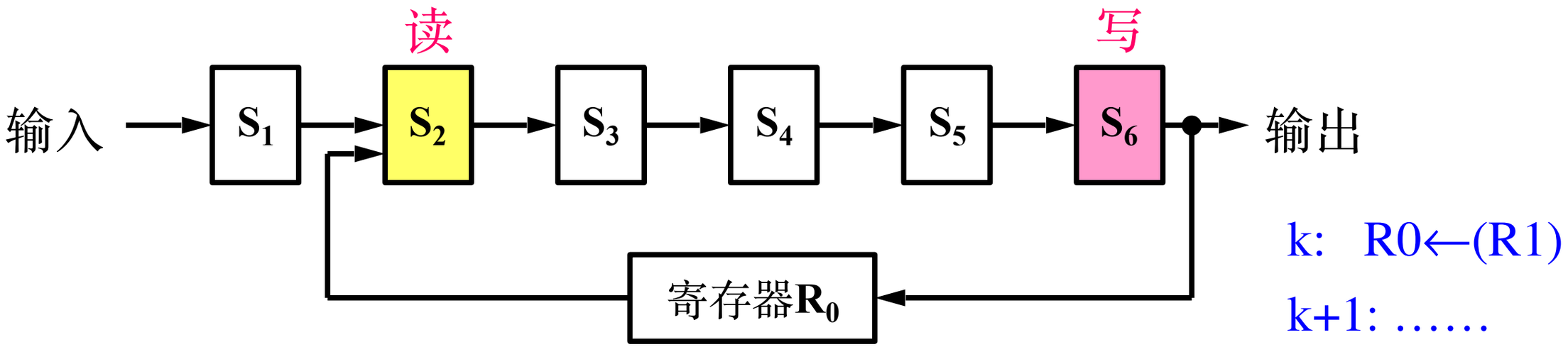
  增加流水线互锁（pipeline interlock）硬件。互锁硬件先要检测流水线中指令的数据相关性，当互锁硬件发现数据相关时，使流水线工作**停顿**下来，直到相关消失为止。

- ✓ 利用**编译器**

  流水线调度/指令调度：编译器可以对指令重新排序或插入空操作指令，使得加载任何冲突数据的操作被延迟，但对程序逻辑或输出不受影响。

解决办法：

- 采用直通（forwarding）技术（相关直接通路）



k: R0←(R1)

k+1: ……

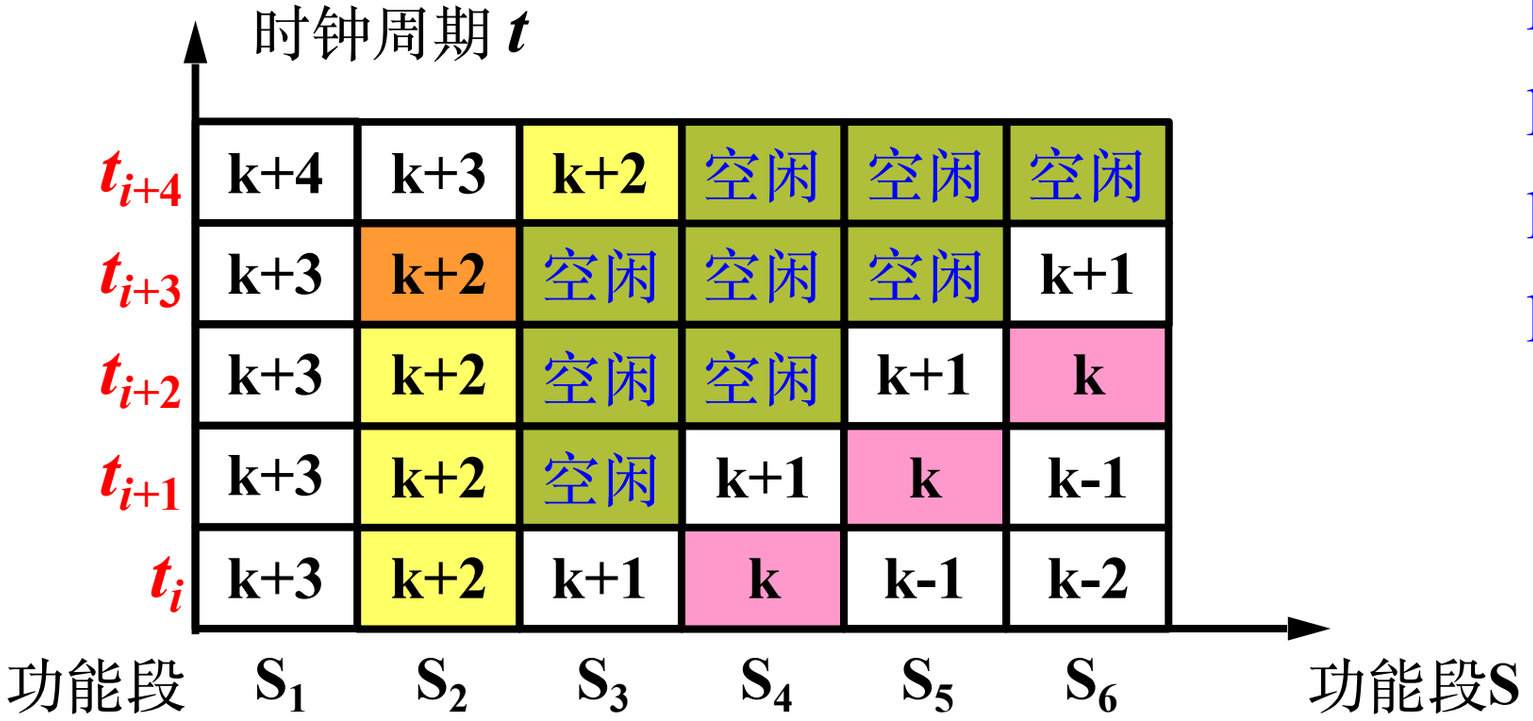
k+2: R2←(R0)+(R3)

k+3: ……

k+4: ……

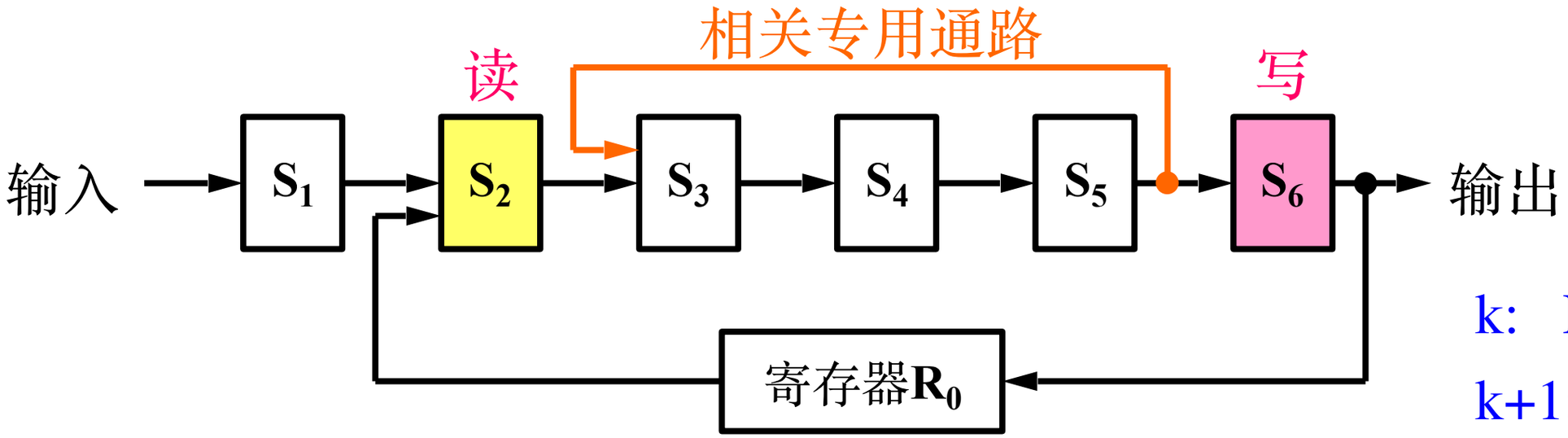
k+5: ……

……



解决办法：

- 采用直通（forwarding）技术（相关直接通路）



k: R0←(R1)

k+1: ……

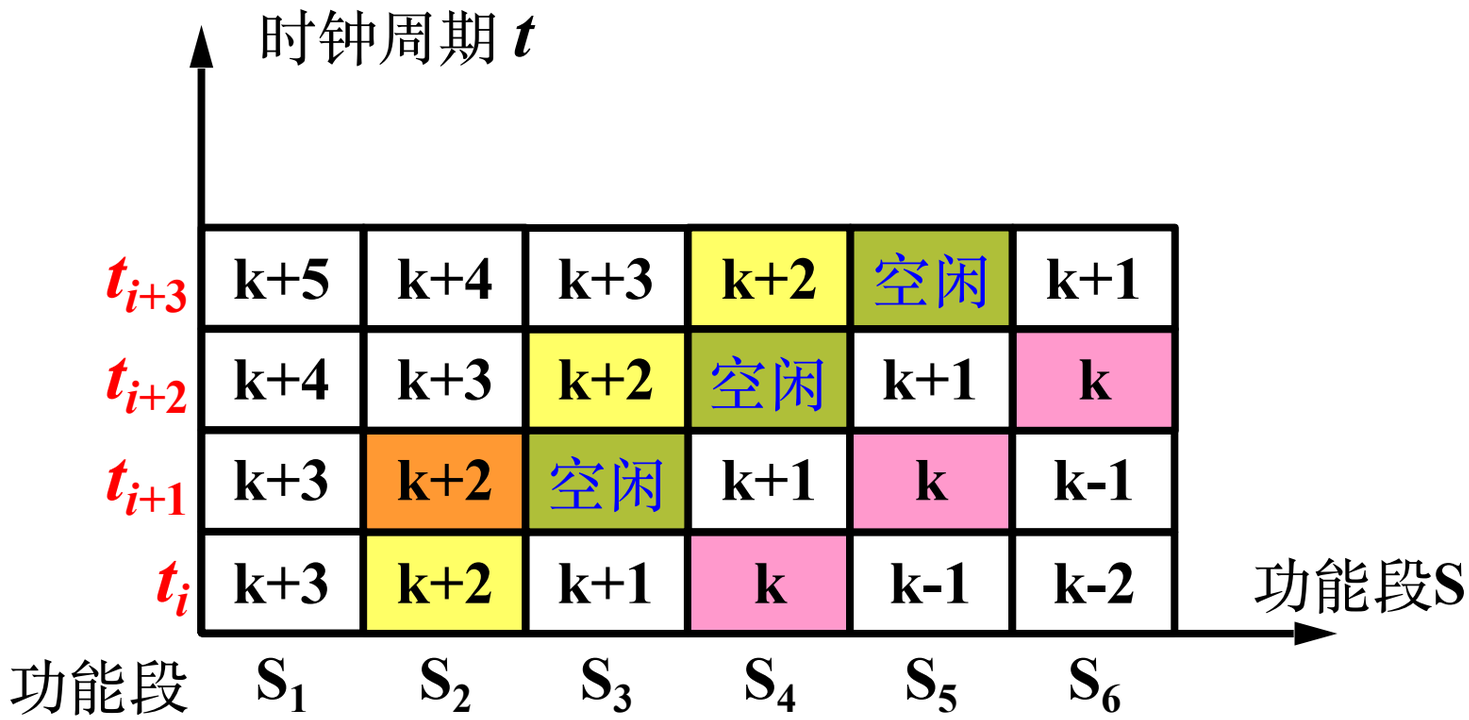
k+2: R2←(R0)+(R3)

k+3: ……

k+4: ……

k+5: ……

……



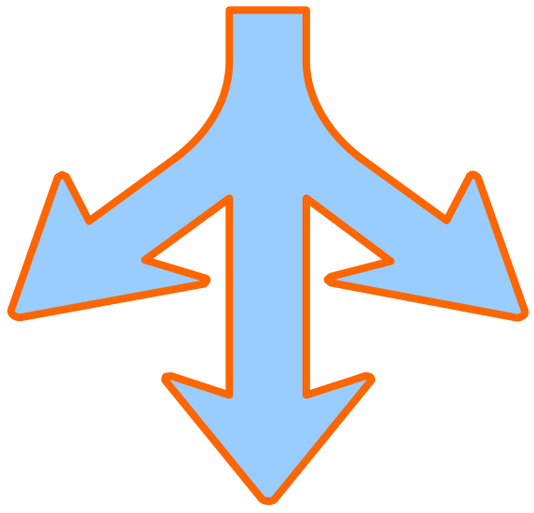
6.3.4 控制相关

当流水线遇到分支指令和其它会改变**PC**值的指令时就发生“控制相关”

- 使程序执行顺序发生改变的转移指令有两类：
  - 无条件转移指令（如无条件跳转、调用、返回指令等）
    - 某些CPU（如UltraSPARC III）：紧跟在无条件转移指令之后的指令必须执行。
    - 另一些CPU：采取相对复杂的方法，如提前计算出转移目标地址。
  - 条件分支转移指令（为零跳转、循环控制指令等）
    - 不仅需要延迟槽，而且一直到流水线的深处，取指单元才能知道到哪里去取下一条指令。
    - 条件分支指令对流水线性能的影响远比无条件转移指令要大。
- 分支延迟槽（branch delay slot）：程序中位于转移指令后面的存储单元位置。

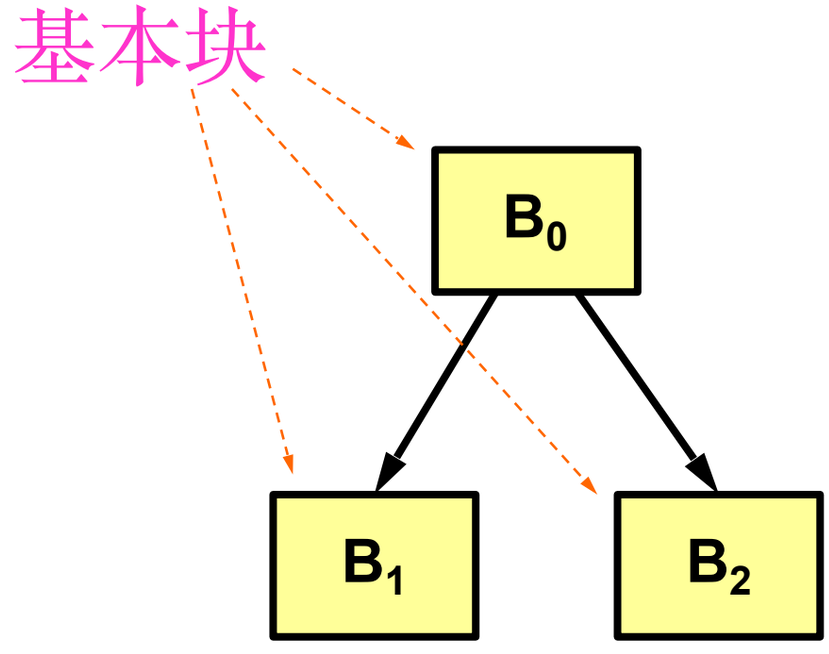
对条件分支指令的处理方法：

1. 冻结流水线
2. 预取分支目标
3. 多流
4. 循环缓冲器
5. 分支预测
   - ① 静态分支预测
     - 预测分支不会发生
     - 预测分支总是发生
     - 由编译器预测
     - 测试法
   - ② 动态分支预测
     - 分支历史表
     - 分支历史移位寄存器
6. 延迟分支



方法1：冻结（freeze）流水线

- 一旦在指令译码段检测到分支指令，就在转移目标地址确定之前保存或删除所有紧随分支指令之后的指令，当分支指令从执行段流出、确定出新的PC值时，流水线才继续依据新PC值填充流水线。
- 会严重地影响流水线的性能。
- 早期的CPU。



两路的条件分支

方法2：预取分支目标（prefetch branch target）

- 当条件分支指令被识别时，除了紧随其后的指令外，分支目标也被预取放在当前流水线，并保存到分支指令被执行。
- 如果分支跳转发生，已预取到的目标指令可立刻执行。
- IBM360/91采用这种方法。

方法3：多流（multiple streams）

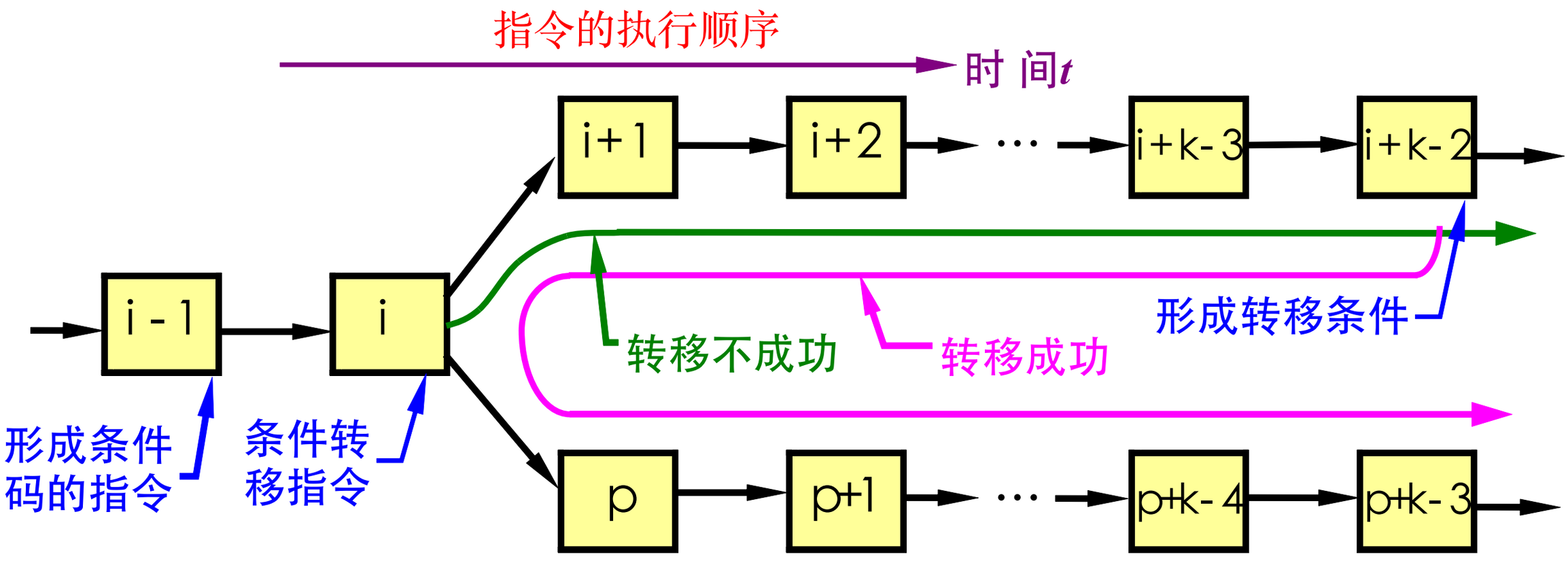
- 在条件分支的两路上同时启动取指令操作，并将指令保存到分支指令被实际执行时。
- 分支指令执行时，“真”的执行通路即刻可以获得。
- 是对预取分支目标的发展。
- 是更暴力的解决方案：一个流水线对应一个分支

方法4：循环缓冲器（loop buffer）

- 循环缓冲器是一个小的、非常高速的存储器，保存着最近获取的n条顺序的指令。如果分支发生，硬件首先检查分支目标是否在缓冲器中。如果在，下一条指令从缓冲器中获取。
- 循环缓冲器有以下好处：
  - 当分支未发生时，顺序获取的指令已在缓冲器中。
  - 如果循环缓冲器有一定的容量，且目标地址仅仅是在分支指令之后的几个单元处，则当分支发生时，目标将已在缓冲器中。对常见的 if-then 和 if-then-else 语句有利。
  - 如果循环缓冲器大到足以容纳一个循环中的全部指令，则循环中的指令仅需从内存中读出一次。
- CDC 的 Star-100、6600、7600 和 CRAY-1。

方法5：分支预测（branch prediction）

猜选 *i*+1 和 *p* 中的一个分支继续流入流水线。



方法5：分支预测（branch prediction）

条件分支在流水线中的执行过程：

因为第*i*条指令所需要的条件码由第*i*-1条指令给出；

在一条由*k*个功能段的流水线中，第*i*-1条指令要等到第*i*+*k*-2条指令进入流水线时才能形成条件码。

- 转移不成功，猜测正确，流水线的吞吐率和效率没有降低；
- 转移成功，猜测错误，要先作废流水线中已经执行的*i*+1、*i*+2、……、*i*+*k*-2指令；然后再从分支点开始执行第*p*、*p*+1、……指令。一条*k*段流水线有*k*-2个功能段是浪费的。

方法5：分支预测（branch prediction）

1. 静态分支预测（static branch prediction）

- 可以采用的预测方法：
  - 预测分支不会发生（predict never taken）
    “出错检测处理”
  - 预测分支总是发生（predict always taken）
    “循环”
  - 由编译器预测
  - 测试法（profiling）
    实际运行该程序(一般是在模拟器上)，然后将有关信息送给编译器。

方法5：分支预测（branch prediction）

1. 静态分支预测（static branch prediction）

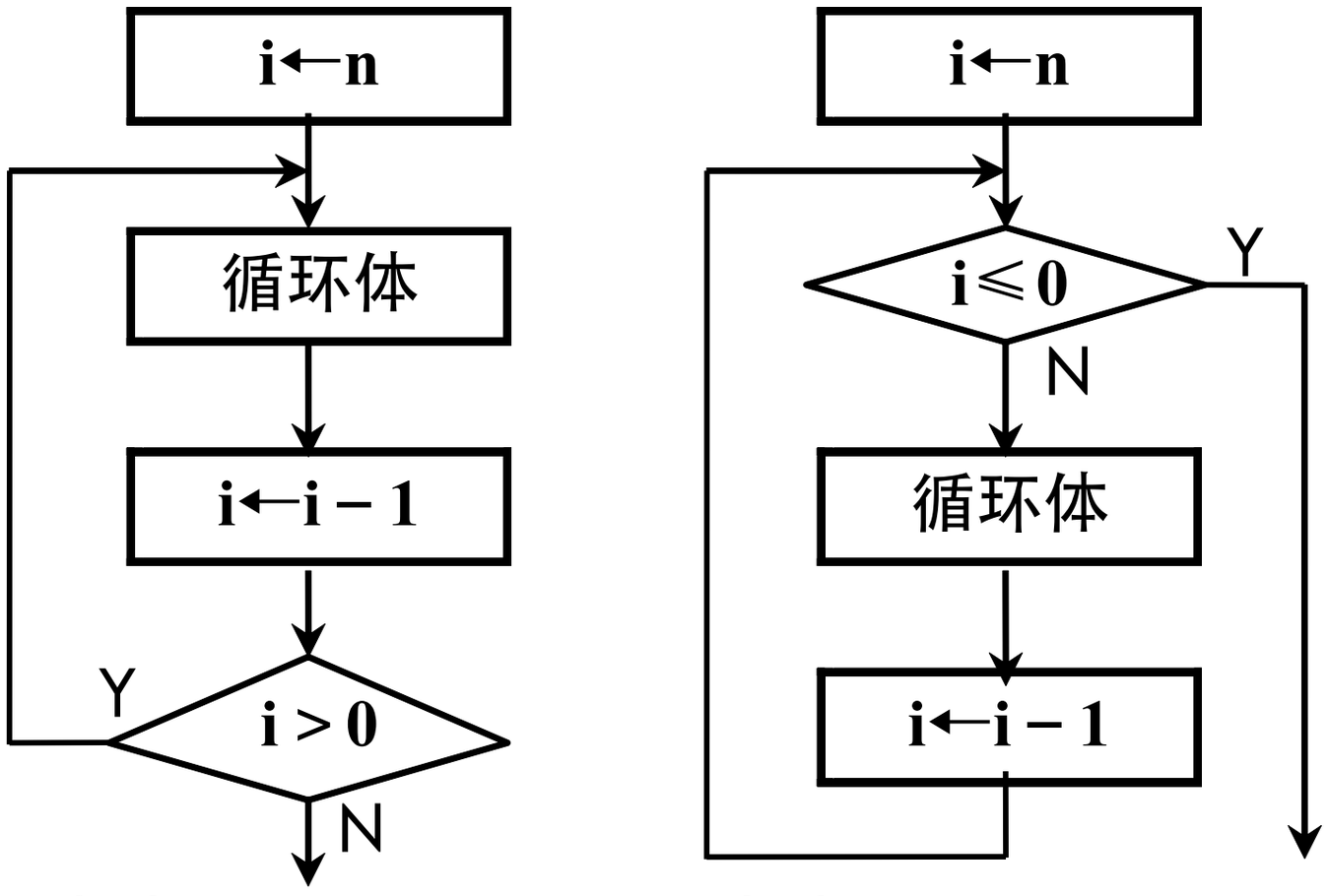
- 如果两个分支概率相近，选 *i*+1、*i*+2、…… 不成功转移分支。
- 转移的两个分支概率不均等，则猜选高概率的分支。

如：

软件猜测法，通过编译器尽量降低转移成功的概率。

方法5：分支预测（branch prediction）

1. 静态分支预测（static branch prediction）



（a）原来程序　　（b）一般条件转移

方法5：分支预测（branch prediction）

1. 静态分支预测（static branch prediction）

- 要保证猜错时可恢复分支点处原来的现场，3种方法：

  ① 只译码和准备好操作数，在转移条件码出现之前不运算。如：IBM360/91

  ② 执行，但不送回运算结果。

  ③ 采用后援寄存器，将原始状态都保存起来，一旦猜错就取出后援寄存器的内容恢复分支点现场。

方法5：分支预测（branch prediction）

2. 动态分支预测（dynamic branch prediction）

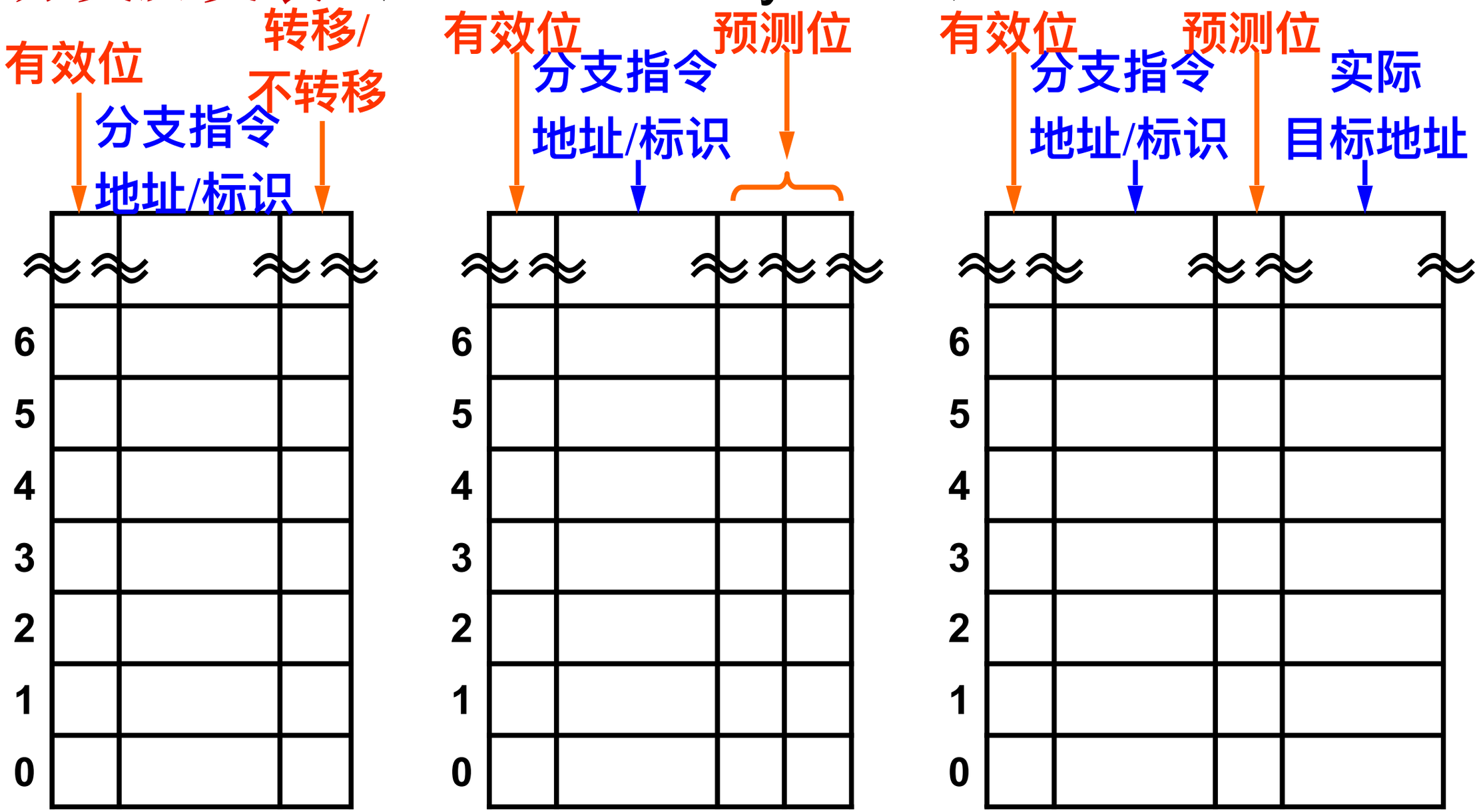
- 通过记录分支指令的近期运行历史，并以此作为预测的依据，来提高分支预测的准确度。
- 分支历史表（branch history table）

  也称分支预测缓存（branch-prediction buffer）

方法5：分支预测（branch prediction）

2. 动态分支预测（dynamic branch prediction）

- 分支历史表（branch history table）



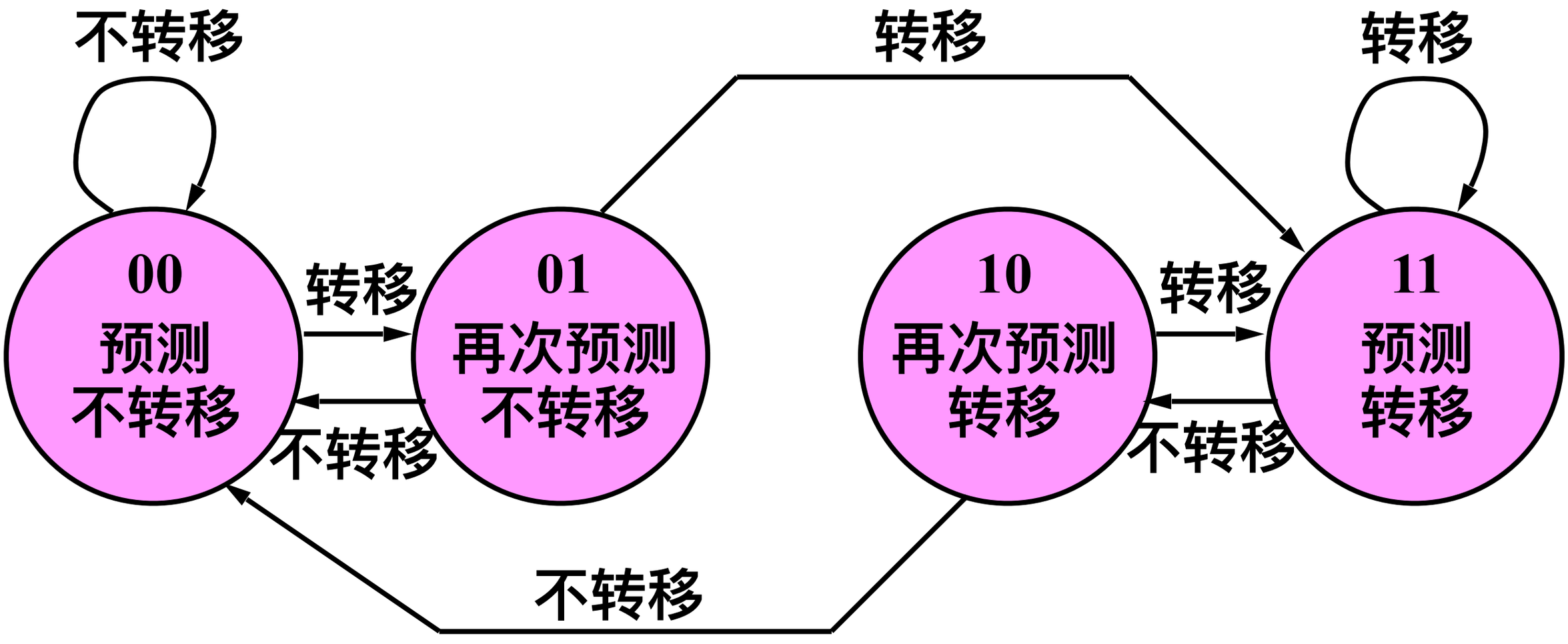
1位跳转历史位　　2位跳转历史位　　分支指令地址和实际目标地址之间的映射

图：分支历史表

方法5：分支预测（branch prediction）

2. 动态分支预测（dynamic branch prediction）

- 分支历史表（branch history table）



图：　用于分支预测的2位有限状态机

方法6：延迟分支（delayed branch）

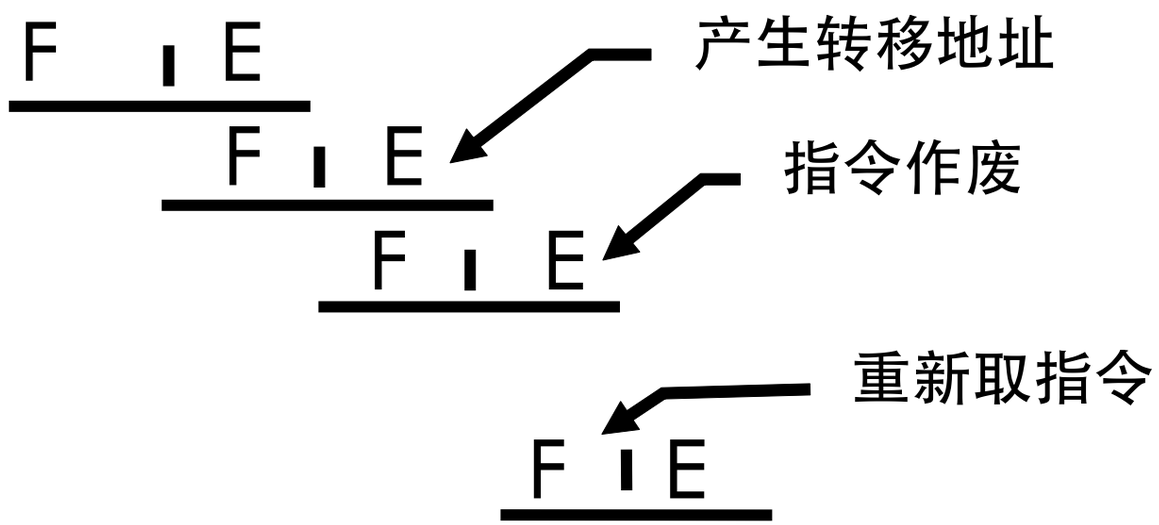
延迟转移技术

- 为了使指令流水线不断流，在转移指令之后插入一条没有数据相关和控制相关的有效指令，而转移指令被延迟执行，这种技术称为延迟转移技术。
- 采用指令延迟转移技术时，指令序列的调整由编译器自动进行，用户不必干预。
- 读采用延迟转移的程序，必须十分小心。

方法6：延迟分支（delayed branch）

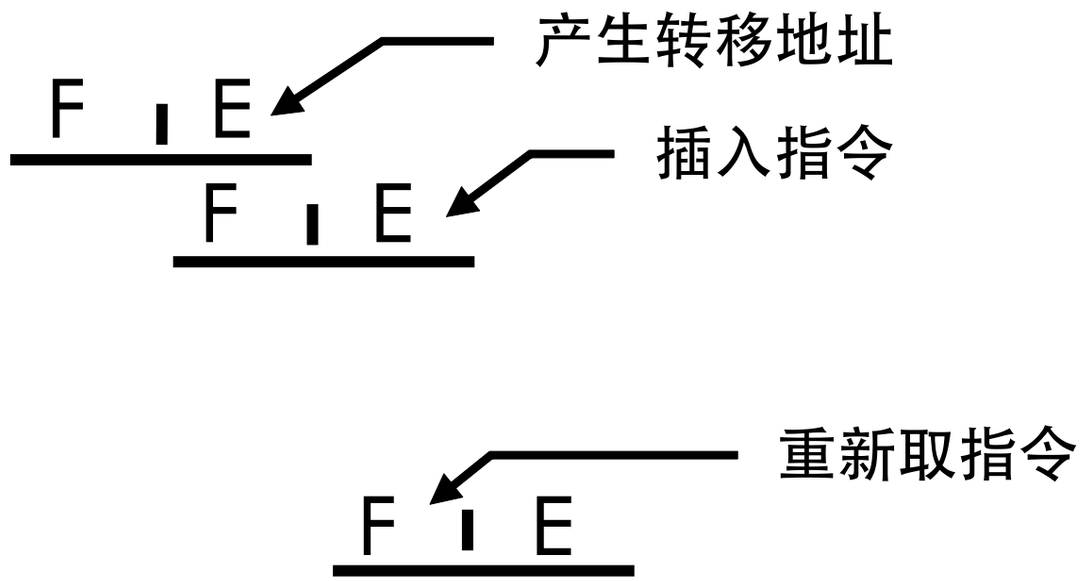
延迟转移技术　无条件转移指令的延迟执行：

```
1:          ADD R1,R2      1:
2:          JMP NEXT2      2:
3: NEXT1:   SUB R3,R4      3:
           ……
n: NEXT2:   MOVE R4,A      n:
```



因转移指令引起的流水线断流

```
1:          JMP NEXT2      1:
2:          ADD R1,R2      2:
3: NEXT1:   SUB R3,R4      3:
           ……
n: NEXT2:   MOVE R4,A      n:
```



采用延时转移技术的指令流水线

方法6：延迟分支（delayed branch）

延迟转移技术

条件转移指令的延迟执行

- 调整前的指令序列：

```
1：  MOVE R1, R2
2：  CMP  R3, R4    ;(R3)与(R4)比较
3：  BEQ  EXIT      ;如果(R3)=(R4)则转移
     ……
NEXT: MOVE  R4, A
```

- 调整后的指令序列：

```
1：  CMP R3, R4     ;(R3)与(R4)比较
2：  BEQ EXIT       ;如果(R3)=(R4)则转移
3：  MOVE R1,R2     ;被插入的指令
     ……
NEXT: MOVE  R4, A
```

方法6：延迟分支（delayed branch）

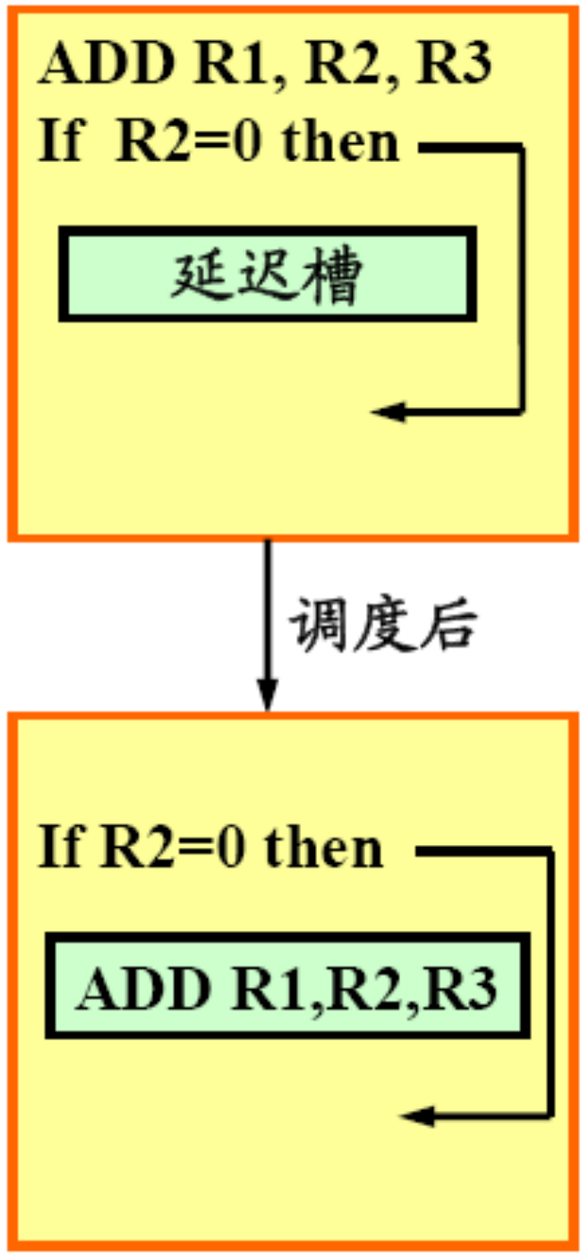
延迟转移技术

- 采用延迟转移技术的两个限制条件：
  - 被移动指令在移动过程中与所经过的指令之间没有数据相关。
  - 被移动指令不破坏条件码，至少不影响后面的指令使用条件码。
- 如果找不到符合上述条件的指令，必须在条件转移指令后面插入空操作。
- 如果指令的执行过程分为多个流水段，则要插入多条指令。

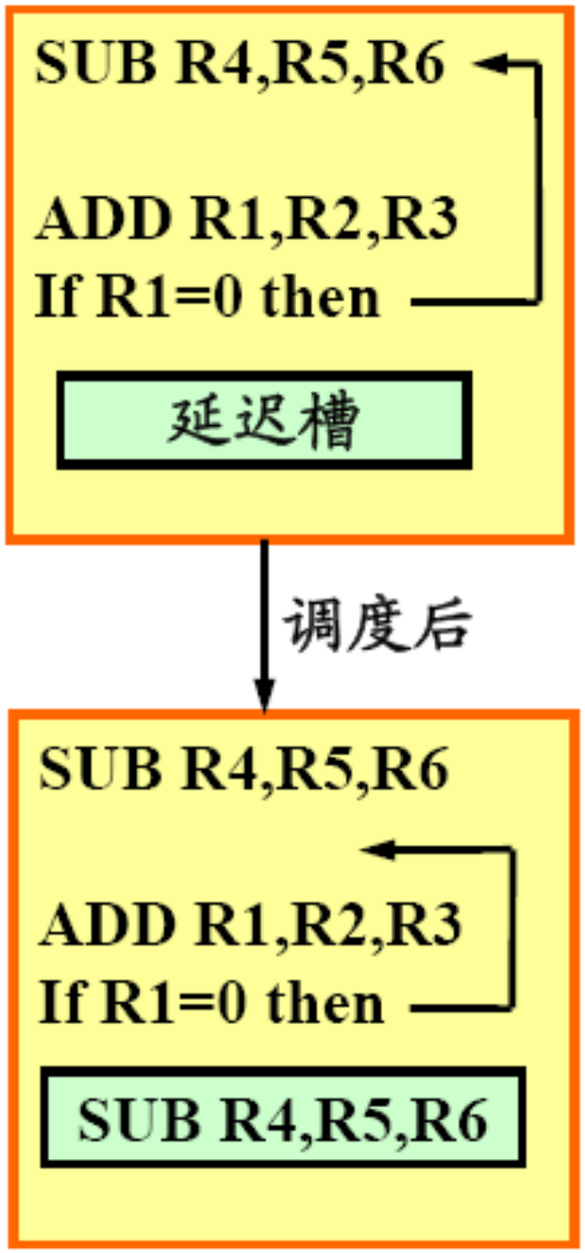
方法6：延迟分支（delayed branch）

- 流水线遇到分支指令时，按正常方式处理，同时执行延迟槽中的指令。
- 编译器的任务就是在延迟槽中放入有用的指令，称为延迟槽调度。有三种调度方法：
  - 从分支前（from before）调入
  - 从目标处（from target）调入
  - 从失败处（from fall-through）调入
- 采用延迟分支法的限制：
  - 放入延迟槽的指令需要满足一定的条件
  - 编译器要有预测分支是否成功的能力

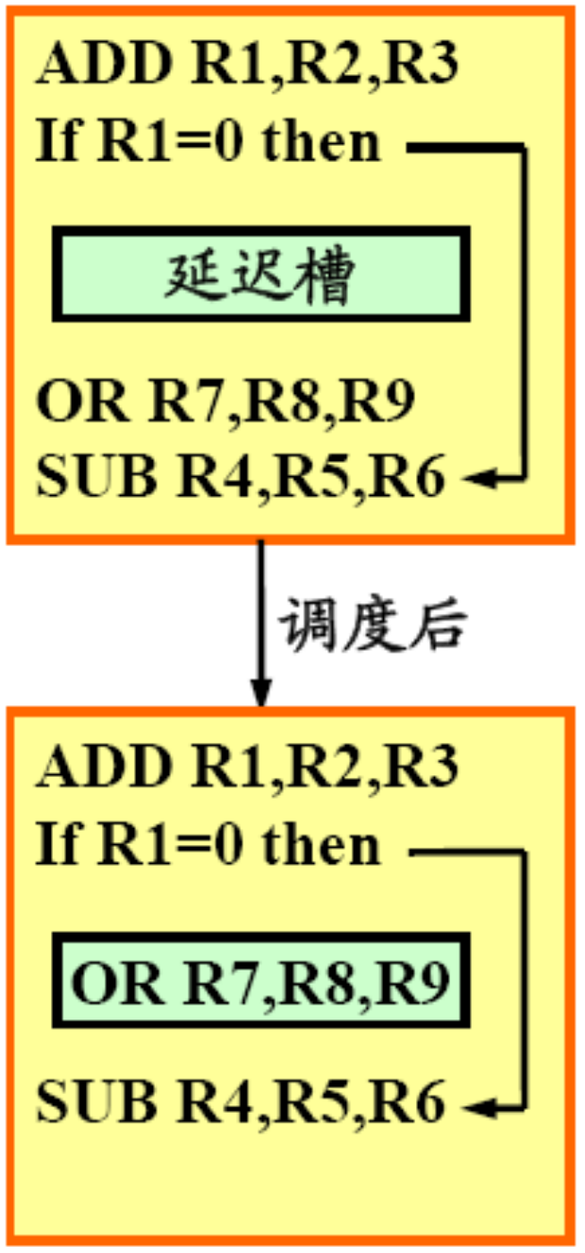
调度延迟槽的方法



(a) 从分支前
(被调度的指令必须与分支无关)



(b) 从目标处
(必须保证在分支失败时执行被调度的指令不会导致错误。有可能需要复制指令)



(c) 从分支失败处
(必须保证在分支成功时执行被调度的指令不会导致错误)

方法6：延迟分支（delayed branch）

指令取消技术

采用指令延时技术，经常找不到可以用来调整的指令，可考虑采用另一种方法：指令取消技术

(1) 向后转移（适用于循环程序）

- 循环体的第一条指令安放在两个位置，分别在循环体的前面和后面。
- 如果转移成功，则执行转移指令(循环体)后面的指令，然后转移到目标地址(循环体的开始位置)；
- 否则取消转移指令(循环体)后面的指令。

(2) 向前转移

- 如果转移不成功，执行转移指令之后的下条指令；
- 否则取消下条指令，转移到目标地址执行。

方法6：延迟分支（delayed branch）

指令取消技术　　向后转移的例子：

```
LOOP: X X X
      Y Y Y
      Z Z Z
      ……
      COMP R1,R2,LOOP
      W W W
```

```
      X X X
LOOP: Y Y Y
      Z Z Z
      ……
      COMP R1,R2,LOOP
      X X X
      W W W
```

效果：

能够使指令流水线在绝大多数情况下不断流。

对于循环程序，由于绝大多数情况下，转移是成功的。

只有最后一次出循环时，转移不成功。

- 转移指令引起的控制相关，对流水计算机吞吐率、效率的影响比数据相关严重的多，所以被称为全局性相关。
- 而数据相关一般被称为局部性相关。

6.4 指令级并行概念

- 指令级并行： Instruction-Level Parallelism
- 开发指令级并行的方法：
  - 依赖于硬件，动态地发现和开发指令级并行。
    Intel的Pentium系列
  - 依赖于软件技术，在编译阶段静态地发现并行。
    Intel的Itanium处理器

6.4.1 指令流水线的限制

- 增加指令发射的宽度和指令流水线的深度，要求复杂硬件电路和高频率时钟的支持。这导致CPU功耗的上升。
- 目前已逐渐形成的共识是，功耗是限制当代处理器发展的首要因素。

6.4.2 突破限制的途径

- CPU时钟频率
- 流水线深度
- 从更深层次地解决流水线中可能存在的各种相关性
- 多核CPU，多指令流水线
  - 流水线内部
  - 流水线之间
  - 指令之间
  - 线程之间

- 现代处理器中，比较成熟的提高指令级并行的技术：

突破限制的途径

技术	简要说明	主要解决问题
直通和旁路	在流水线段间建立直接的连接通路	潜在的数据相关停顿
简单转移调度	冻结流水线，预取分支目标，多流，循环缓冲器等	控制相关停顿
延迟分支	利用编译器调度，填充延迟槽	控制相关停顿
基本动态调度（记分板）	乱序执行	真相关引起的数据相关停顿
重命名动态调度	**WAW**和**WAR**停顿，乱序执行	数据相关停顿、反相关和输出相关引起的停顿
分支预测	动态分支预测，静态分支预测	控制相关停顿
多指令发射	多指令流出（超标量和超长指令字）	理想**CPI**
硬件推测	用于多指令发射，使用重排序缓存	数据相关和控制相关停顿

突破限制的途径

表 提高指令级并行的技术（续）

技术	简要说明	主要解决问题
循环展开	将循环展开为直线代码，消除判断、分支开销，加速流水	控制相关停顿
基本编译器流水线调度	对数据相关指令重排序	数据相关停顿
编译器相关性分析	利用编译器发现相关	理想CPI，数据相关停顿
软件流水线，踪迹调度	软件流水：对循环进行重构，使得每次迭代执行的指令是属于原循环的不同迭代过程的。 踪迹调度：跨越IF基本块的并行度。	数据相关和控制相关停顿
硬件支持编译器推测	软硬件推测结合	理想CPI，数据相关停顿，转换相关停顿

6.5 指令级高度并行的超级处理器

1. 多指令流出技术

1. **一个时钟周期内流出多条指令，CPI＜1，IPC＞1**
2. **多指令流出处理器有三种基本结构：**

1. 超标量（Superscalar）

每个时钟周期流出的指令数不定，它既可以通过编译器静态调度，也可以通过动态调度

2. 超流水（Super Pipeline）

将每个功能部件进一步流水化，特别是取指令或指令流出被分解为多个段，使得一个功能部件在一拍中可以处理多条指令。

1. 超长指令字VLIW（Very Long Instruction Word，简记为VLIW）

每个时钟周期流出的指令数是固定的，它们构成一条长指令，或说是一个混合指令包，这种处理器目前只能通过编译静态调度。

2. 指令级高度并行的超级处理器

超标量处理机

超长指令字处理机

超流水线处理机

超标量超流水线处理机

超级计算机

表　基于指令流水线技术的4种不同类型处理机的性能比较

机器类型	k段流水线基准标量处理机	m度超标量处理机	n度超流水线处理机	(m, n)度超标量超流水线处理机
机器流水线周期	1个时钟周期	1	1/n	1/n
同时发射指令条数	1条	m	1	m
指令发射等待时间	1个时钟周期	1	1/n	1/n
指令及并行度ILP	1	m	n	m×n

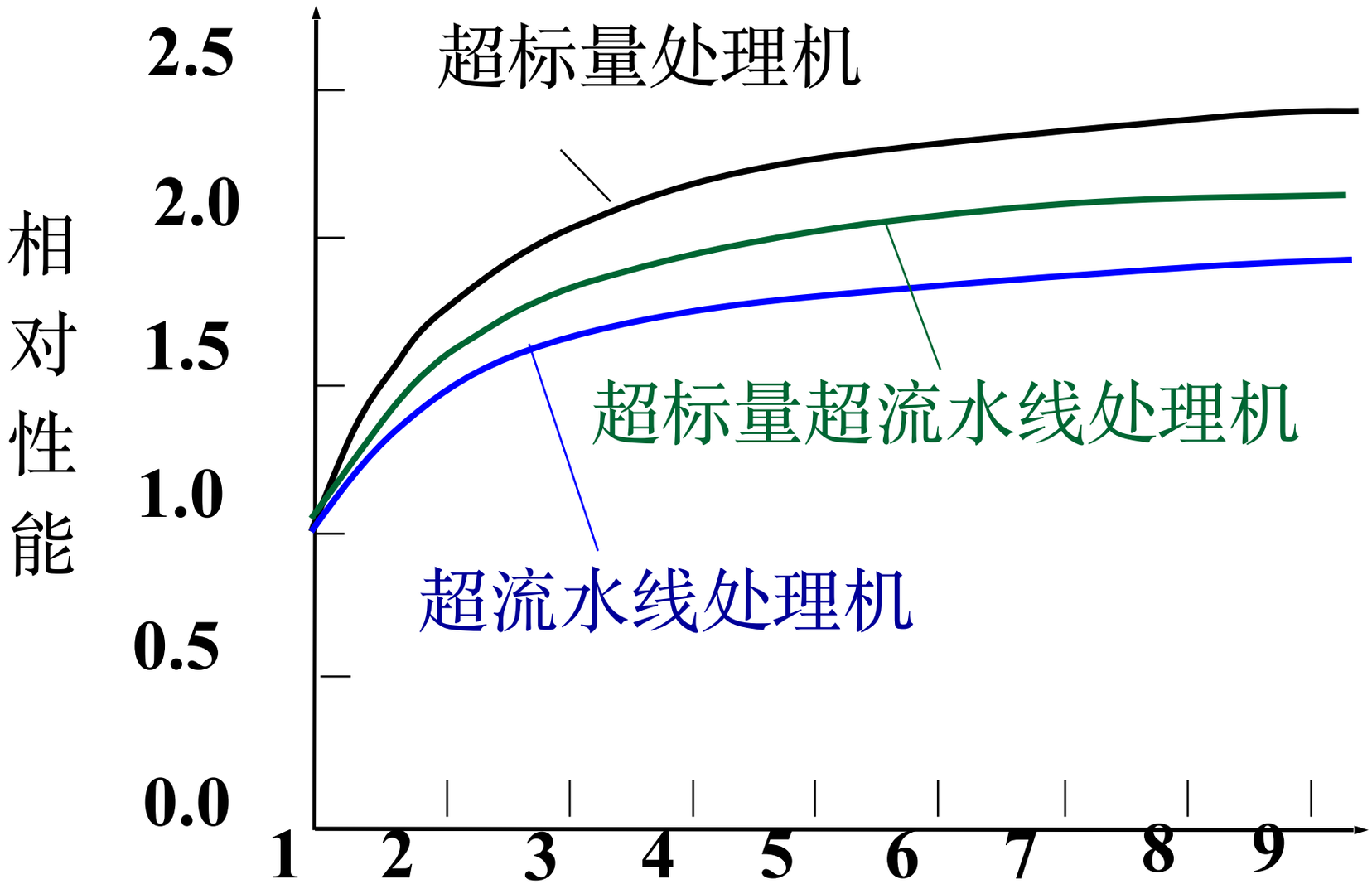


图 3种指令并行处理机的相对性能

可得出以下结论：

（1）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。主要有三方面原因：

- 超标量处理机在每个时钟周期的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，指令之间的启动延时比超标量处理机大；
- 条件转移造成的损失，超流水线处理机比超标量处理机大；
- 在指令执行过程中的每一个功能段，超标量处理机都设置有多个相同的操作部件，而超流水线处理机只是把同一条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比超流水线处理机小。

(2) 当横坐标表示的设计指令级并行度较小时，处理机实际指令并行度提高较快。但当设计指令级并行度进一步增加时，处理机实际指令并行度提高变缓，且越来越慢。因此实际设计超标量、超流水线或超标量超流水线处理机的指令并行度应适当，否则花费大量硬件代价，而可能得不到指令并行度的期望值。

(3) 一个特定程序由于受到本身的数据相关、控制相关等限制，其指令并行度的最大值是确定的。这个最大值由程序自身的语义决定，与这个程序运行于哪一种处理机无关。因此图中的三条曲线对于某一特定程序，会收拢于同一个点上。当然对不同程序，其收拢点位置是不同的。

（4）并非流水线级数越多会越好，也并非处理机工作频率越高越好。例如Pentium4的深度流水线和极高的主频并没有带来所期望的高性能和高效率，迫使Intel不得不放弃从这一途径来提高处理器性能，转而开发多核处理器和超线程技术，开创了处理器的多核时代和深化了线程级并行技术，使处理机性能迈上一个新的台阶。

- 多核处理器：将指令级并行上升到了线程级并行
  - 是目前可以替代并超越超标量处理器和超长指令字处理器的最佳选择