Empty Page

流水线3

李瑞　副教授
蒋志平　讲师
计算机科学与技术学院

叙旧
&
温故
&
知新

仍然关于流水线优化...

Branch Target Prefetching (预取分支目标)

- Use more hardware:
  - Target of branch is prefetched in a buffer (queue)
  - The instructions following the branch are executed normally in pipeline
- Keep target until branch is executed

Multi-Streaming
(多流)

- Use even more hardware: two full pipelines!
  - Fetch each branch into a separate pipeline and execute them in parallel
  - When decision was made, use appropriate pipeline, discard the other
- Problems:
  - Increased bus & register contention
  - Multiple branches in a row lead to further pipelines being needed

Loop Buffer
(循环缓存)

- Extends the prefetch approach in another way
  - Very fast memory (instruction cache!)
  - Maintained by fetch stage of pipeline
    - Check buffer before fetching from memory
- Very good for small jumps (if-else, if- then-else) and loops
  - Buffer size designed to be able to store all instructions in loop

方法6：延迟分支（delayed branch）

Compiler analyzes the instructions before and after the branch and rearranges the program sequence by inserting useful instructions in the delay steps

Using no-operation instructions

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

Rearranging the instructions

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

问题：CPU如何实现Delayed Brach呢?

答：通过延迟槽（Delay Slot)机制实现

So, what is Delay Slot（延迟槽)？

In computer architecture, a delay slot is an instruction slot that gets executed without the effects of a preceding instruction. The most common form is a single arbitrary instruction located immediately after a branch instruction on a **RISC or DSP** architecture; this instruction will execute even if the preceding branch is taken. —— Wikipedia

要点：

1. RISC or DSP平台

2. RISC/DSP平台上，跳转指令后面紧跟的那一条指令；

3. 无法是否跳转，延迟槽中的命令都会被执行

方法6：延迟分支（delayed branch）

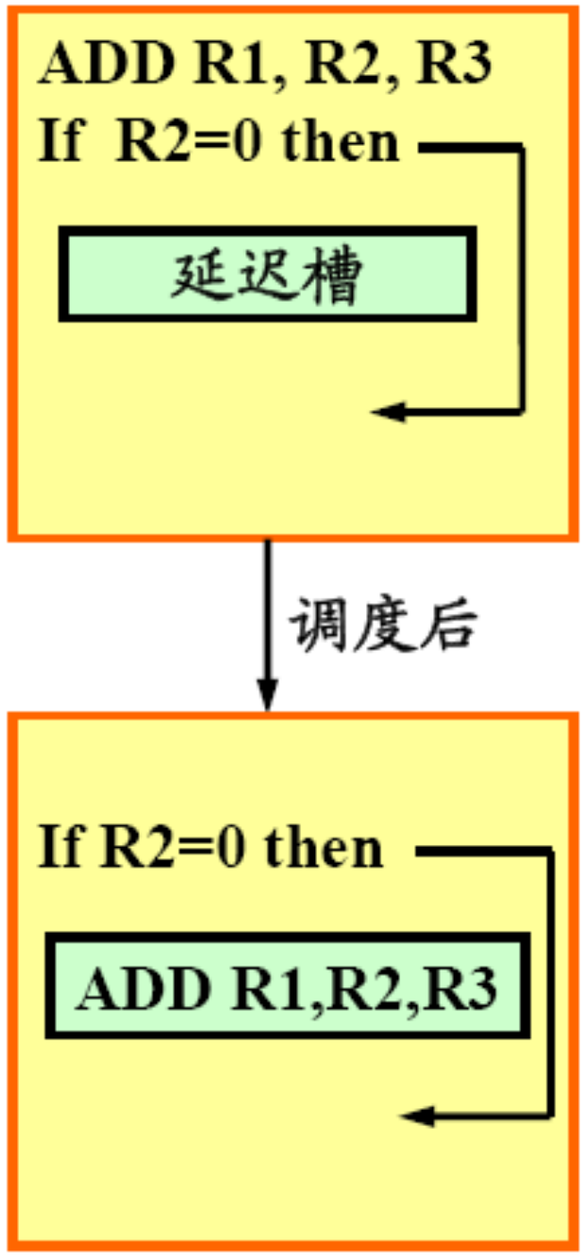
延迟转移技术

- 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。

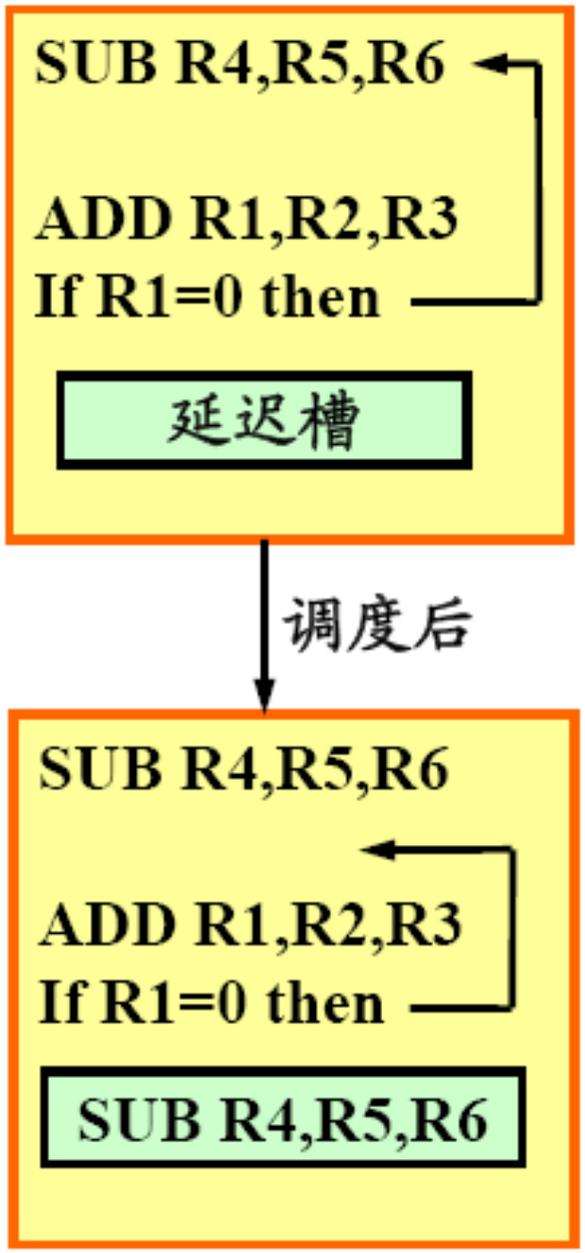
方法6：延迟分支（delayed branch）

- 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

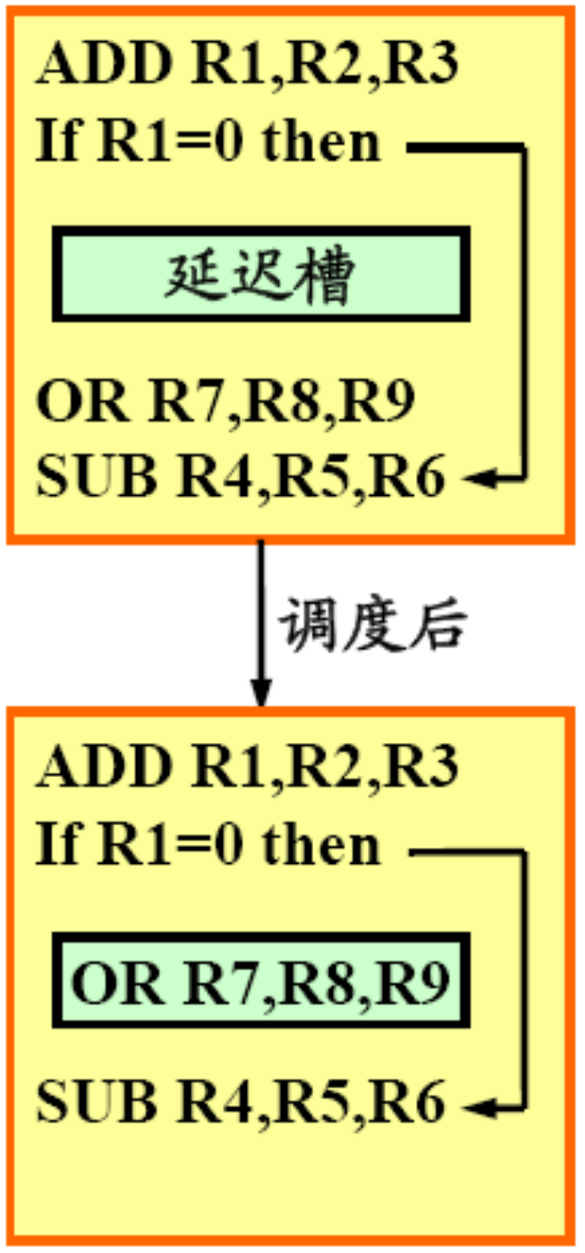
调度延迟槽的方法



(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)

- 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。

Some Latest Updates

- **Hybrid Predictor（混合预测器): 在芯片中布置奇数个不同策略的预测器，通过投票决定分支方向；**
- **Loop Predictor（循环专用预测器):专用用于for/while循环，实现策略简单，主流CPU均内置;**
- **Overriding Branch Predictor(决策覆盖预测器): 决策本身也需要时间，因此快的预测与好的预测之间往往不可兼得；决策覆盖预测器同时使用快的和好的预测器，当好的结果出来时，覆盖快的结果；**
- **Neural Branch Predictor(神经网络预测器): 使用多层感知器(MLP)实现预测器的学习功能；AMD Ryzen就使用了NBP；**

Some Latest Updates

- CPU的性能优化是计算机科学最hardcore的内容，它需要在太多因素之间平衡：吞吐量/延迟/芯片设计/热设计/成本...
- 太多被凝结在芯片中的智慧，不要被课程或考试所淹没...
- 毕竟，计算机体系结构，是一门不断提升人类智能上限的科学...

叙旧
&
温故
&
知新

温故 —关于流水线及“相关”(1/4)

- 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
- 流水线的瓶颈在哪里?
- 如何消减瓶颈?

温故 —关于流水线及“相关”(2/4)

- 什么是“流水线相关”?
- 分别简述结构相关、数据相关以及控制相关
- 结构相关的原因？优化方案?
- 数据相关的原因?
- 数据相关分几类？哪些可能会造成冲突?
- 数据相关有哪些解决方案?
- 控制相关的原因？使执行顺序发生改变的指令有哪两类?
- 优化控制相关有哪些方法?

温故 —关于流水线及“相关”(2/4)

- 简述冻结流水线方法?
- 简述预取分支目标法?
- 简述多流法?
- 简述循环缓冲器法?
- 简述分支预测法？它和上面的方法有何区别?
- 何为静态分支预测？ 分几类?
- 从性能优化的角度看，分支跳转/不跳转有何影响?

温故 —关于流水线及“相关”(4/4)

- 何为动态分支预测？简述其原理
- 概述1-bit分支预测方法?
- 概述2-bit分支预测方法?
- 何为延迟分支? 其动机与原理?
- 采用延迟分支时，有何限制条件?

So many ways to deal w/conditional branch !

- **Do nothing (Freezing / Stalling)**
- **Prefetch branch target**
- **Multiple streams**
- **Loop buffer**
- **Branch prediction**
  - **Static / dynamic**
- **Delayed branching**

Improve Instruction Fetching (IF) speed

Improve Pipeline Utilization

叙旧
&
温故
&
知新

“You mustn't be afraid to dream a little bigger, darling.”

–Eames, quote from movie Inception(2010)

如何全面提升指令执行性能?

- 流水线工作...
- 指令预取、指令缓存、分支预测...
- and ... ?

- 指令级并行(Instruction-Level Parallelism)
  - 静态/动态调度
  - 超长指令字
  - 超流水
  - 超标量
  - 乱序执行
  - 预测执行
  - 微指令缓存
  - 超线程
  - .....

为什么要调度?

尽可能地对指令重排序，使程序中的相关指令尽可能地消除。

静态调度

编译器发现并分离出程序中存在相关的指令，进行指令调度，并对代码进行优化。在出现数据相关时，消除或减少流水线空转。

- 由编译器完成
- 主要消除控制相关
- 无法完全消除数据相关

动态调度

CPU在运行时重新安排指令的执行顺序，减少流水线空转。

- 能调度编译时不知道的竞争情况
- 符号程序执行的实际情况
- 具有更高的效率和准确性
- 简化编译器设计
- 代码移植性强

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

与微指令控制编译中的“水平型微指令”类似，VLIW将可以并行执行的多条指令合并在一起，构成一个位数“超长”的指令。

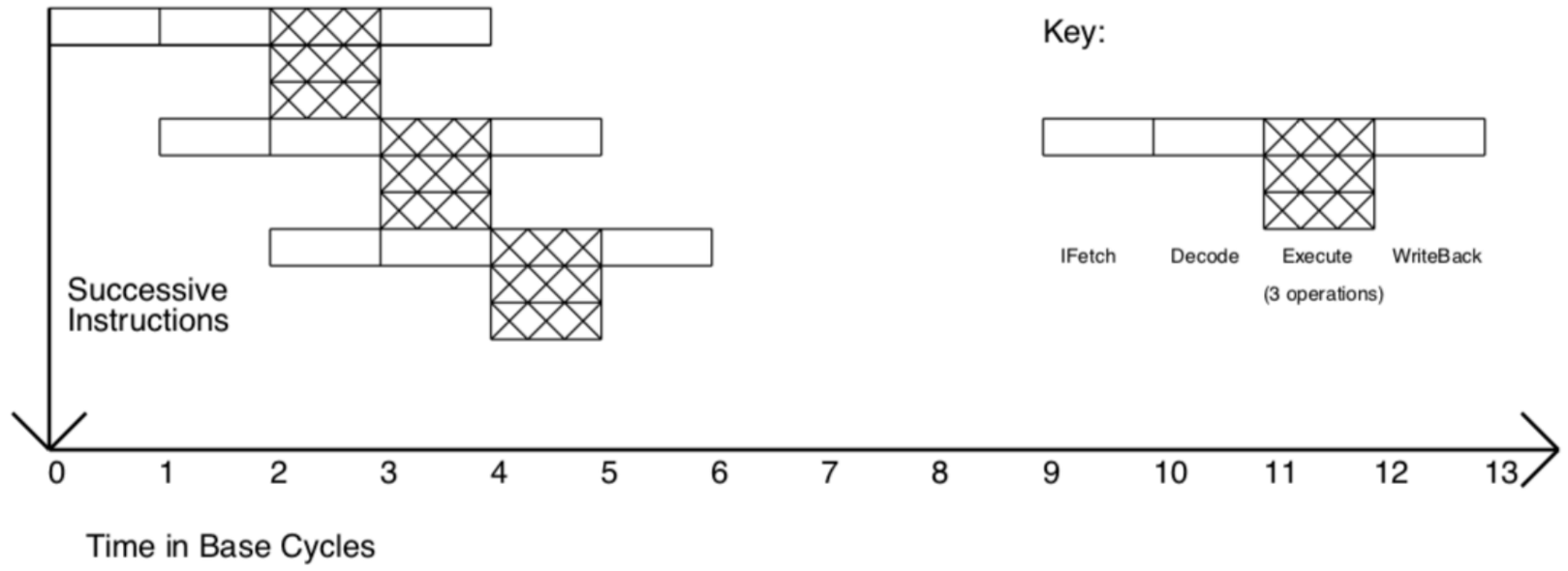


Figure 6: Execution in a VLIW machine

例如，将以下指令打包在一条48bit长的指令里

```
f12 = f0 * f4, f8 = f8 + f12, f0 = dm(i0, m3), f4 = pm(i8, m9);
```

为什么这样设计？ 又是如何实现的?

超流水线（Super Pipeline）

将流水线的IF、ID、ALU等关键步骤进一步细分为多个stages，故称为super-pipeline

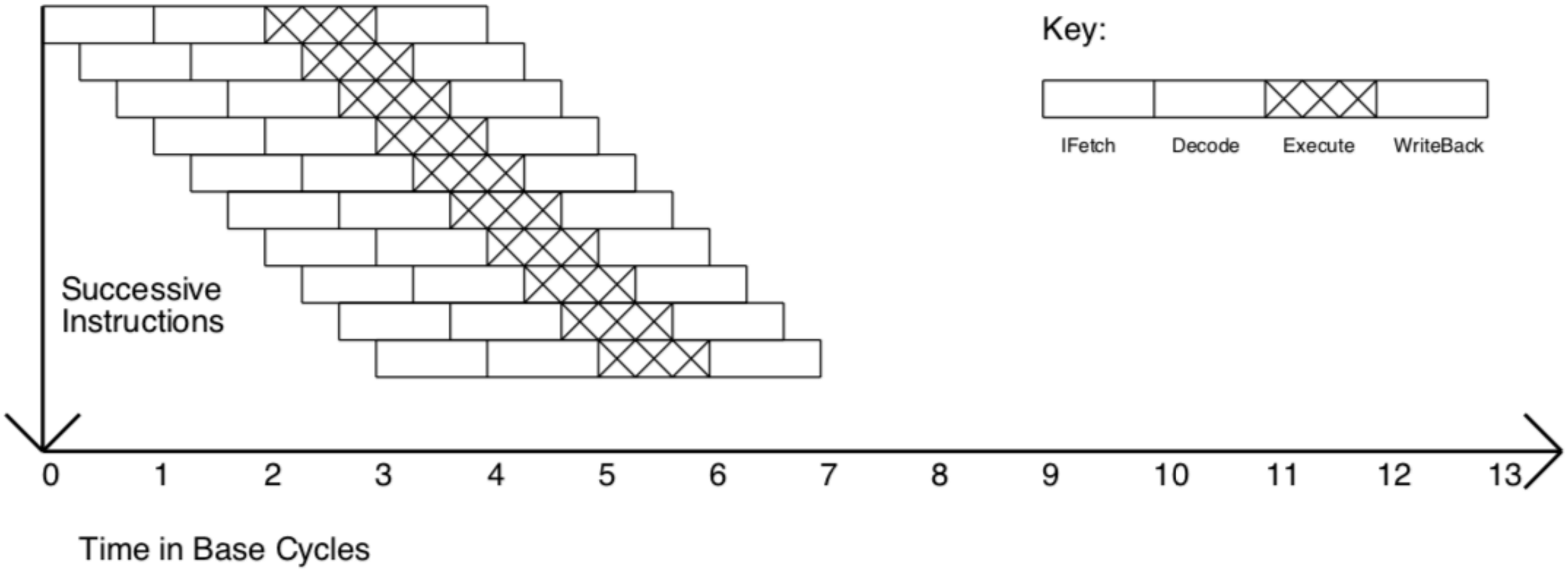
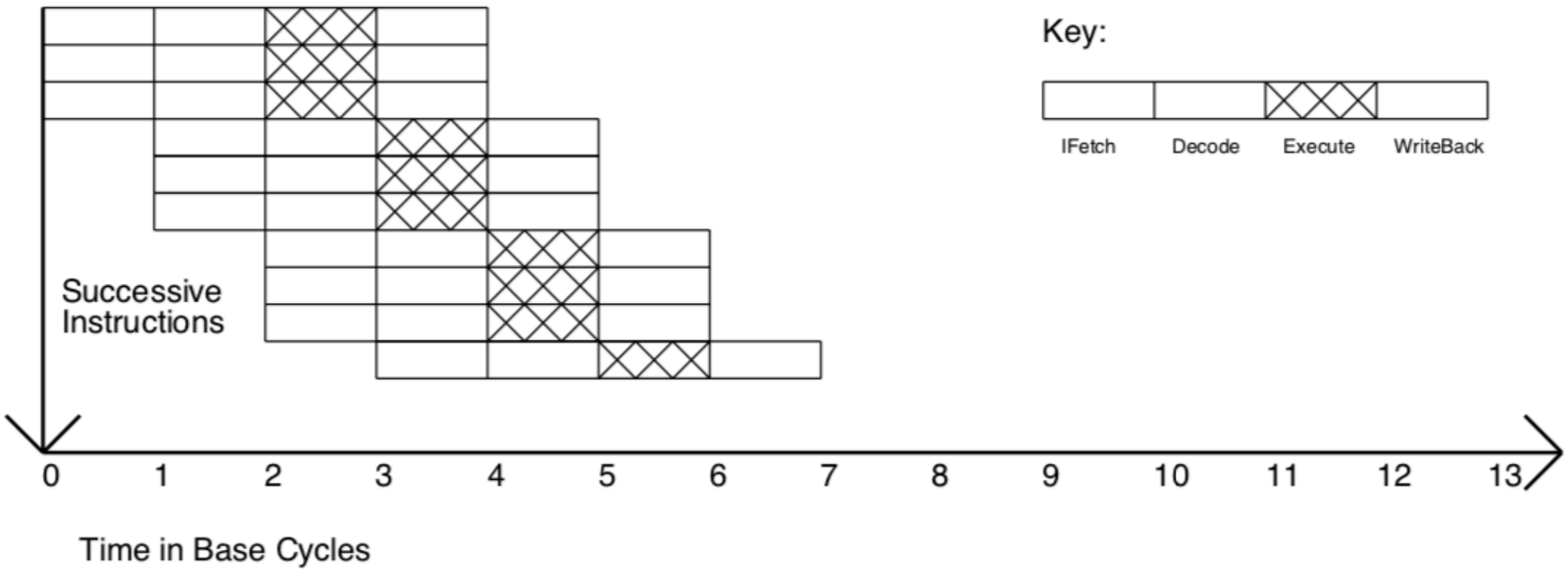


Figure 7: Superpipelined execution (m=3)

超标量流水线（Superscalar）

简单来说：内置多个流水线一起跑！借助硬件资源重来实现空间的并行，实现IPC > 1！



为了实现super-scalar，需要额外堆砌哪些硬件资源?

状态寄存器、译码器、ALU、寄存器组、cache等

超标量超流水（Superpipelined Superscalar）

简单来说：多流水线 + 指令级细分

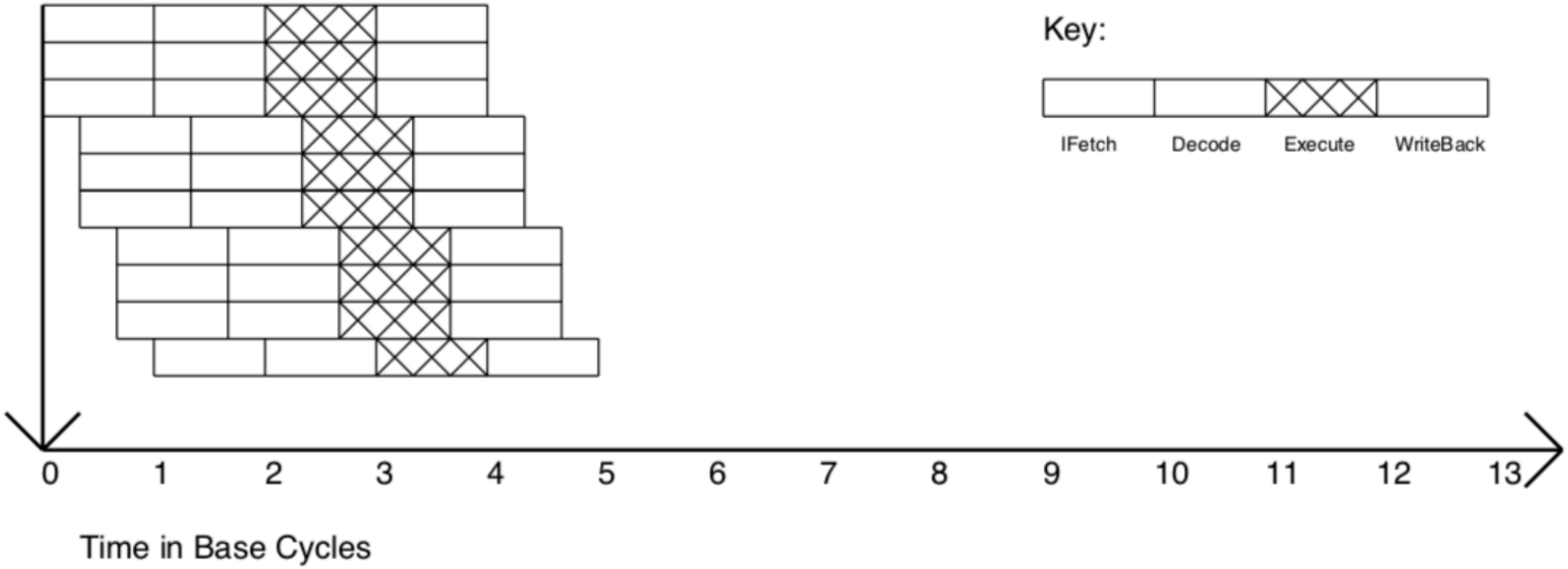


表　基于指令流水线技术的4种不同类型处理机的性能比较

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

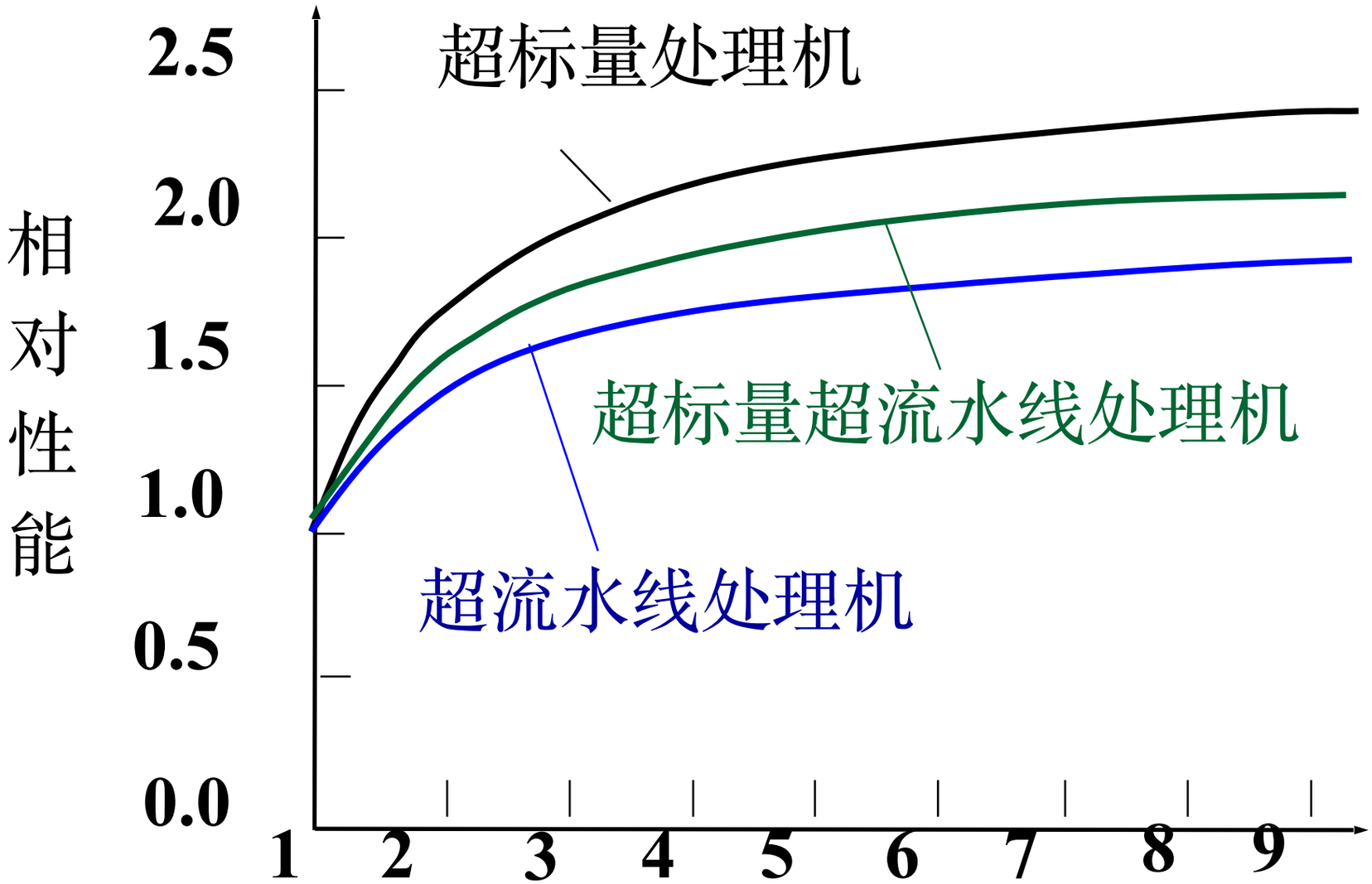


图 3种指令并行处理机的相对性能

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。

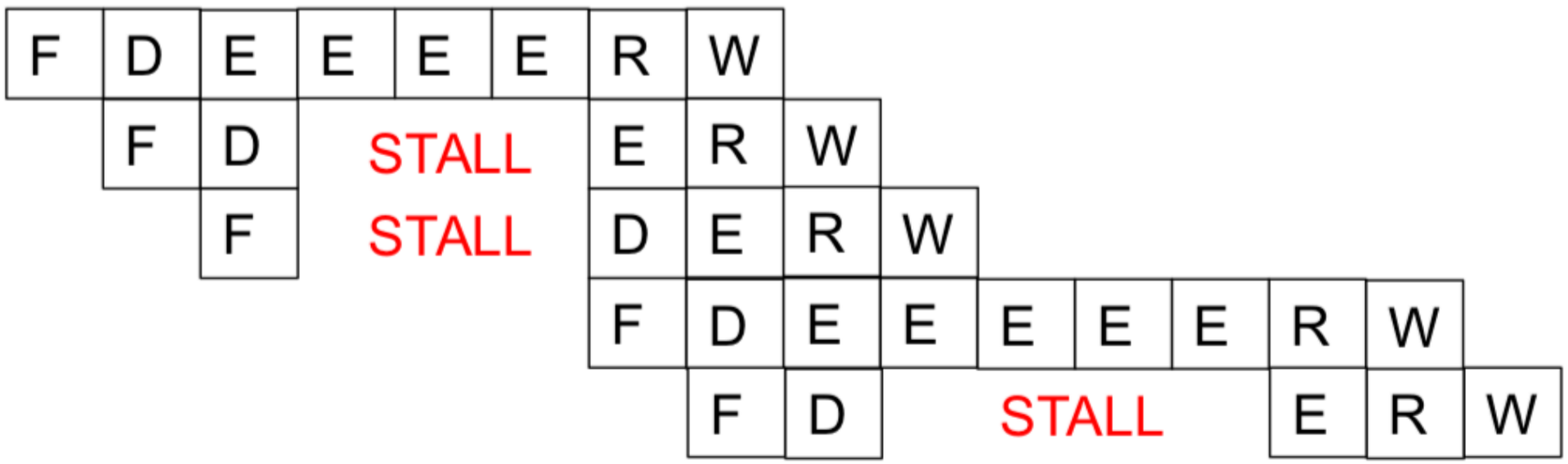
（4）并非流水线级数越多会越好，也并非处理机工作频率越高越好。例如Pentium4的深度流水线和极高的主频并没有带来所期望的高性能和高效率，迫使Intel不得不放弃从这一途径来提高处理器性能，转而开发多核处理器和超线程技术，开创了处理器的多核时代和深化了线程级并行技术，使处理机性能迈上一个新的台阶。

- 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

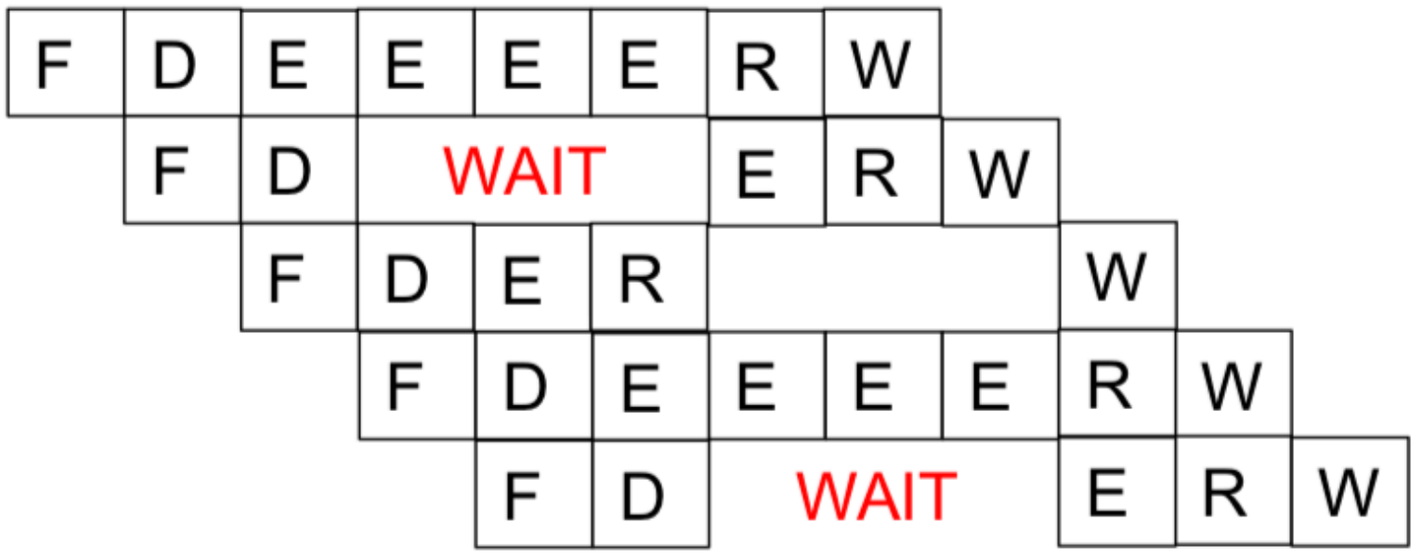
乱序执行（Out-of-Order Execution， OoOE）

相比顺序执行(In-Order Execution)，OoOE在不需要编译器辅助的情况下，就可以实现“面向资源”的指令执行顺序优化。

乱序执行（Out-of-Order Execution， OoOE）



IMUL R3 ← R1, R2
ADD R3 ← R3, R1
ADD R1 ← R6, R7
IMUL R5 ← R6, R8
ADD R7 ← R3, R5



- 16 vs. 12 cycles

乱序执行（Out-of-Order Execution， OoOE）

OoOE的实现需要复杂的硬件和算法支持，但原理上，主要依赖“资源可用性检查”、“寄存器重命名”以及“记分牌”等硬件功能，以及Tomasulo’s等经典OoOE调度算法。

另一方面，OoOE是在超标量CPU基础上的进一步优化，（为什么这么说?)

OoOE一定是SuperScalar，但superScalar不一定是OoOE。

一个In-order的SuperScalar CPU在遇到“相关”时，会有极大的性能损失。

预测执行（Speculative Execution, SE）

相比预取指令、分支预测等，SE是更为激进的一种CPU端优化方法：当面对条件分支时，在多个流水线并行地分别执行“跳转”与“不跳转”分支的指令；当条件结果出来时，再“回滚”错误分支，并提交正确分支的结果。

但显然，“回滚”操作并不完美....

Spectre, Meldown等基于SE的内存访问攻击

并行多线程或“超线程”（Simultaneous Multithreading, SMT, or HyperThreading, HT)

想象一下，如果一个CPU中有10条独立的流水线，如何充分利用？能“轻易”实现并行任务量x10么？

一点揭示：从操作系统中多任务调度的角度去思考...

操作系统通过时分(Time Divided)方法给所有任务分配计算资源；
对于每一个时间片，CPU都完全用于处理一个任务；
但显然，不是所有任务都能用到10条独立的流水线；

如果将每5条流水线“打包”成一个“逻辑CPU”，则OS就多了一倍计算资源！

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