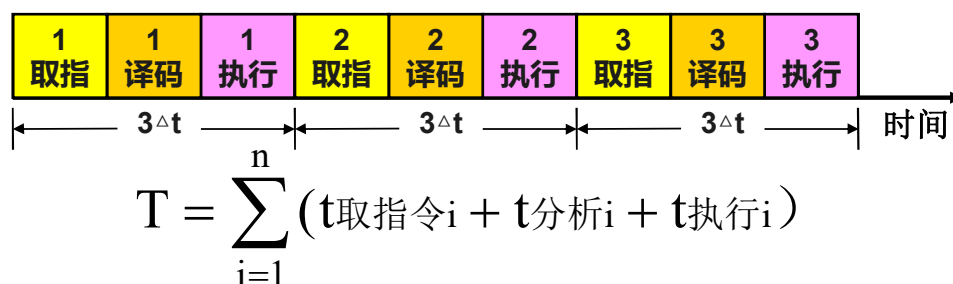




从指令重叠到流水线



如果每段时间均为 t ，则执行 n 条指令所用的时间为：

$$T = 3nt$$

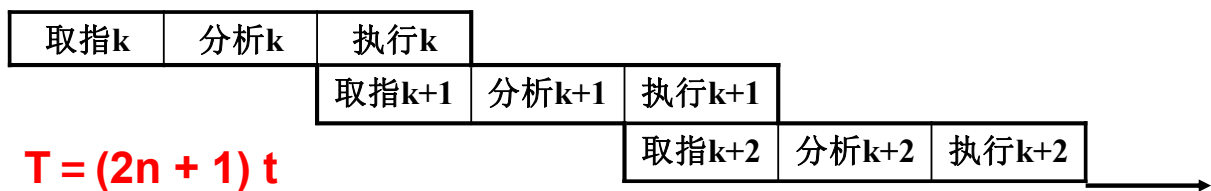
优点：控制简单，成本低；

缺点：执行速度慢，部件利用率低。

指令重叠



从指令重叠到流水线



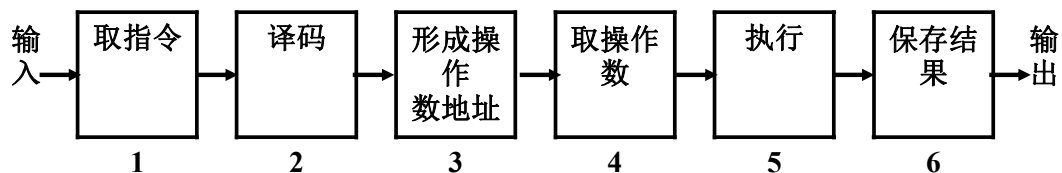
优点和缺点?



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的表示方法

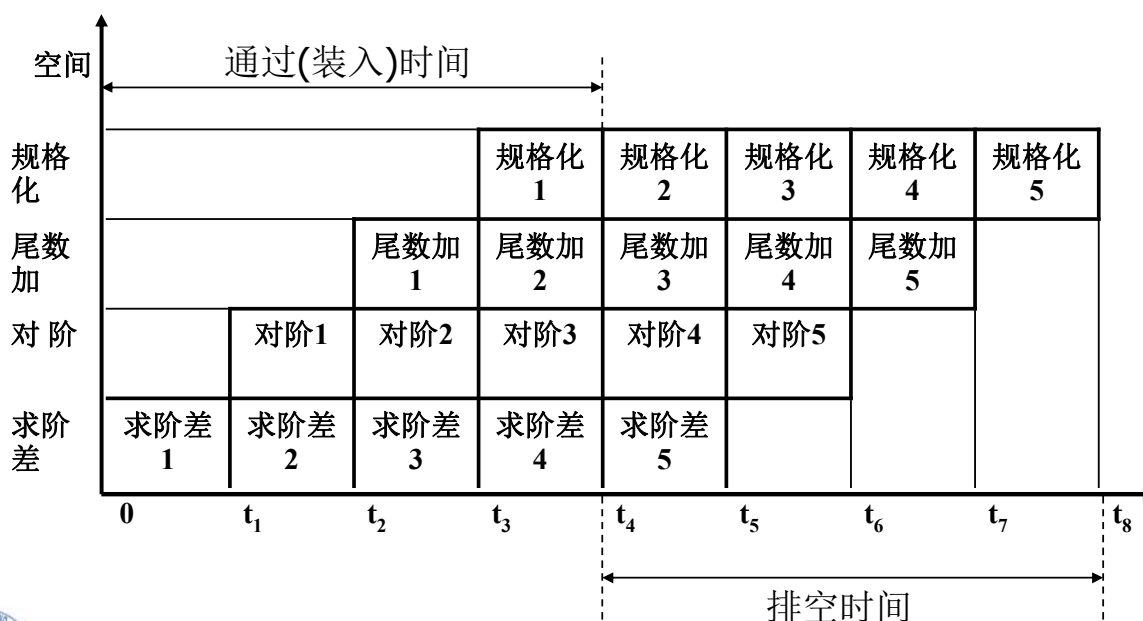
■ 连接图 - 逻辑关系



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

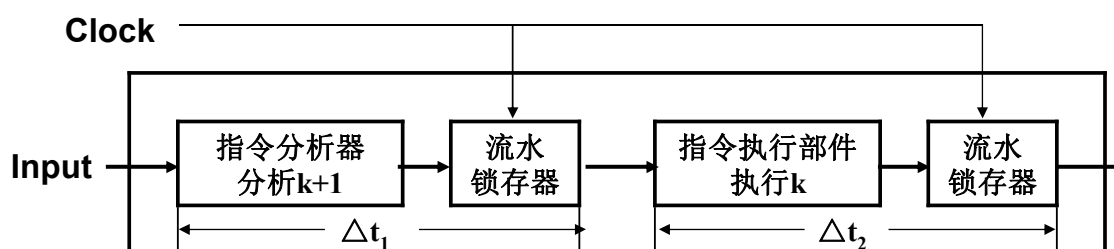
流水线的表示方法

• 时空图 - 时间关系



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的锁存器

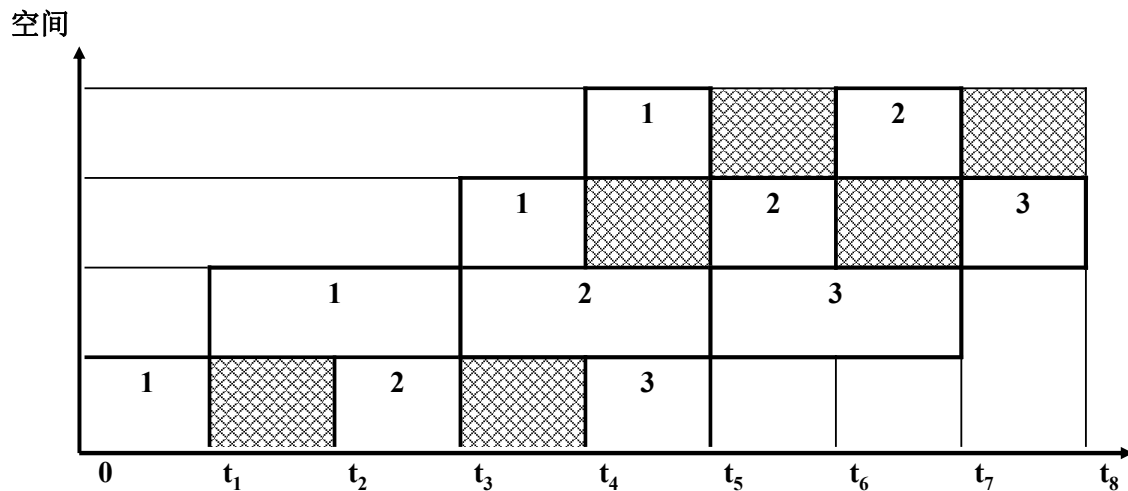


- 流水线每一个功能部件的后面都要有一个缓冲寄存器（锁存器），称为**流水寄(锁)存器**。
- 作用：在相邻的两段之间传送数据，以保证提供后面要用到的数据，并把各段的处理工作相互隔离。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

各段执行时间不相等的流水线



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的特点

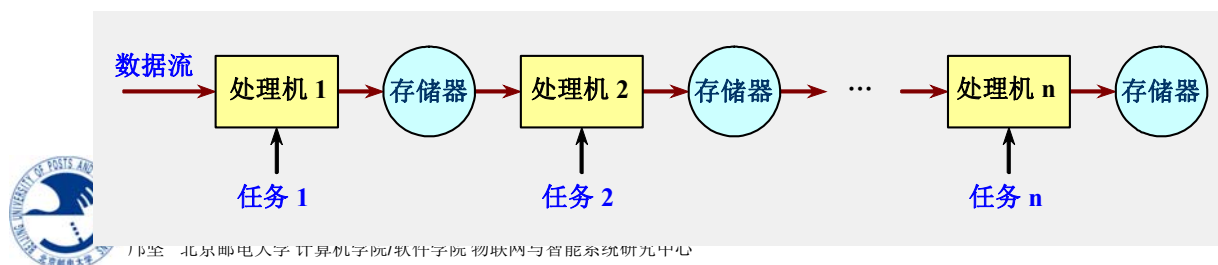
- 流水线把一个处理过程分解为若干个子过程（段），每个子过程由一个专门的功能部件来实现。
- 流水线中各段的时间应尽可能相等，否则将引起流水线堵塞、断流。
 - 时间长的段将成为流水线的瓶颈。
- 效率
 - 连续不断地使用（同一种功能）
 - 通过(装入)时间 – 第一个任务从进入到流出
 - 排空时间 – 最后一个任务从进入到流出



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

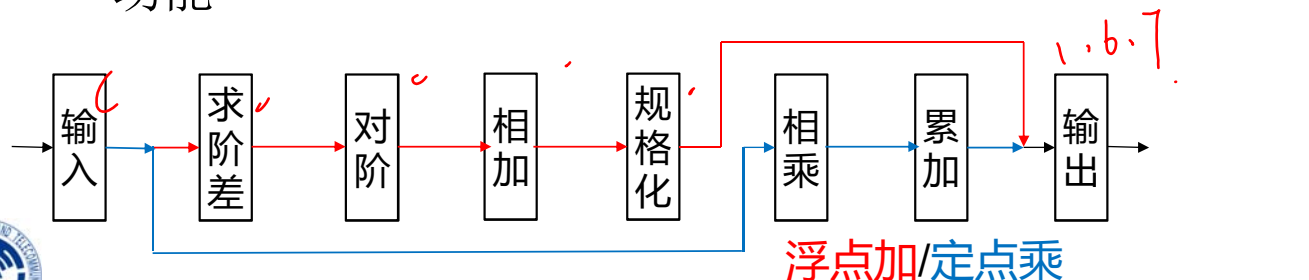
流水线的分类

- 按等级 - 部件级、处理机级和系统级流水线
 - 部件级 - 把处理机中的部件分段相互连接，使运算操作按流水方式处理，如浮点加，也称运算操作流水线 (Arithmetic Pipeline)
 - 处理机级 - 把指令的执行过程分解为若干个子过程，每个在独立的功能部件中执行，即指令流水线 (Instruction Pipeline)
 - 系统级 - 把多个处理机串行连接，对同一数据流进行处理，每个完成任务中的一部分。前一台的结果放入存储器，作为后一台的输入，又称宏流水线 (Macro Pipeline)



流水线的分类

- 按完成功能的多倍性 - 单功能与多功能流水线
 - 单功能(Unifunction Pipeline) - 流水线各段之间的连接固定不变，只能完成一种功能
 - 多功能(Multifunction Pipeline) - 段之间的连接可以变化，不同的连接方式可以完成不同的功能



流水线的分类

- 静态和动态流水线 – 多功能流水线的进一步分类

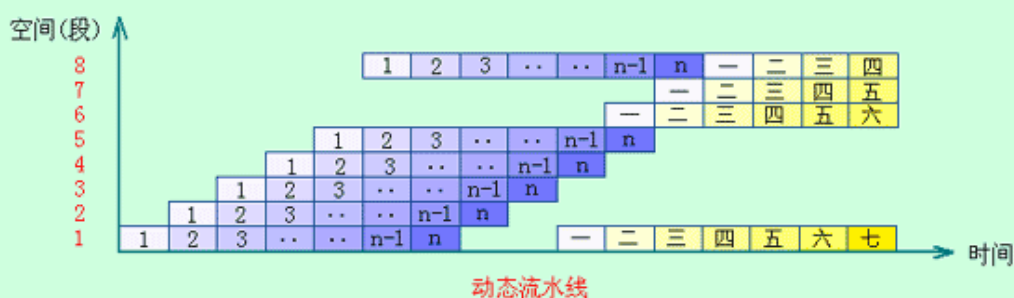
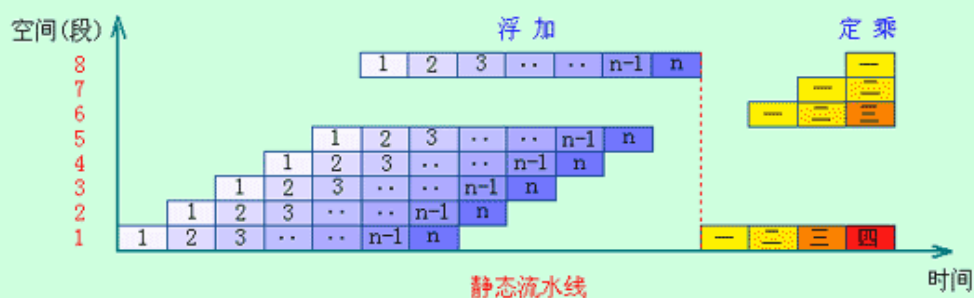
- **静态(Static Pipeline)** - 在同一时间内，多功能流水线中的各段只能按同一种功能的连接方式工作
- **动态(Dynamic Pipeline)** - 在同一时间内，多功能流水线中的各段可以按照不同的方式连接，同时执行多种功能



流水线的分类

静、动态流水线的时空图

假设该流水线要先做几个浮点加法，然后再做一批定点乘法。



流水线的分类

- 按流水线中是否有反馈回路分类 - 线性与非线性流水线
 - 线性流水线(Linear Pipeline) - 流水线的各段串行连接，没有反馈回路。数据通过流水线中的各段时，每一个段最多只流过一次
 - 非线性流水线(Nonlinear Pipeline) - 流水线中除了有串行的连接外，还有反馈回路

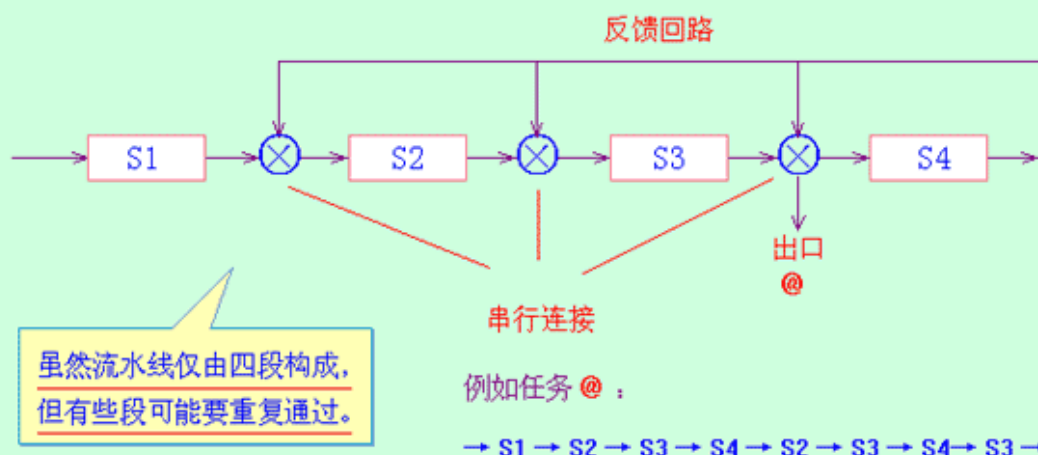


邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的分类

非线性流水线

(举例)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的分类

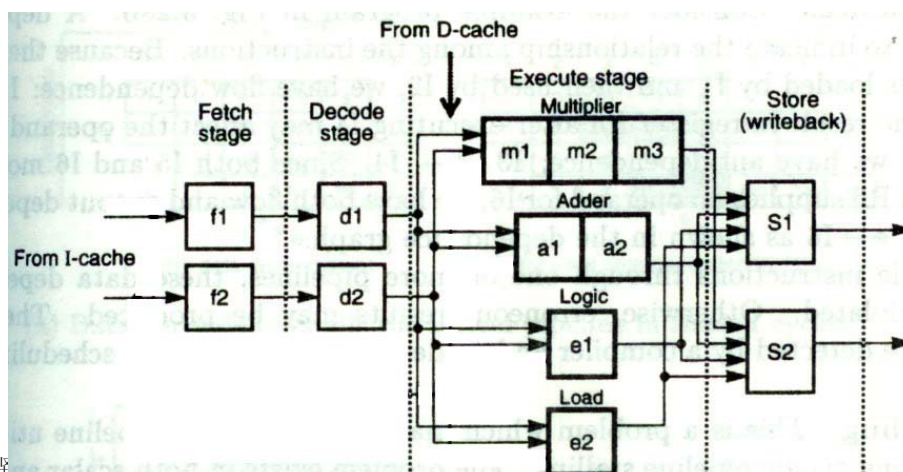
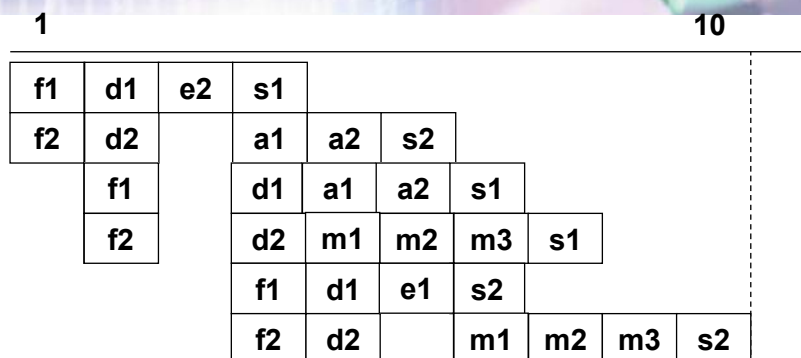
- 根据任务流入和流出的顺序是否分类 - 顺序与乱序流水线
 - 顺序流水线(In-order Pipeline) - 流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。每一个任务在流水线的各段中是一个跟着一个顺序流动的
 - 乱序流水线(Out-of-order Pipeline) - 流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，允许后进入流水线的任务先完成（从输出端流出）。也称为无序流水线、错序流水线、异步流水线



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的分类

LOAD R1, M(A)
 ADD R2, R2, R1
 ADD R3, R3, R4
 MUL R4, R4, R5
 NEG R6, R6
 MUL R6, R6, R7

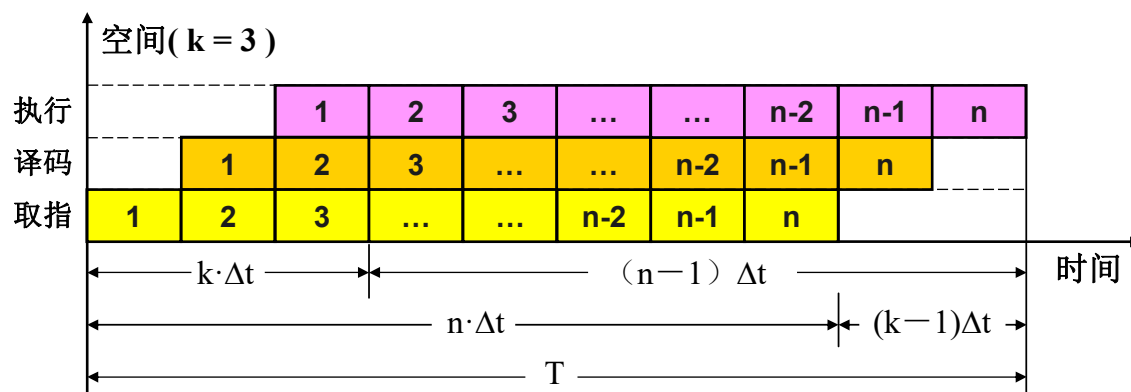


邱坚

流水线的性能指标

■ 吞吐率 (ThroughPut)

- 单位时间流水线所完成的任务数量，或输出结果的数量



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

■ 计算流水线吞吐率的最基本公式:

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

- 各段执行时间相等，输入连续任务情况下：完成n个连续任务需要的总时间为：

$$T_k = (k + n - 1) \Delta t$$

其中：k 为流水线的段数， Δt 为时钟周期。

■ 吞吐率为：

$$TP = \frac{n}{(k + n - 1) \Delta t}$$

流水线的性能

1. 吞吐量

2. 效率

3. 加速比



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

- 最大吞吐率为:

$$TP_{\max} = \lim_{n \rightarrow \infty} \frac{n}{(k + n - 1)\Delta t} = \frac{1}{\Delta t}$$

- 各段执行时间不相等, 输入连续任务

- 吞吐率:

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

- 最大吞吐率:

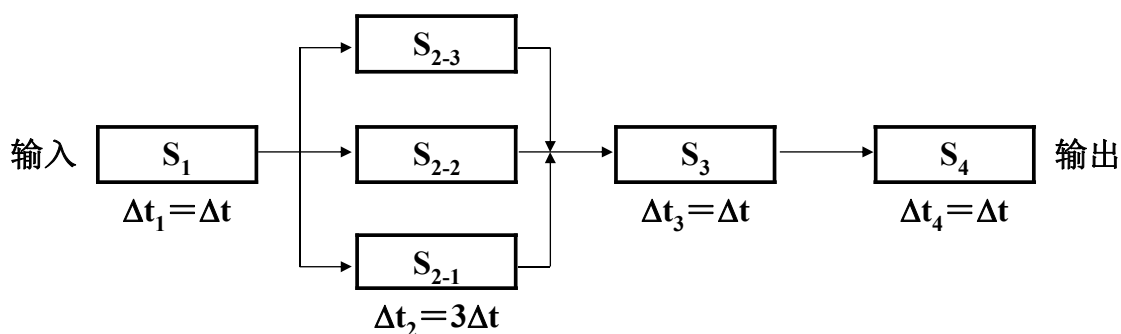
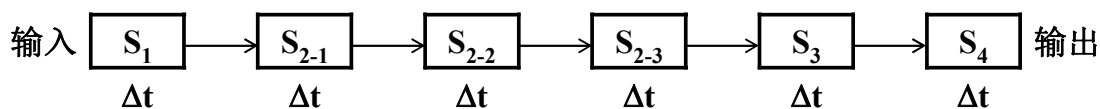
$$TP_{\max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

- 解决“瓶颈”



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

■ 加速比 (Speedup)

- 同一任务，不使用流水线所使用时间与使用流水线所用时间比

$$S = \frac{\text{顺序执行时间 } T_s}{\text{流水线执行时间 } T_k}$$

■ 各段执行时间相等，输入连续任务

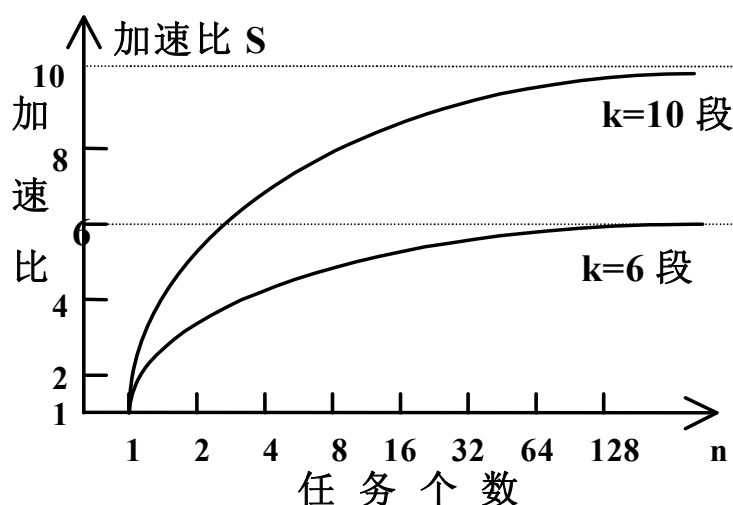
- 加速比：
$$S = \frac{k \cdot n \cdot \Delta t}{(k + n - 1) \Delta t} = \frac{k \cdot n}{k + n - 1}$$

- 最大加速比为：
$$S_{\max} = \lim_{n \rightarrow \infty} \frac{k \cdot n}{k + n - 1} = k$$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

- 各段执行时间不相等，输入连续任务情况下，实际加速比：

$$S = \frac{n \cdot \sum_{i=1}^k \Delta t_i}{\sum_{i=1}^k \Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \dots, \Delta t_k)}$$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

- 效率（Efficiency）- 流水线设备的利用率

$$E = \frac{n \text{ 个任务占用有效面积}}{\text{对应总面积}}$$

- 各流水段执行时间相等，输入n个连续任务，流水线的效率为：

$$E = \frac{k \cdot n \cdot \Delta t}{k \cdot (k + n - 1) \cdot \Delta t} = \frac{n}{k + n - 1}$$

- 流水线的最高效率为：

$$E_{\max} = \lim_{n \rightarrow \infty} \frac{n}{k + n - 1} = 1$$

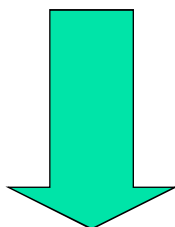


邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能指标

- 流水线的吞吐率、加速比与效率的关系：

$$TP = \frac{n}{(k + n - 1)\Delta t} \quad S = \frac{k \cdot n}{k + n - 1} \quad E = \frac{n}{k + n - 1}$$

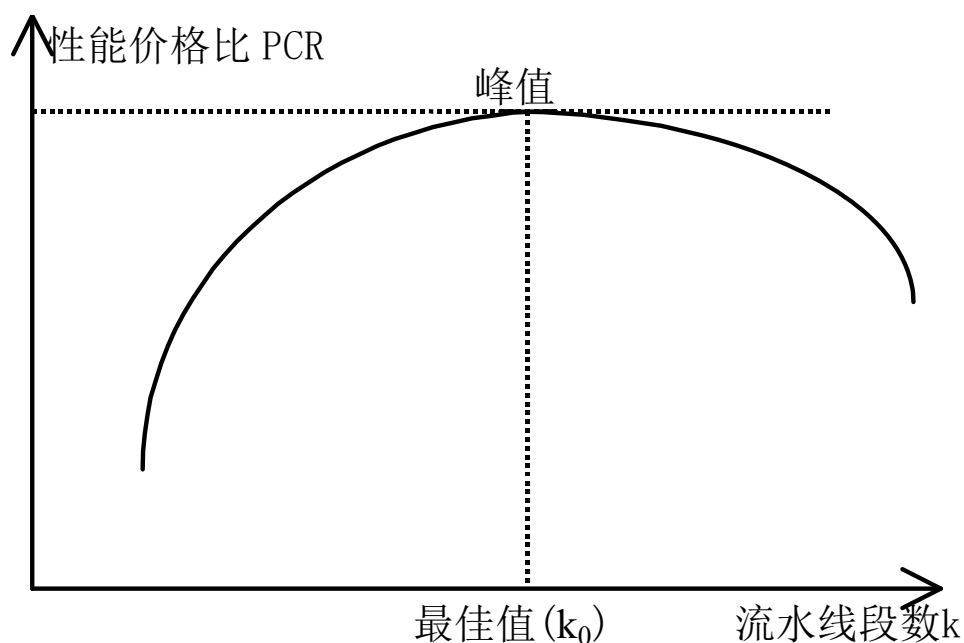


$$E = TP \cdot \Delta t, \quad S = k \cdot E$$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

最佳流水段数



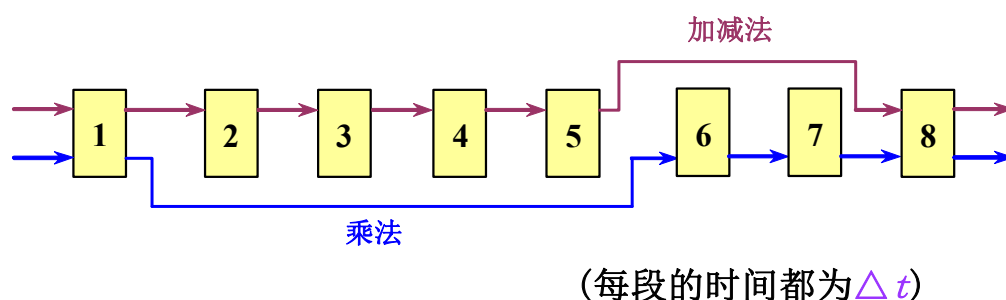
邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能分析举例

例3.1 设在下图所示的静态流水线上计算：

$$\prod_{i=1}^4 (A_i + B_i)$$

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。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

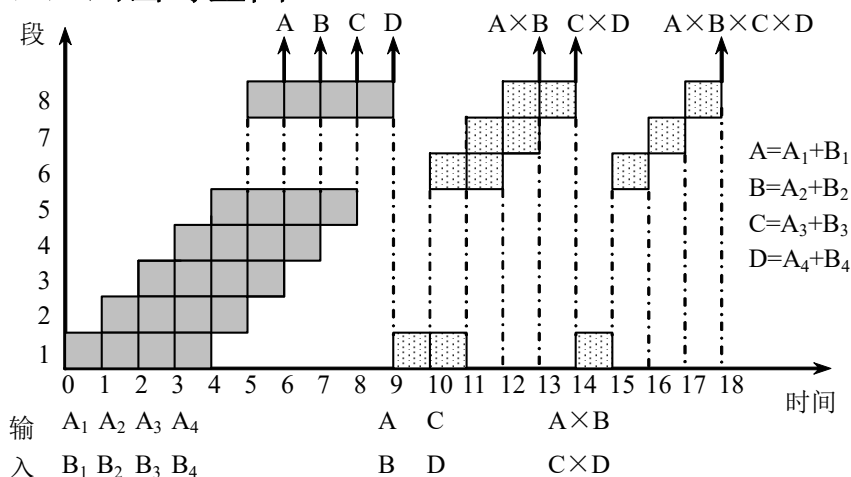
流水线的性能分析举例

解：（1）选择适合于流水线工作的算法

- 先计算 A_1+B_1 、 A_2+B_2 、 A_3+B_3 和 A_4+B_4 ；
- 再计算 $(A_1+B_1) \times (A_2+B_2)$ 和 $(A_3+B_3) \times (A_4+B_4)$ ；
- 然后求总的乘积结果。

（2）画出时空图

1, 2, 3, 4, 5, 8.
1, 6, 7.



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能分析举例

■ 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。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的性能分析举例

(3) 计算性能

- 在18个 Δt 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

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



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线设计中的若干问题

■ 瓶颈问题

- 理想情况下，流水线在工作时，其中的任务是同步地每一个时钟周期往前流动一段。
- 当流水线各段不均匀时，机器的时钟周期取决于瓶颈段的延迟时间。
- 在设计流水线时，要尽可能使各段时间相等。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线设计中的若干问题

■ 流水线的额外开销

- 流水寄存器延迟 - 流水寄存器需要建立时间和传输延迟
 - 建立时间：在触发写操作的时钟信号到达之前，寄存器输入必须保持稳定的时间。
 - 传输延迟：时钟信号到达后到寄存器输出可用的时间。
- 时钟偏移开销
 - 流水线中，时钟到达各流水寄存器的最大差值时间（时钟到达各流水寄存器的时间不是完全相同）



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线设计中的若干问题

■ 几个问题

- 流水线并不能减少（而且一般是增加）单条指令的执行时间，但却能提高吞吐率。
- 增加流水线的深度（段数）可以提高流水线的性能。
- 流水线的深度受限于流水线的额外开销。
- 当时钟周期小到与额外开销相同时，流水已没意义。因为这时在每一个时钟周期中已没有时间来做有用的工作。

■ 冲突问题

- 流水线设计中要解决的**重要问题之一**。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线的相关与冲突 Dependences/Hazards



北京邮电大学
Beijing University of Posts and Telecommunications

RISC MIPS 指令系统 PowerPC

RISC的结构特征

- 大多数指令在单周期内完成
- LOAD/STORE结构
- 硬布线控制逻辑
- 减少指令和寻址方式的种类
- 固定的指令格式
- 注重编译优化技术

MIPS

——卡内基梅隆大学（Carnegie Mellon）

RISC指令系统



典型的RISC特征

- 单一指令长度（4bytes）
- 较少的寻址方式（一般 ≤ 5 ）
- 无间接寻址
- 每条指令最多有一个存储器操作数
- **LOAD/STORE指令**不会与算术运算混在一起
- 整数寄存器有32个以上
- 浮点寄存器（如果有）有16个以上



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

RISC思想的精华

减少每条指令执行的周期数

- CPU性能公式 — **RISC的思想是减少CPI**

$$T_{CPU} = I_N \times CPI \times t = \frac{(I_N \times CPI)}{f}$$

■ 代价

- 同样的程序，在T相同的前提下，RISC的指令条数大约为CISC的1.3~1.4倍



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

RISC & CISC的发展

最近的RISC设计，
以PowerPC为代表，
不再是“纯”RISC了

最近的CISC设计，
以Pentium为代表，
不再是“纯”CISC了

随着芯片密度和硬件速度的提高，RISC也更“复杂”

8088: $CPI > 20$

80286: $CPI \approx 5.5$

80386: $CPI \approx 4$

80486: $CPI \approx 2$

Pentium: $CPI \approx RISC$

超标量处理机的CPI已经达到0.5，实际应该用IPC来衡量。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

RISC
MIPS 指令系统
PowerPC

MIPS指令系统结构

- MIPS64的一个子集，简称为MIPS。

- MIPS的寄存器

- 32个64位通用寄存器（GPRs）

- R0, R1, ..., R31

- 也被称为整数寄存器

- R0的值永远是0

32个64位通用寄存器.



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

- 32个64位浮点数寄存器（FPRs）

- F0, F1, ..., F31

- 用来存放32个单精度浮点数（32位），也可以用来存放32个双精度浮点数（64位）。

- 存储单精度浮点数（32位）时，只用到FPR的一半，其另一半没用。

- 一些特殊寄存器

- 它们可以与通用寄存器交换数据。

- 例如，浮点状态寄存器用来保存有关浮点操作结果的信息。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ MIPS的数据表示

■ 整数

- 字节（8位） 半字（16位）
- 字（32位） 双字（64位）
- 字节、半字或者字在装入64位寄存器时，用零扩展或者用符号位扩展来填充该寄存器的剩余部分。装入以后，它们将按照64位整数的方式进行运算。

扩展。

■ 浮点数

- 单精度浮点数（32位） 双精度浮点数（64位）



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ MIPS的数据寻址方式

- 立即数寻址与偏移量寻址，立即数字段和偏移量字段都是16位的。
- 寄存器间接寻址是通过把0作为偏移量来实现的
- 16位绝对寻址是通过把R0（其值永远为0）作为基址寄存器来完成的
- MIPS的存储器是按字节寻址的，地址为64位
- 所有存储器访问都必须都是边界对齐的

按字节寻址。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

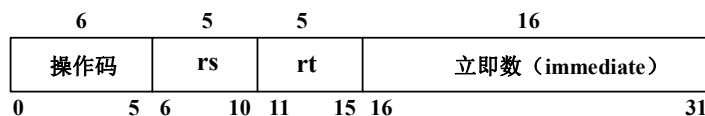
MIPS指令系统结构

MIPS 所有指令都是32位

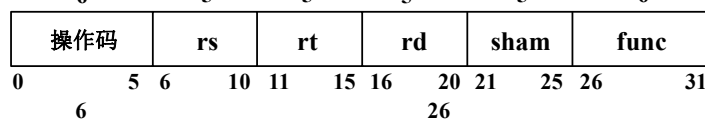
■ MIPS的指令格式

- 寻址方式编码到操作码中
- 所有的指令都是32位的
- 操作码占6位
- 3种基本指令格式:

■ I类指令



■ R类指令



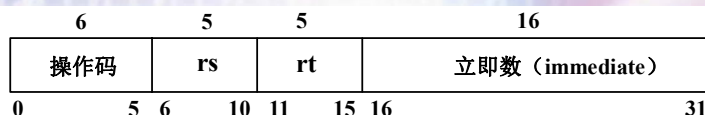
■ J类指令



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ I类指令



- 包括所有的load和store指令、立即数指令、分支指令、寄存器跳转指令、寄存器链接跳转指令。
- 立即数字段为16位，用于提供立即数或偏移量。

■ load指令

- 访存有效地址: $\text{Regs}[\text{rs}] + \text{immediate}$
- 从存储器取来的数据放入寄存器rt

举例
LWU R2, 40(R3) ;load word(32位无符号数)
LB R2, 30(R3) ;load byte(8位有符号数)

■ store指令

- 访存有效地址: $\text{Regs}[\text{rs}] + \text{immediate}$
- 要存入存储器的数据放在寄存器rt中

SW R4, 300(R5) ;store word(32位)
SD R4, 300(R5) ;store Dword(64位)

■ 立即数指令

- $\text{Regs}[\text{rt}] \leftarrow \text{Regs}[\text{rs}] \text{ op immediate}$

DSLL R1, R2, #5 ;Dword逻辑左移

■ 分支指令

- 转移目标地址: $\text{Regs}[\text{rs}] + \text{immediate}$
- 要判断的数据放在寄存器rt中

BNE R3, R4, name ;r3与r4不相等转移

■ 寄存器跳转、寄存器跳转并链接

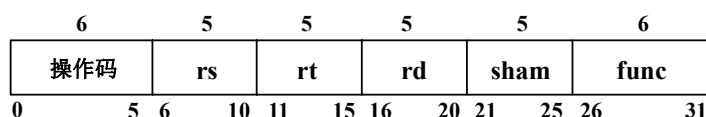
JALR R5, R7 ;寄存器跳转并链接转移到
Reg[r7],返回地址保存到r5



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ R类指令



- 包括ALU指令、专用寄存器读/写指令、move指令等。
- ALU指令
 - $\text{Regs}[\text{rd}] \leftarrow \text{Regs}[\text{rs}] \text{ func } \text{Regs}[\text{rt}]$
 - func为具体的运算操作编码
- 举例
 - **DADDU R1, R2, R3** ;双字无符号加, $r1=r2+r3$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ J类指令



- 包括跳转指令、跳转并链接指令、自陷指令、异常返回指令。
- 在这类指令中，指令字的低26位是偏移量，它与PC值相加形成跳转的地址。
- 举例
 - **J name** ;无条件转移, $PC_{27..0} \leftarrow \text{name} \ll 2$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ MIPS的操作

■ MIPS指令可以分为四大类

- load和store
- ALU操作
- 分支与跳转
- 浮点操作

■ RISC中R0的特殊处理

- R0的值永远（或可能）取0值，它可以用来合成一些常用的操作。
- 例如：
 - ~~DADD~~IU R1,R0,#100 //给寄存器R1装入常数100
 - ~~DADD~~ R1,R0,R2 //把寄存器R2中的数据传送到寄存器R1



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS指令系统结构

■ MIPS的浮点操作

- 由操作码指出操作数是单精度（SP）或双精度（DP）
 - 后缀S：表示操作数是单精度浮点数
 - 后缀D：表示是双精度浮点数
- 浮点操作
 - 包括加、减、乘、除，分别有单精度和双精度指令
- 浮点数比较指令
 - 根据比较结果设置浮点状态寄存器中的某一位，以便于后面的分支指令BC1T（若真则分支）或BC1F（若假则分支）测试该位，以决定是否进行分支。
- 举例
 - SUB(S)F2, F4, F8 ;单精度浮点减法, $f2=f4-f8$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心



流水线参考模型- MIPS 5段流水

	ALU	Load/Store	Branch
IF		取 指	
ID		译码, 读寄存器堆	
EX	执行	计算访存有效地址	计算目标地址, 设置条件码
MEM	(空操作)	访问存储器	若条件成立, 转移目标地址送PC
WB	计算结果写回寄存器堆	Load数据写回寄存器堆	(空操作)



现实及问题

SUB R1, R9, R6 ; R9 - R6 → R1
 ADD R3, R1, R2 ; R1 + R2 → R3
 ORI R5, R3, 0x1 ; R3 xor 0x1 → R5

指令之间在
R1, R3上
存在先写后
读相关

现
实

	ALU
IF	取 指
ID	译码, 读寄 存器堆
EX	执行
MEM	(空操作)
WB	计算结果写 回寄存器堆

理想时空图

SUB	IF	ID	EX	MEM	WB 写R1		
ADD		IF	ID 读R1	EX	MEM	WB 写R3	
ORI			IF	ID 读R3	EX	MEM	WB



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线冲突/冒险

实际的时空图

SUB	IF	ID	EX	ME	WB						
ADD		IF	stall	stall	stall	ID	EX	ME	WB		
ORI						IF	stall	stall	stall	ID	EX

流水线冲突/冒险(Hazard) – that prevent the next instruction in the instruction stream from executing during its designated clock cycle.

数据冲突/冒险(Data hazard)

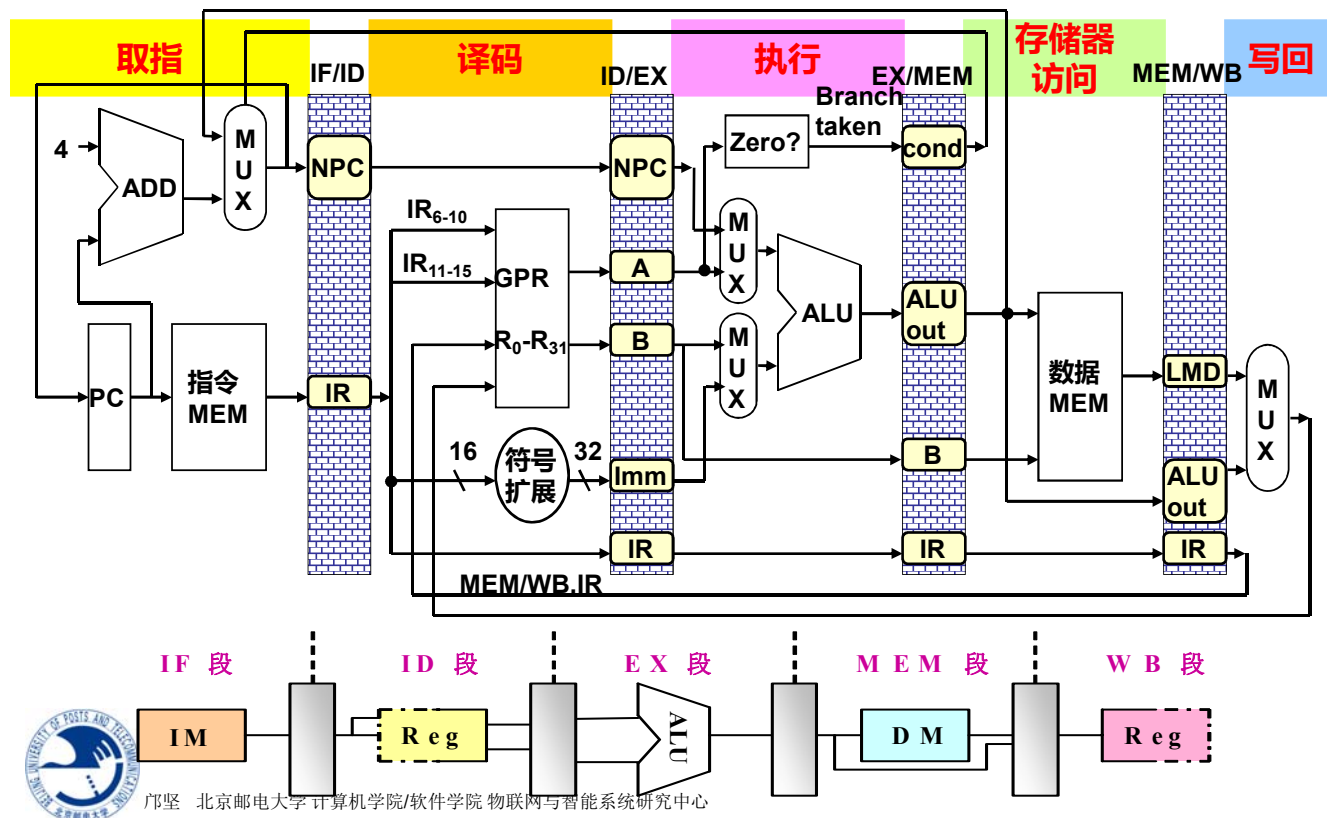
结构冲突/冒险(Structural hazard)

控制冲突/冒险(Control hazard)



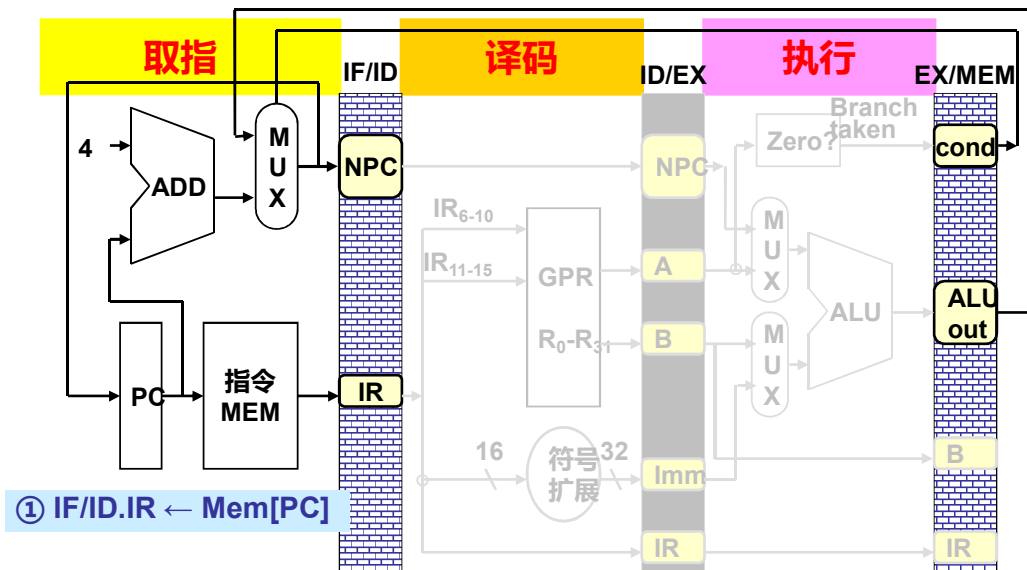
邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS流水线



MIPS流水线 - 取指(IF)

② IF/ID.NPC, PC \leftarrow (if (branch & EX/MEM.cond) {EX/MEM.ALUout} else {PC+4});

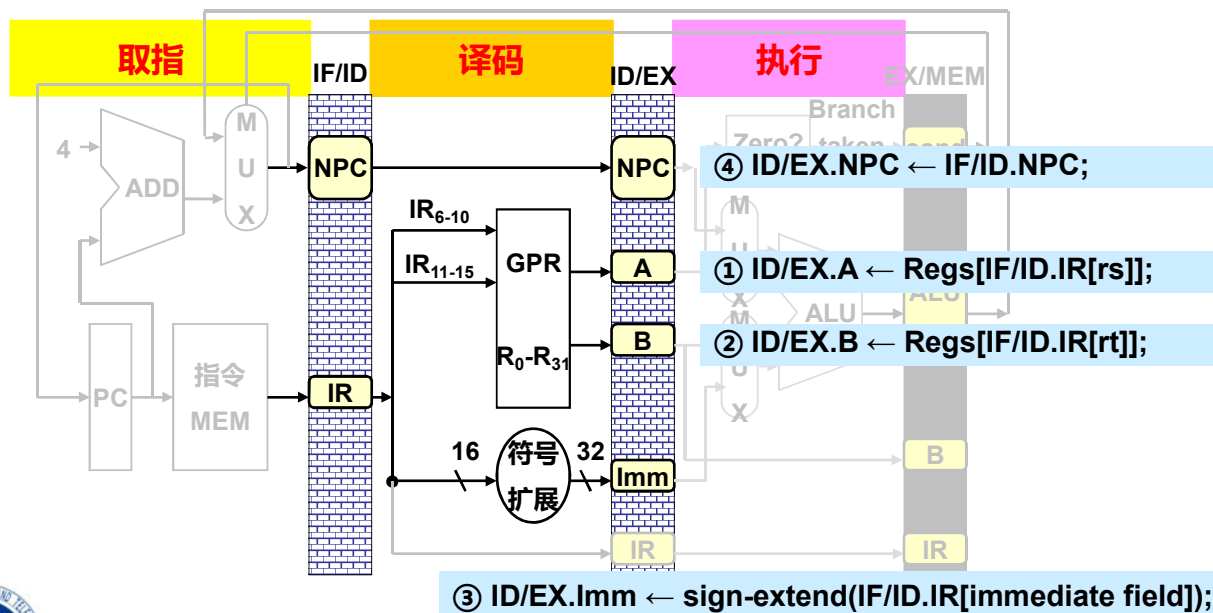


MIPS流水线 - 译码(ID)

0	5	6	10	11	15	16	20	21	31
操作码	rS		rT		rD				子功能码
操作码	rS		rT		立即数				

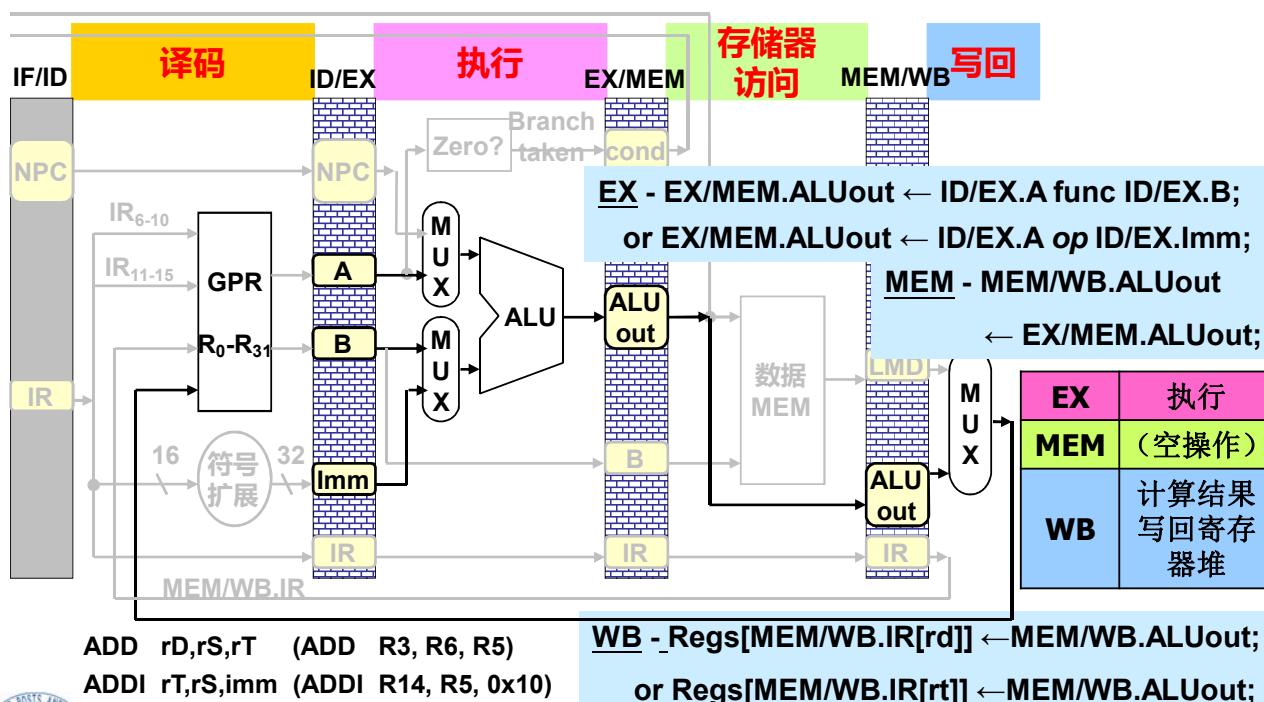
ADD rD,rS,rT (ADD R3, R6, R5)

ADDI rT,rS,imm (ADDI R14, R5, 0x10)



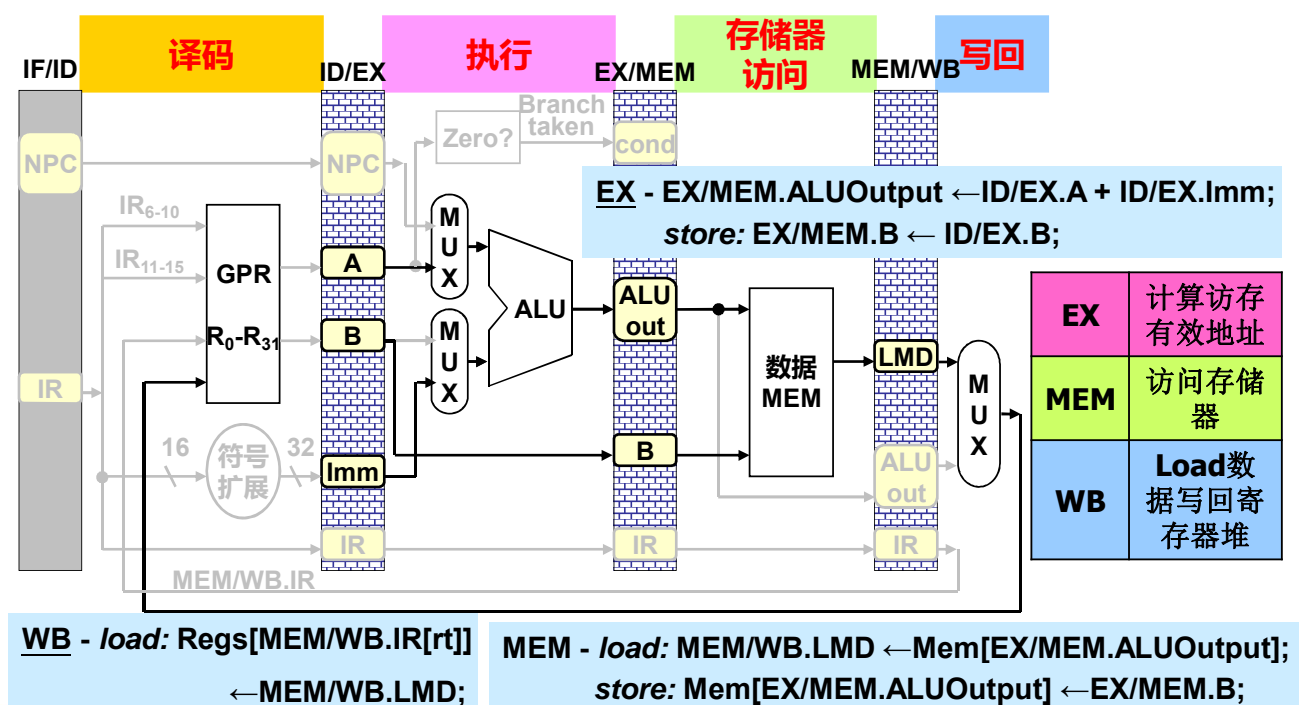
邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS流水线 - ALU指令执行/写回(EX/WB)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS流水线 – LOAD/STORE指令



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS流水线分析初步

- 要保证不会在同一时钟周期要求同一个功能段做两件不同的工作。
 - 例如，不能要求ALU同时做有效地址计算和算术运算。
- 避免IF段的访存（取指令）与MEM段的访存（读/写数据）发生冲突。
 - 可以采用分离的指令存储器和数据存储器；
 - 一般采用分离的指令Cache和数据Cache。
- ID段和WB段都要访问同一寄存器文件。
 - ID段：读 WB段：写
- 流水线为了能够每个时钟周期启动一条新的指令，就必须在每个时钟周期进行PC值的加4操作。但分支指令也可能改变PC的值，而且是在MEM段进行。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

冲突源于相关 - Dependence

- **相关**：两条指令之间存在某种依赖关系。
 - 如果两条指令相关，则它们就有可能不能在流水线中重叠执行或者只能部分重叠执行。
- 相关有3种类型
 - 数据相关（也称真数据相关）
 - 名相关
 - 控制相关



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

数据相关

- 数据相关
 - An instruction j is data dependence on instruction i if either of the following holds:
 - instruction i produce a result that may be used by instruction j , or
 - instruction j is data dependent on instruction k , and instruction k is data dependent on instruction i .
 - 对于两条指令 i （在前，下同）和 j （在后，下同），如果下述条件之一成立，则称指令 j 与指令 i 数据相关。
 - 指令 j 使用指令 i 产生的结果；
 - 指令 j 与指令 k 数据相关，而指令 k 又与指令 i 数据相关。

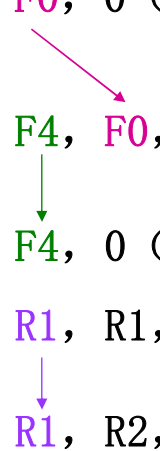


邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

数据相关

- 例如：下面这一段代码存在数据相关。

```
Loop:  L.D      F0, 0(R1)      // F0为数组元素
        ADD.D   F4, F0, F2     // 加上F2中的值
        S.D     F4, 0(R1)     // 保存结果
        DADDIU  R1, R1, -8     // 数组指针递减8个字节
        BNE     R1, R2, Loop   // 如果R1≠R2, 则分支
```



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

数据相关

- 指令之间数据供求关系上的依赖性，因流水机制所表现的冲突。
 - 数据相关具有传递性。
 - 数据相关反映了数据的流动关系，即如何从其产生者流动到其消费者。
- 当数据的流动是经过寄存器时，相关的检测比较直观和容易。
- 当数据的流动是经过存储器时，检测比较复杂。
 - 相同形式的地址其有效地址未必相同，如不同指令中的`10(R5)`。
 - 形式不同的地址其有效地址却可能相同。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

名(name)相关

- **名**：指令所访问的寄存器或存储器单元的名称。
- 如果两条指令使用相同的名，但是它们之间并没有数据流动，则称这两条指令存在**名相关**。
- 指令j与指令i之间的名相关有两种：
 - **反(Anti)相关**：如果指令j写的名与指令i读的名相同，则称指令i和j发生了反相关。

指令j写的名=指令i读的名

DIV.D F2, F6, F4

ADD.D F6, F0, F12

SUB.D F8, F6, F14

;DIV.D和ADD.D存在反相关



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

名(name)相关

- **输出(Output)相关**：如果指令j和指令i写相同的名，则称指令i和j发生了输出相关。
- 指令j写的名=指令i写的名
- 名相关的两条指令之间并没有数据的传送。
 - 如果一条指令中的名改变了，并不影响另外一条指令的执行。
 - **换名(Renaming)技术** - 通过改变指令中操作数的名来消除名相关。对于寄存器操作数进行换名称为寄存器换名(Register Renaming)。如：

DIV.D F2, F6, F4

ADD.D S, F0, F12

SUB.D F8, S, F14

;反相关消除

既可以用编译器静态实现，也可以用硬件动态完成。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

数据冲突类型

- RAW(read after write) – 写后读
 - j tries to read a source before i write it
 - be the most common case
- WAW(write after write) – 写后写
 - j tries to write a operand before it is written by i
 - Example?
- WAR(write after read) – 读后写
 - j tries to write a destination before it is read by i
 - Example?



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制相关

- 控制(Control)相关是指由分支指令引起的
相关。
 - 为了保证程序应有的执行顺序，必须严格按控制相关确定的顺序执行。
- 典型的程序结构是 “if-then”结构，如：

```
if p1 {  
    S1;  
};  
S;  
if p2 {  
    S2;  
};
```



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制相关

- 控制相关带来了以下两个限制：
 - 与一条分支指令控制相关的指令不能被移到该分支之前，否则这些指令就不受该分支控制。
 - 对上例，**then** 部分中的指令不能移到**if**语句之前。
 - 如果一条指令与某分支指令不存在控制相关，就不能把该指令移到该分支之后。
 - 对上例子，不能把**S**移到**if**语句的**then**部分中。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Control Dependences

1. An instruction that is dependent on a branch cannot be moved *before* the branch so that its execution *is no longer controlled* by the branch.
2. An instruction that is not control dependent on a branch cannot be moved *after* the branch so that its execution *is controlled* by the branch.



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线冲突

- **流水线冲突(Pipeline Hazard)的3种类型：**
 - **结构冲突：**因硬件资源满足不了指令重叠执行的要求而发生的冲突。
 - **数据冲突：**当指令在流水线中重叠执行时，因需要用到前面指令的执行结果而发生的冲突。
 - **控制冲突：**流水线遇到分支指令和其他会改变PC的指令所引起的冲突。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线冲突

- **带来的问题：**

流水线可能会出现停顿，从而降低流水线的效率和实际的加速比甚至导致错误的执行结果
- **约定：**

当一条指令被暂停时，在该暂停指令之后流出的所有指令都要被暂停，而在该暂停指令之前流出的指令则继续进行（否则就永远无法消除冲突）

。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

结构冲突

- Arise from resource conflicts when the HW cannot support all possible combination of instructions simultaneously in **overlapped execution**.
- 某些指令组合在流水线重叠执行过程中，如果硬件资源满足不了指令重叠执行的要求，便会产生资源冲突
 - 资源份数不够 - 在同一个时钟周期内争用同一个功能部件
 - 功能部件不是完全流水



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

结构冲突举例 – 访存冲突

有访存冲突

I(Load)	IF	ID	EX	ME	WB					
I+1		IF	ID	EX	ME	WB				
I+2			IF	ID	EX	ME	WB			
I+3				IF	ID	EX	ME	WB		

- 方法1：插入暂停周期

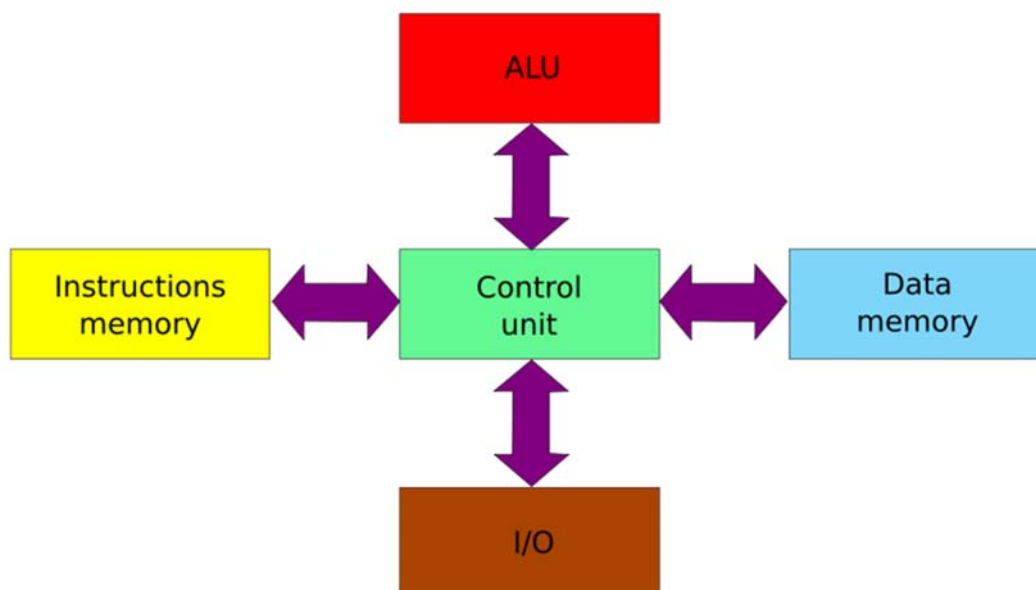
I(Load)	IF	ID	EX	ME	WB					
I+1		IF	ID	EX	ME	WB				
I+2			IF	ID	EX	ME	WB			
I+3				stall	IF	ID	EX	ME	WB	

- 方法2：指令与数据的分离（分时）存取处理：
 - 分离cache、双端口RAM、Harvard architecture等



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Harvard architecture



wikipedia



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

数据冲突

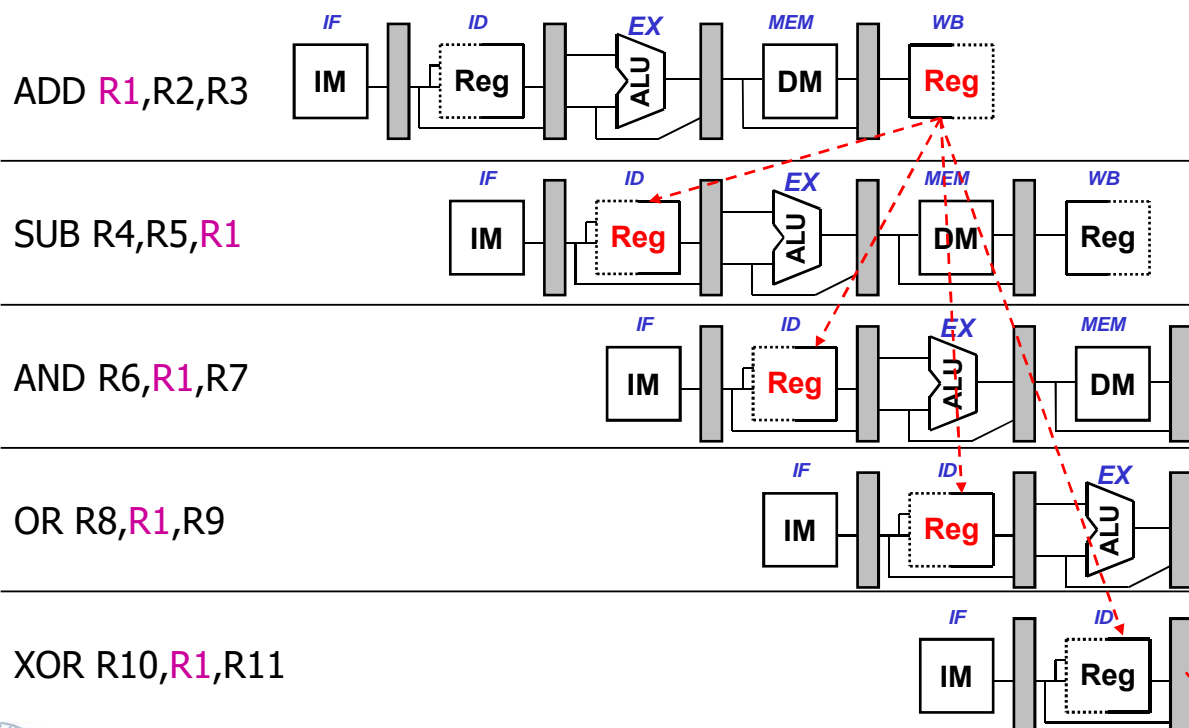
- 当相关的指令靠得足够近时，它们在流水线中的重叠执行或者重新排序会改变指令读/写操作数的顺序，使之不同于它们非流水实现时的顺序，则发生了**数据冲突**。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

典型的数据冲突

写后读



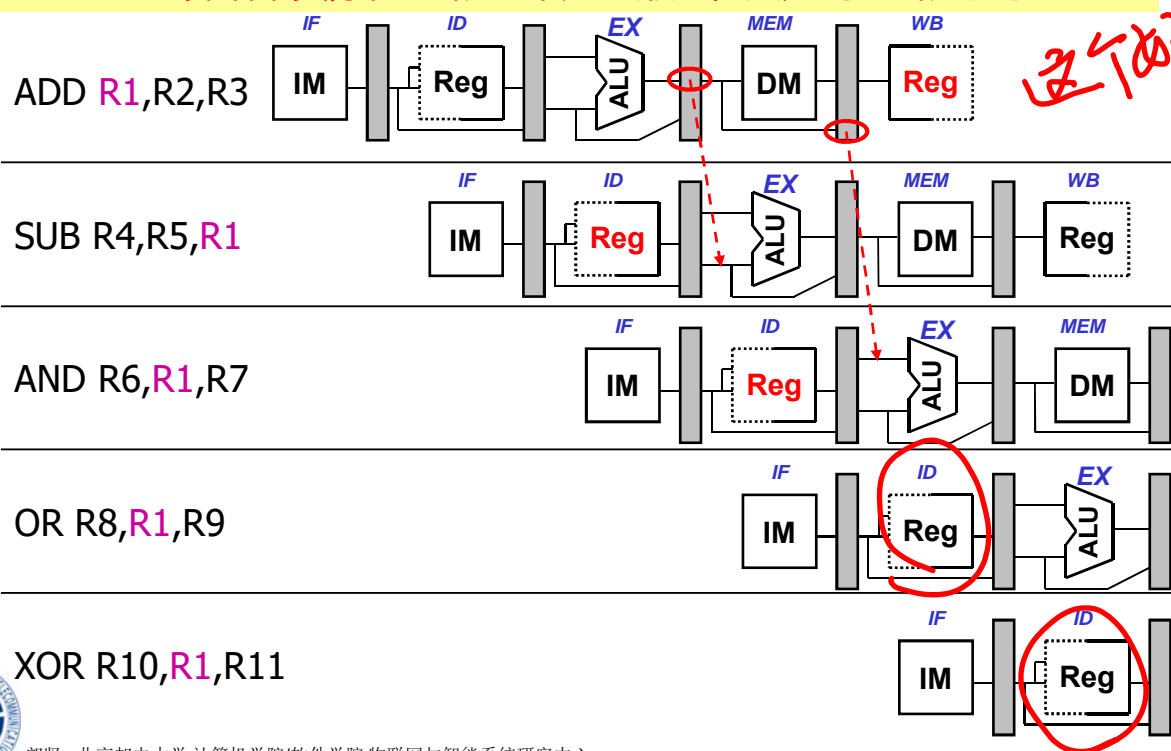
邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

★ 对于寄存器堆的操作，采用前半周期写入，后半周期读出

定向传送(Forwarding)技术

定向传送

★ ADD的运算结果实际在EX段的末尾已形成，只是到WB段才写入R1

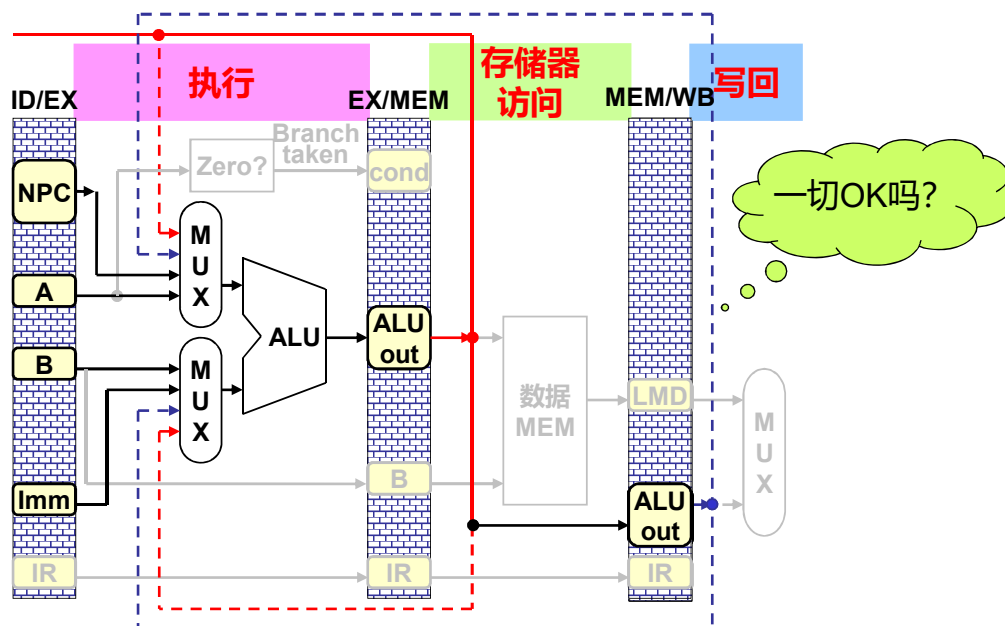


这个例子很经典
前半周期读
后半周期写



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水线针对定向传送的改进



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

定向传送(Forwarding)技术

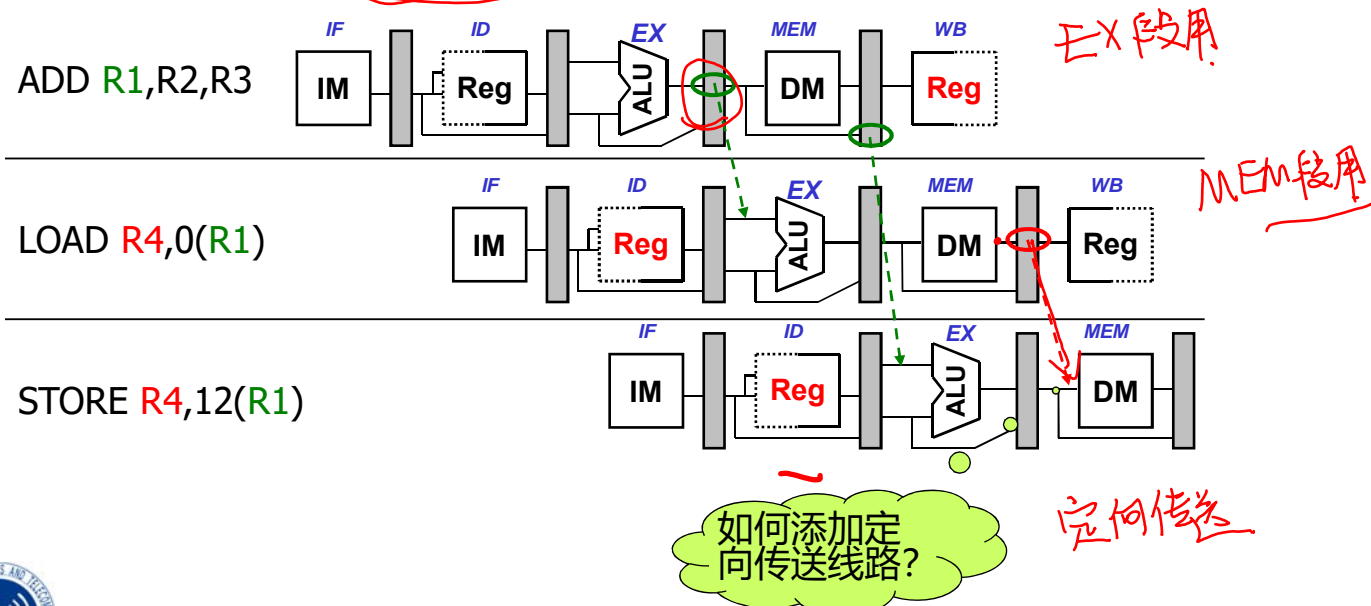
- 定向技术也称为旁路(Bypass)或短路(short-circuiting)
- **关键思想**: 在某条指令产生计算结果之前, 其他指令并不真正立即需要该计算结果, 如果能够将该计算结果从其产生的地方直接送到其他指令需要它的地方, 那么就可以避免停顿。
- 当定向硬件检测到前面某条指令的结果寄存器就是当前指令的源寄存器时, 控制逻辑会将前面那条指令的结果直接从其产生的地方定向到当前指令所需的位置。
- 结果数据不仅可以从某一功能部件的输出定向到其自身的输入, 而且还可以定向到其他功能部件的输入。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

定向传送 – LOAD/STORE操作

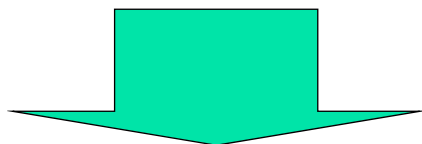
★ 所LOAD的数据在MEM段末尾已可用，同样可以定向传送到MEM入口。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Pipeline interlock - 互锁

LOAD R1, 0(R2)	IF	ID	EX	ME	WB			
ADD R4, R1, R7		IF	ID	EX	ME	WB		
SUB R6, R1, R8			IF	ID	EX	ME	WB	
OR R3, R1, R9				IF	ID	EX	ME	WB



LOAD R1, 0(R2)	IF	ID	EX	ME	WB			
ADD R4, R1, R7		IF	ID	stall	EX	ME	WB	
SUB R6, R1, R8			IF	stall	ID	EX	ME	WB
OR R3, R1, R9				stall	IF	ID	EX	ME

互锁
(默认有定向)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

依靠编译器解决数据冲突(静态调度)

- 让编译器重新组织指令顺序来消除冲突，这种技术称为指令调度或流水线调度。

重新组织

- 例如：采用典型的代码生成方法，表达式 $A=B+C$ 的代码会导致暂停

LD Rb, B	IF	ID	EX	MEM	WB				
LD Rc, C		IF	ID	EX	MEM	WB			
DADD Ra, Rb, Rc			IF	ID	stall	EX	MEM	WB	
SD Ra, A				IF	stall	ID	EX	MEM	WB



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

画图

依靠编译器解决数据冲突

- 举例：请为下列表达式生成没有暂停的指令序列：

$A=B+C$;

IF ID EX MEM. WB

$D=E-F$;

IF ID EX

MEM. WB

IF ID

EX MEM

调度前的代码	调度后的代码
LD Rb, B	LD Rb, B
LD Rc, C	LD Rc, C
DADD Ra, Rb, Rc	LD Re, E ✓
SD Ra, A	DADD Ra, Rb, Rc
LD Re, E	LD Rf, F
LD Rf, F	SD Ra, A
DSUB Rd, Re, Rf	DSUB Rd, Re, Rf
SD Rd, D	SD Rd, D



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Static scheduling

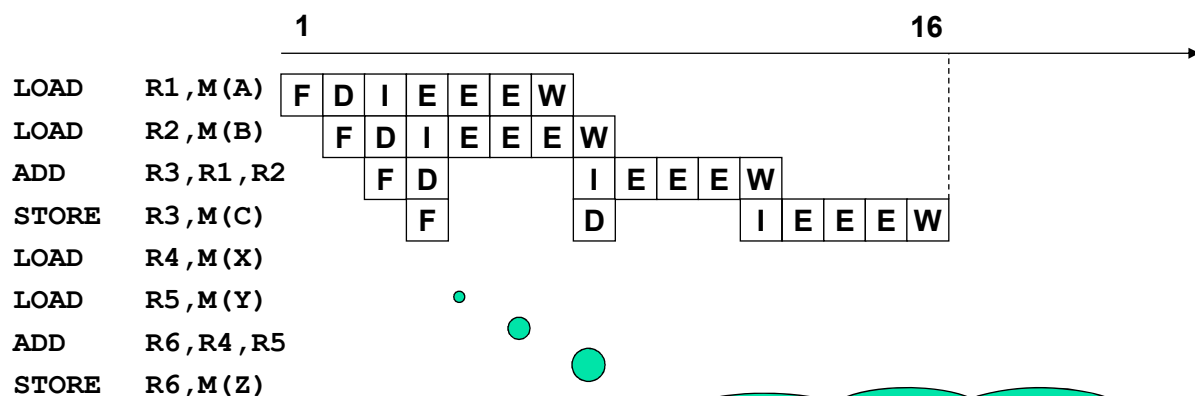


- 执行段(E)为多功能部件，含LOAD/STORE
- 3个执行段不能停顿
- 发射段(I)能预约资源，提供互锁，允许与写回段(W)重叠
- 每个功能部件均有自己的写回部件，只要不向同一目标写回，则无冲突



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Static scheduling



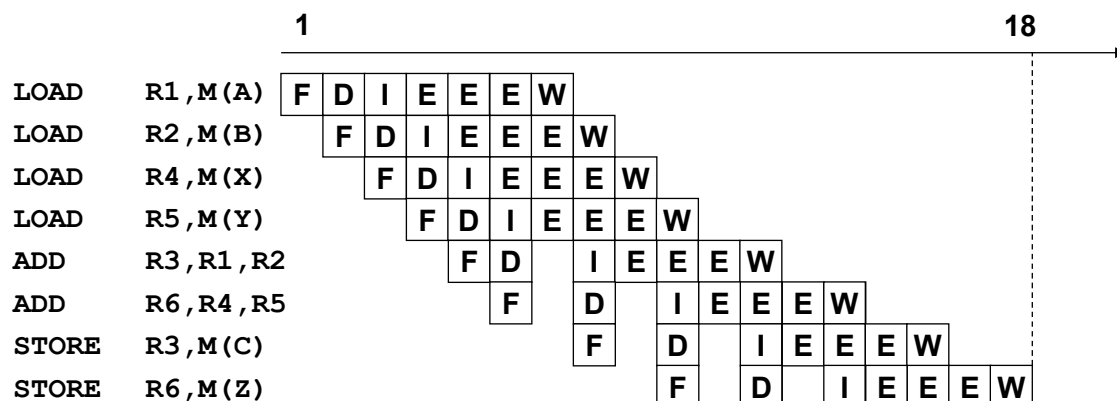
后续4条指令的时空图？

共需要多少个周期？



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Static scheduling



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突 - Control hazard

- 执行分支指令的结果有两种
 - **分支成功**: PC值改变为分支转移的目标地址。在条件判定和转移地址计算都完成后, 才改变PC值。
 - **不成功或者失败**: PC的值保持正常递增, 指向顺序的下一条指令。
- 处理分支指令最简单的方法 - “冻结”或者“排空”流水线。

BRANCH	IF	ID	EX	ME	WB				
i+1		IF	stall	stall	IF	ID	EX	ME	WB

Freeze

局部相关 - 如数据相关, 仅仅影响到本指令的附近少数指令;

全局相关 - 如控制相关, 其影响的范围要大得多, 流水线性能损失更大。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

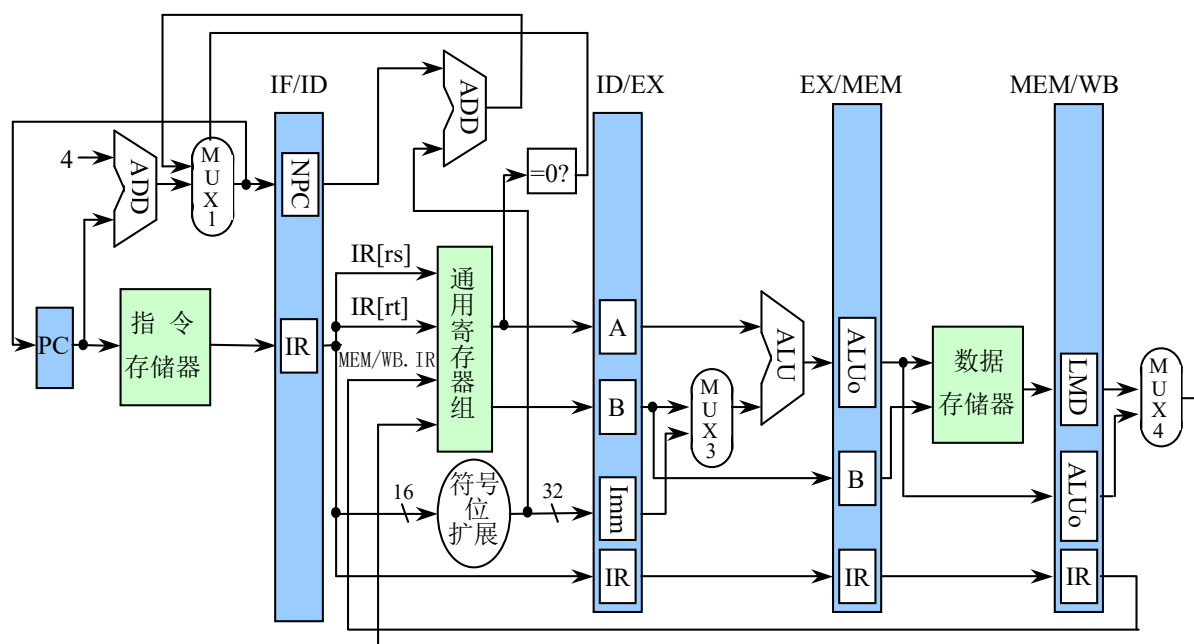
控制冲突

- 把由分支指令引起的延迟称为**分支延迟**。
- 分支指令在目标代码中出现的频度
 - 若每3~4条指令就有一条是分支指令。
 - 假设：分支指令出现的频度是30%，流水线理想CPI=1，
 - 那么：流水线的实际 CPI = ? 。
- 可采取两种措施来减少分支延迟
 - 在流水线中尽早判断出分支转移是否成功；
 - 尽早计算出分支目标地址。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

ID zero test for branch



BRANCH	IF	ID	EX	ME	WB				
i+1		<i>stall</i>	IF	ID	EX	ME	WB		



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

- 3种通过软件（编译器）来减少分支延迟的方法：
 - 预测分支失败-Treat every branch as not taken
 - 预测分支成功-Treat every branch as taken
 - 延迟分支- Delayed branch



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

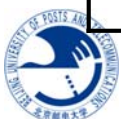
控制冲突

- Treat every branch as not taken, or
- Treat every branch as taken

预测分支失败

Untaken Branch	IF	ID	EX	ME	WB				
i+1		IF	ID	EX	ME	WB			
i+2			IF	ID	EX	ME	WB		
i+3				IF	ID	EX	ME	WB	

Taken Branch	IF	ID	EX	ME	WB				
i+1		IF							
target			IF	ID	EX	ME	WB		
t+1				IF	ID	EX	ME	WB	



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

■ Delayed branch(延迟分支)

branch inst

sequential successor

branch target if taken

Branch delay slot

- Sequential successor instruction is executed whether or not the branch is taken



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

Untaken Branch	IF	ID	EX	ME	WB				
Branch delay		IF	ID	EX	ME	WB			
i+2			IF	ID	EX	ME	WB		
i+3				IF	ID	EX	ME	WB	

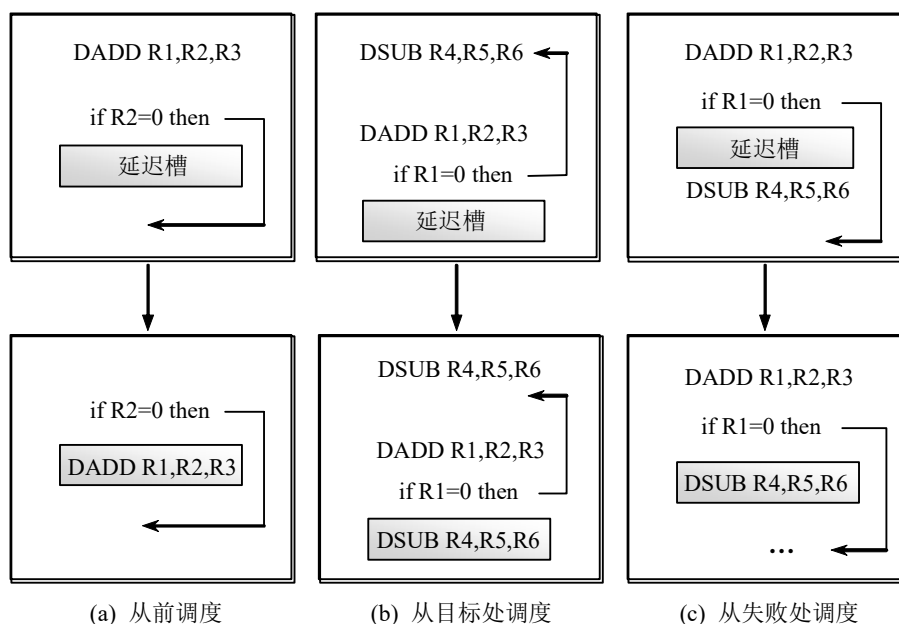
Taken Branch	IF	ID	EX	ME	WB				
Branch delay		IF	ID	EX	ME	WB			
target			IF	ID	EX	ME	WB		
t+1				IF	ID	EX	ME	WB	



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

- 在延迟槽中放入有用的指令由编译器完成。能否带来好处取决于编译器能否把有用的指令调度到延迟槽中。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

- 三种方法的要求及效果

调度策略	对调度的要求	什么情况下起作用
从前调度	被调度的指令 <u>必须与分支无关</u>	任何情况
从目标处调度	必须保证在分支失败时 <u>执行被调度的指令不会导致错误</u> 。 <u>有可能需要复制指令</u>	分支成功时 (但由于复制指令, 有可能会增大程序空间)
从失败处调度	必须保证在分支成功时 <u>执行被调度的指令不会导致错误</u> <u>不会导致错误</u>	分支失败时



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

- 分支延迟受到两个方面的限制：
 - 可以被放入延迟槽中的指令要满足一定的条件
 - 编译器预测分支转移方向的能力
- 进一步改进：分支取消机制（取消分支）
 - 当分支的实际执行方向和事先所预测的一样时，执行分支延迟槽中的指令，否则就将分支延迟槽中的指令转化成一个空操作。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

控制冲突

分支失败	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	idle	idle	idle	idle			
	指令 i+2			IF	ID	EX	MEM	WB		
	指令 i+3				IF	ID	EX	MEM	WB	
	指令 i+4					IF	ID	EX	MEM	WB

分支成功	分支指令i	IF	ID	EX	MEM	WB				
	延迟槽指令 i+1		IF	ID	EX	MEM	WB			
	分支目标指令j			IF	ID	EX	MEM	WB		
	分支目标指令j+1				IF	ID	EX	MEM	WB	
	分支目标指令j+2					IF	ID	EX	MEM	WB

预测分支成功的情况下，分支取消机制的执行情况



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心



Branch process in PowerPC

Predict By Opcode

Instruction-level Parallelism

- ILP(of a program) – a measure of the average number of instructions in a program that a processor *might* be able to execute at the same time
 - Mostly determined by the number of data/control dependencies in relation to the number of other instructions

LOAD **R1**, M(A)

ADD **R3**, **R1**, R2

STORE **R3**, M(B)

LOAD R1, M(A)

ADD R3, R4, R5

STORE R2, M(B)



Machine Parallelism

- Machine parallelism (of a processor) – a measure of the ability of the processor to take advantage of the ILP of the program
 - Determined by the number of instructions that can be fetched and executed at the same time

To achieve high performance, need both ILP and machine parallelism - HW+SW



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

指令级并行概念

- 流水线处理机的实际CPI
 - 理想流水线的CPI加上各类停顿的时钟周期数：
$$CPI_{\text{流水线}} = CPI_{\text{理想}} + \text{停顿}_{\text{结构冲突}} + \text{停顿}_{\text{数据冲突}} + \text{停顿}_{\text{控制冲突}}$$
 - 理想CPI是衡量流水线最高性能的一个指标。
- 基本程序块(Basic Block)
 - 基本程序块：一段除了入口和出口以外不包含其他分支的线性代码段。
 - 程序平均每4~7条指令就会有一个分支。
 - 在基本程序块中能开发出的并行性有限，必须跨越多个基本块开发ILP



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

指令级并行概念

- 循环级并行性(Loop-Level Parallelism): 开发循环不同迭代之间存在的并行性。是指令级并行研究的重点之一。例如:

```
for (i=1; i<=500; i=i+1)
{ a[i]=a[i]+s; };
```

- 在每一次循环的内部, 没有任何的并行性。
- 但每一次循环都可以与其他的循环重叠并行执行;



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

- 相关与流水线冲突
 - 相关是程序固有的一种属性, 它反映了程序中指令之间的相互依赖关系。
 - 具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿, 则是流水线的属性。
- 可以从两个方面来解决相关问题:
 - 保持相关, 但避免发生冲突。
 - 如静态指令调度 – 软件方法; 动态调度? - 硬件方法
 - 通过代码变换, 消除相关。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

- 可能的改变方式是：
 - 程序顺序：由源程序确定的在完全串行方式下指令的执行顺序。
 - 并不需要在所有相关的地方都保持程序顺序；
 - 只有在可能会导致错误的情况下，才保持程序顺序。
 - 控制相关并不是一个必须严格保持的关键属性
 - 其含义是：在保持程序正确性的前提下，可以执行本不该执行的指令



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

- 然而，对于正确地执行程序来说，必须保持的最关键的两个属性是：**异常行为**和**数据流**。
- **保持异常行为(Exception Behavior)**是指：无论怎么改变指令的执行顺序，都不能改变程序中异常的发生情况。
 - 即原来程序中是怎么发生的，改变执行顺序后还是怎么发生。
 - 经常被弱化为：指令执行顺序的改变不能导致程序中发生新的异常。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

□ 例如：

```
DADDU    R2, R3, R4
BEQZ     R2, L1
LW       R1, 0(R2)
```

L1 :

- 其中，如果只考虑LW指令与前两条指令的数据相关性，LW指令可以移到BEQZ指令之前，但会因为没有遵循控制相关，则产生异常(Memory Protection Exception)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

- **数据流(Data Flow)**：指数据值从其产生者指令到其消费者指令的实际流动。
 - 分支指令使得数据流具有动态性，因为它使得给定指令的数据可以有多个来源。
 - 仅仅保持数据相关性是不够的，只有再加上保持控制顺序，才能够保持程序顺序。

□ 举例：

```
DADDU    R1, R2, R3
BEQZ     R4, L1
DSUBU    R1, R5, R6
```

L1 : ...

```
OR       R7, R1, R8
```

- 其中，OR指令中R1值的生产者可能为DADDU/DSUBU二者之一，由分支指令BEQZ决定



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

相关与指令级并行

- 但有时，不遵守控制相关既不影响异常行为，也不改变数据流。
 - 可以大胆地进行指令调度，把失败分支中的指令调度到分支指令之前。

□ 举例：

DADDU	R1, R2, R3	如果已知R4在Skipnext之后不再使用，且DSUBU不会产生异常，该指令可被移到分支指令之前。
BEQZ	R12, Skipnext	
DSUBU	R4, R5, R6	
DADDU	R5, R4, R9	
Skipnext: OR	R7, R8, R9	



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

指令的动态调度

- 动态调度(Dynamic Scheduling)
 - 在程序的执行过程中，依靠专门硬件对代码进行调度，减少数据相关导致的停顿。
- 优点：减少数据相关导致的停顿。
 - 能够处理一些在编译时情况不明的相关（比如涉及存储器访问的相关），并简化了编译器；
 - 能够使本来是面向某一流水线优化编译的代码在其他的流水线（动态调度）上也能高效地执行。
- 代价：硬件复杂性的显著增加。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态调度的基本思想

- 到目前为止我们所使用5段流水线的最大的局限性 - 指令必须按序流出和执行

DIV.D F4, F0, F2

SUB.D F10, F4, F6

ADD.D F12, F6, F14

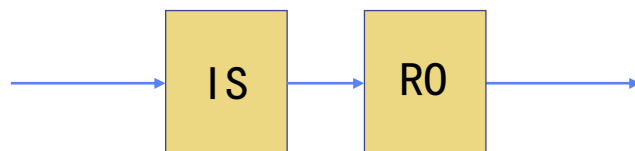
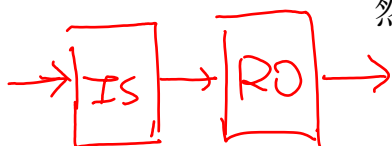
- SUB.D指令与DIV.D指令关于F4相关，导致流水线停顿。
- ADD.D指令与流水线中的任何指令都没有相关，但也因此受阻。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态调度的基本思想

- 在之前的基本流水线中：
 - ID段既检测结构冲突，也检测数据冲突
 - 一旦一条指令受阻，其后的指令都将停顿
- 进一步的解决办法 – Out-of-Order
 - 我们将5段流水线的译码阶段再分为两个阶段：
 - 流出(Issue, IS): 指令译码，检查是否存在结构(资源)冲突 - in-order issue
 - 读操作数(Read Operands, RO): 等待数据冲突消失，然后读操作数 - out of order execution



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

检测结构冲突 检测数据冲突

Dynamic Scheduling

- All instructions pass through the **issue stage in order(in-order issue)**; however, they can be stalled or bypass each other in the second stage (read operands) and thus enter **execution out of order**.



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态调度的基本思想

- 在前述5段流水线中，是不会发生WAR冲突和WAW冲突的。但乱序执行就使得它们可能发生了。

存在反相关 {	DIV. D	F10, F0, F2	} 存在输出相关
	SUB. D	F10, F4, F6	
	ADD. D	F6, F8, F14	

- 动态调度的流水线要求支持多条指令同时处于执行当中
- 典型的动态调度算法
 - 计分牌(**Scoreboard**)方法，最先在CDC 6600大型机中采用，采用集中控制方法。
 - **Tomasulo**方法，又称为公共数据总线(CDB: Common Data Bus)法，或令牌法等。采用分散控制的办法处理数据相关。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

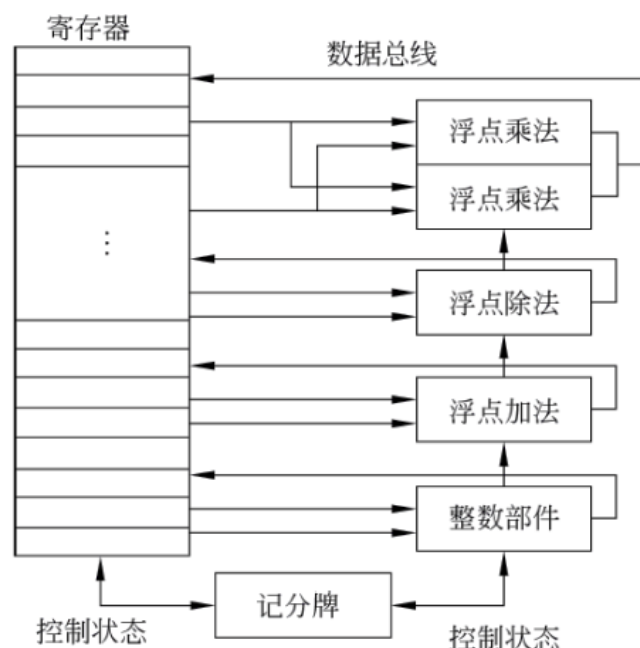
- 记分牌(Scoreboard)方法 - 起源于最早采用此功能的CDC 6600计算机。
 - CDC 6600具有16个独立的功能部件：4个浮点部件，5个访存部件，7个整数操作部件。
 - 记分牌硬件维护着三张表，分别用于记录指令的执行状态、功能部件状态，寄存器状态以及数据相关关系等。
 - 把ID段分成两个段：流出和读操作数；允许多条指令同时处于执行段。记分牌全面负责和管理这指令的流出和执行，也包括检测所有的冲突。
 - 在没有结构(资源)冲突时，尽可能早地执行没有数据冲突的指令。如果某条指令被暂停，而后面的指令与流水线中正在执行或被暂停的指令都不相关，那么这些指令就可以跨越它们，继续流出和执行下去(out of order)。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

- 一个具有记分牌的MIPS处理器模型
 - 记分牌仅用于浮点部件
 - 2个乘法器、1个加法器、1个除法器
和1个整数部件，
整数部件用来处理所有的存储器访问、分支处理和整数操作。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

- 处理步骤：流出，读操作数，执行，写结果

- 流出：

- 如果有结构相关或 WAW 冲突，指令不流出。
- 如果当前流出指令所需的功能部件空闲，并且所有其他正在执行的指令的目的寄存器与该指令的不同，记分牌就向功能部件流出该指令，并修改记分牌内部的记录表。
- 即：如果存在结构相关或 WAW 冲突，指令不流出 - 解决了 WAW 冲突。当然也阻止了后面指令的流出。

- 读操作数：

- 记分牌监测源操作数的可用性，一旦数据可用，它就通知功能部件从寄存器读出源操作数并开始执行（动态地解决了 RAW 冲突），否则等待。
- 数据可用：如果所有前面已流出且还在执行的指令都不对该寄存器进行写操作，则该寄存器的数据是可用的。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

- 执行：

- 取到操作数后，功能部件开始执行。当产生出结果后，就通知记分牌它已经完成执行。

- 写结果：

- 记分牌一旦知道执行部件完成了执行，就检测是否存在 WAR 冲突。
 - 如果不存在 WAR 冲突，或者原有的 WAR 冲突已消失，记分牌就通知功能部件把结果写入目的寄存器，并释放该指令使用的所有资源。
 - 如果检测到 WAR 冲突，就不允许该指令将结果写到目的寄存器，这发生在以下情况：
 - 前面的某条指令（按顺序流出）还没有读取操作数，而且其某个源操作数寄存器与本指令的目的寄存器相同。在这种情况下，记分牌必须等待，直到该冲突消失。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

■ 记分牌结构

- 指令状态表：记录正在执行的各条指令已经进入到了哪一段。

指令	指令状态表			
	流出	读操作数	执行	写结果

- 功能部件状态表：记录各个功能部件的状态。每个功能部件一项。

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk

- Busy: 忙标志，功能部件是否忙，初值为no；
- Op: 该功能部件正在执行或将要执行的操作；
- Fi: 目的寄存器编号；
- Fj/Fk: 源寄存器编号；
- Qj/Qk: 指出向源寄存器Fj/Fk写数据的功能部件（即Fj/Fk中的数据将由它们产生）；
- Rj/Rk: 标志位，yes表示Fi/Fk中的操作数就绪且还未被取走，否则就被置为no。

源寄存器且还未被取走



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

■ 记分牌结构

- 结果寄存器状态表Result: 每个寄存器在该表中有一项，用于指出哪个功能部件（编号）将把结果写入该寄存器。如果当前正在运行的指令都不以它为目的寄存器，则其相应项置为no。
Result各项的初值为no(全零)。

结果寄存器状态表									
F0	F2	F4	F6	F8	F10	...	F30		

■ 功能部件

- Integer, Multi1, Multi2, Add, Divide



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard举例

已执行完，结果已写入F6

已执行完，结果还未写F2

与F2有RAW相关，在流出段等待

与F0有RAW相关，在流出段等待

与SUB.D结构相关，不能流出

指令	指令状态表			
	流出	读操作数	执行	写结果
L.D F6, 34(R2)	✓	✓	✓	✓
L.D F2, 45(R3)	✓	✓	✓	
MULT.D F0, F2, F4	✓			
SUB.D F8, F6, F2	✓			
DIV.D F10, F0, F6	✓			
ADD.D F6, F8, F2				

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2	R3				no	
Mult1	yes	MULT.D	F0	F2	F4	Integer		no	yes
Mult2	no								
Add	yes	SUB.D	F8	F6	F2		Integer	yes	no
Divide	yes	DIV.D	F10	F0	F6	Mult1		no	yes

部件名称	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	...	F30
Mult1		Integer						
Add								
Divide								



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard举例

- 假设浮点流水线中各部件的延迟：
 - 加法2个时钟周期
 - 乘法10个时钟周期
 - 除法40个时钟周期
- 当MULT.D准备写结果之前的记分牌状态：

已完成

还未写结果

已完成，F2相关已解除，加法只需2T

与F0有RAW相关，不能读操作数

与F6有WAR相关，阻塞在执行段

指令	指令状态表			
	流出	读操作数	执行	写结果
L.D F6, 34(R2)	✓	✓	✓	✓
L.D F2, 45(R3)	✓	✓	✓	✓
MULT.D F0, F2, F4	✓	✓	✓	
SUB.D F8, F6, F2	✓	✓	✓	✓
DIV.D F10, F0, F6	✓			
ADD.D F6, F8, F2	✓	✓	✓	

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	yes	MULT.D	F0	F2	F4			no	no
Mult2	no								
Add	yes	ADD.D	F6	F8	F2			no	no
Divide	yes	DIV.D	F10	F0	F6	Mult1		no	yes

部件名称	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	...	F30
Mult1								
Add								
Divide								



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard举例

- 当DIV.D准备写结果之前的记分牌状态。

指令	指令状态表			
	流出	读操作数	执行	写结果
L.D F6, 34(R2)	✓	✓	✓	✓
L.D F2, 45(R3)	✓	✓	✓	✓
MULT.D F0, F2, F4	✓	✓	✓	✓
SUB.D F8, F6, F2	✓	✓	✓	✓
DIV.D F10, F0, F6	✓	✓	✓	
ADD.D F6, F8, F2	✓	✓	✓	✓

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	no								
Mult2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6			no	no

部件名称	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	...	F30
						Divide		



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Scoreboard

- Scoreboard特点
 - 可实现Out of Order
 - 有WRW冲突和结构冲突不能流出
 - 有WAR冲突也只能等待
 - 无Forwarding硬件机制
 - ...



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

指令的动态调度 - Tomasulo

- Tomasulo's algorithm is a computer architecture hardware algorithm for **dynamic scheduling** of instructions that allows **out-of-order execution**, designed to efficiently utilize multiple execution units.
- It was developed by Robert Tomasulo at IBM in 1967, and first implemented in the IBM System/360 Model 91's floating point unit.
- IBM 360体系结构只有4个双精度浮点寄存器，限制了编译器调度的有效性。
- 360/91的访存时间和浮点计算时间都很长。
- 核心思想
 - 记录和检测指令相关，操作数一旦就绪就立即执行，把发生**RAW冲突**的可能性减少到最小；
 - 通过寄存器换名来消除**WAR冲突**和**WAW冲突**。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

指令的动态调度 - Tomasulo

- 寄存器换名可以消除**WAR冲突**和**WAW冲突**

	DIV.D	F0, F2, F4	
反相关 (F8) 导致WAR冲突	{	ADD.D	F6, F0, F8
		S.D	F6, 0 (R1)
		SUB.D	F8, F10, F14
		MUL.D	F6, F10, F8
			输出相关 (F6) 导致WAW冲突

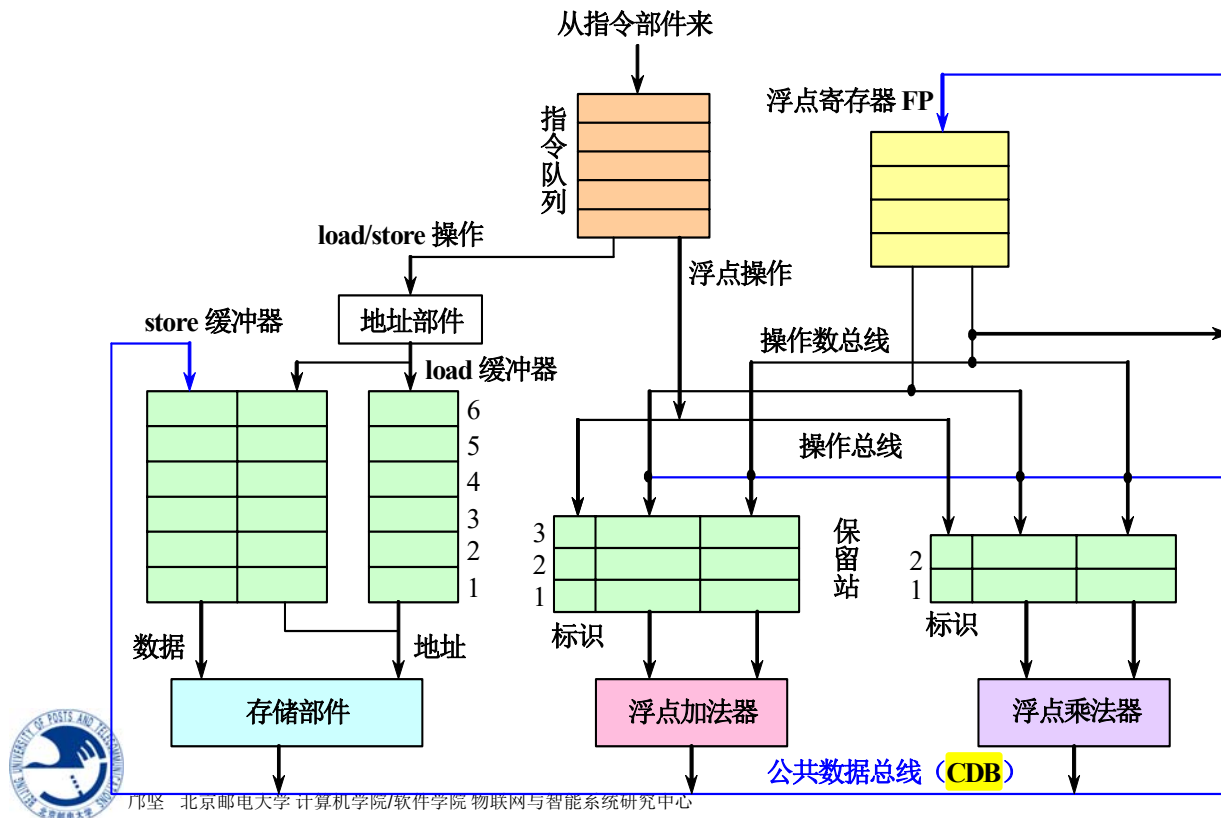
- 消除名相关 - 引入两个临时寄存器S和T

	DIV.D	F0, F2, F4	
	ADD.D	S, F0, F8	两个F6都换名为S
	S.D	S, 0 (R1)	
两个F8都换名为T	{	SUB.D	T, F10, F14
		MUL.D	F6, F10, T



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件



基于Tomasulo算法的MIPS处理器浮点部件

■ 保留站(reservation station)

- 每个保留站中保存一条已经流出并等待到本功能部件执行的指令（相关信息）。
- 包括：操作码、操作数以及用于检测 and 解决冲突的信息。
 - 在一条指令流出到保留站的时候，如果该指令的源操作数已经在寄存器中就绪，则将之取到该保留站中。
 - 如果操作数还没有计算出来，则在该保留站中记录将产生这个操作数的保留站的标识。
- 浮点加法器有3个保留站：ADD1，ADD2，ADD3
- 浮点乘法器有两个保留站：MULT1，MULT2
- 每个保留站都有一个标识字段，唯一地标识了该保留站。



基于Tomasulo算法的MIPS处理器浮点部件

- 公共数据总线**CDB**(Common data bus, 一条重要的数据通路)
 - 所有功能部件的计算结果都是送到**CDB**上, 由它把这些结果直接送到(播送到)各个需要该结果的地方。
 - 在具有**多个执行部件且采用多流出**(即每个时钟周期流出多条指令)的流水线中, 需要采用多条**CDB**。
- **load缓冲器和store缓冲器**
 - 存放读/写存储器的数据或地址
 - **load缓冲器的作用有3个:**
 - 存放用于计算有效地址的分量;
 - 记录正在进行的**load**访存, 等待存储器的响应;
 - 保存已经完成了的**load**的结果(即从存储器取来的数据), 等待**CDB**传输。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

- **store缓冲器的作用有3个:**
 - 存放用于计算有效地址的分量;
 - 保存正在进行的**store**访存的目标地址, 该**store**正在等待存储数据的到达;
 - 保存该**store**的地址和数据, 直到存储部件接收。
- **浮点寄存器FP**
 - 共有**16个浮点寄存器: F0, F2, F4, ..., F30**。
 - 它们通过一对总线连接到功能部件, 并通过**CDB**连接到**store缓冲器**。
- **指令队列**
 - 指令部件送来的指令放入指令队列
 - 指令队列中的指令按先进先出的顺序流出
- **运算部件**
 - 加法器完成加/减法操作, 乘法器完成乘/除法操作



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

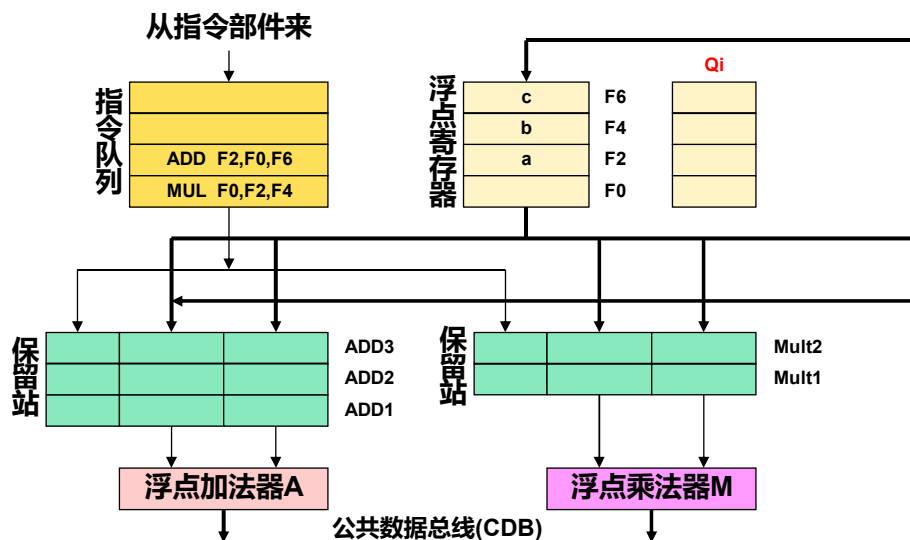
- **寄存器换名**是通过保留站和流出逻辑来共同完成的。
 - 当指令流出时，如果其操作数还没有计算出来，则将该指令中相应的寄存器号**换名为将产生这个操作数的保留站的标识**。
 - 指令流出到保留站后，其操作数寄存器号或者换成了数据本身（如果该数据已经就绪），或者换成了保留站的标识，**不再与寄存器有关系**。
- **Tomasulo算法**具有以下两个特点：
 - 冲突检测和指令执行控制是分布的。
 - 每个功能部件的保留站中的信息决定了什么时候指令可以在该功能部件开始执行。
 - 计算结果通过**CDB**直接从产生它的保留站传送到所有需要它的功能部件，而不用经过寄存器。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

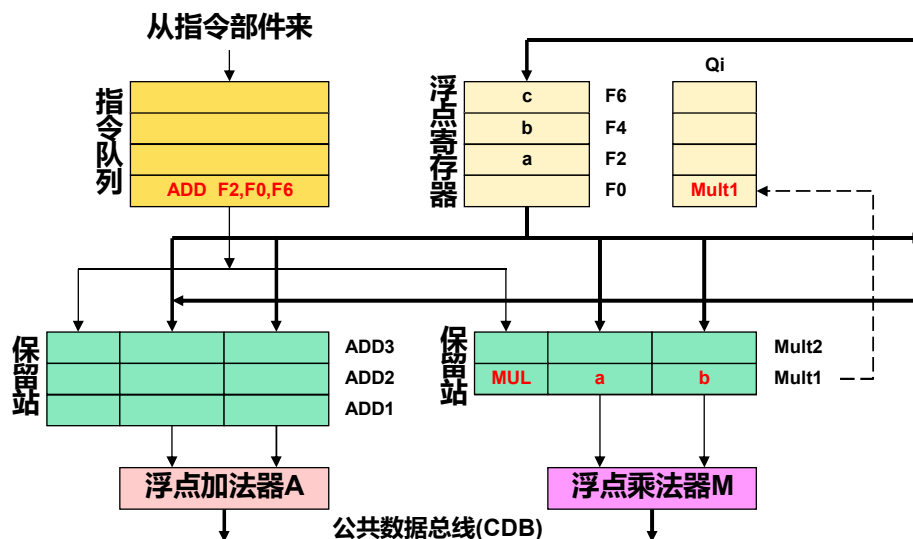
- **简化的demo**：增加一个寄存器状态表 Q_i ，其中的一项指出哪个保留站将产生该寄存器的值。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

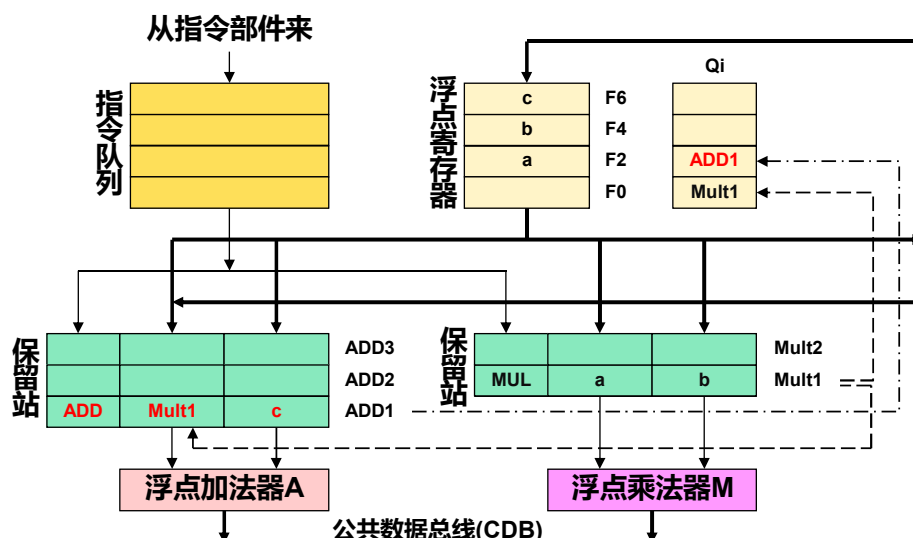
- MUL流出到保留站Mult1时，由于操作数a和b就绪，取到保留站，执行时从保留站取数据；
- 将F0对应的Qi标志置为Mult1，表示该寄存器内容将由保留站Mult1提供。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

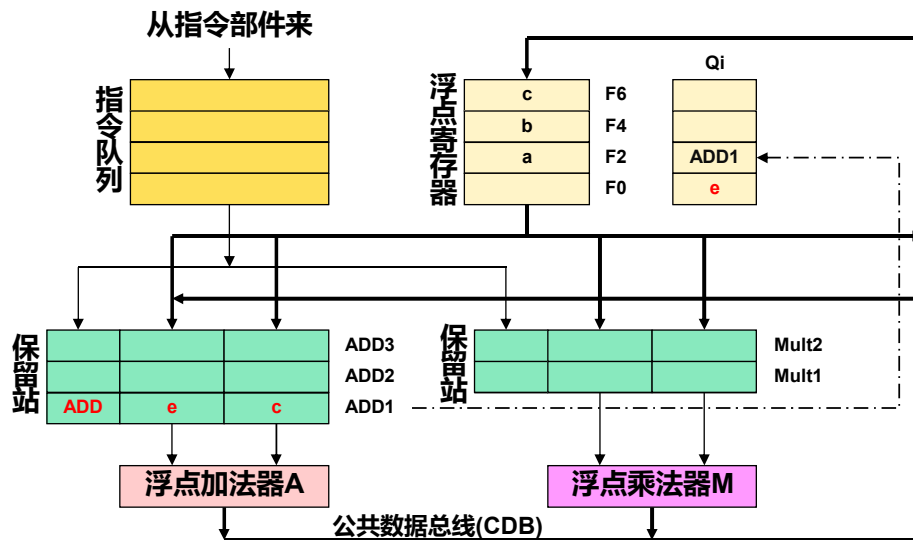
- ADD流出到保留站Add1时，操作数c取到保留站，但由于F0中的操作数没有就绪，标识其提供者Mult1；
- 将F2对应的Qi标志置为Add1，表示该寄存器内容将由保留站Add1提供。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于Tomasulo算法的MIPS处理器浮点部件

- 当Mult1结果(e)产生后，将数据放在总线上广播，所有等待该数据的自动取数据；
- Add1中的ADD指令得到该数据后，可以开始执行。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo指令执行步骤

- 流出：从指令队列的头部取一条指令。
 - 如果该指令的操作所要求的保留站有空闲的，就把该指令送到该保留站（设为r）。
 - 如果其操作数在寄存器中已经就绪，就将这些操作数送入保留站r。
 - 如果其操作数还没有就绪，就把将产生该操作数的保留站的标识送入保留站r。
 - 一旦被记录的保留站完成计算，它将直接把数据送给保留站r。
(寄存器换名和对操作数进行缓冲，消除WAR冲突)
- 完成对目标寄存器的预约工作
(消除了WAW冲突)
- 如果没有空闲的保留站，指令就不能流出。
(发生了结构冲突)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo指令执行步骤

■ 执行

- 当两个操作数都就绪后，本保留站就用相应的功能部件开始执行指令规定的操作。
- **load**和**store**指令的执行需要两个步骤：
 - 计算有效地址（要等到基地址寄存器就绪）
 - 把有效地址放入**load**或**store**缓冲器

■ 写结果

- 功能部件计算完毕后，就将计算结果放到**CDB**上，所有等待该计算结果的寄存器和保留站（包括**store**缓冲器）都同时从**CDB**上获得所需要的数据。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo保留站

保留站						
Busy	Op	Vj	Vk	Qj	Qk	A

■ 保留站

- **Busy**: 为“yes”表示本保留站或缓冲单元“忙”。
- **Op**: 要对源操作数进行的操作。
- **Vj/Vk**: 源操作数的值。
 - 对于每一个操作数来说，**V**或**Q**字段只有一个有效。
 - 对于**load**来说，**Vk**字段用于保存偏移量。
- **Qj/Qk**: 将产生源操作数的保留站号。
 - 等于0表示操作数已经就绪且在**Vj**或**Vk**中，或者不需要操作数。
- **A**: 仅**load**和**store**缓冲器有该字段。开始是存放指令中的立即数字段，地址计算后存放有效地址。

■ Qi: 寄存器状态表

- 每个寄存器在该表中有对应的一项，用于存放将把结果写入该寄存器的保留站的站号。
- 为0表示当前没有正在执行的指令要写入该寄存器，也即该寄存器中的内容就绪。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo算法举例

- 对于下述指令序列，给出当第一条指令完成并写入结果时，Tomasulo算法所用的各信息表中的内容。

L.D F6, 34(R2)

L.D F2, 45(R3)

MUL.D F0, F2, F4

SUB.D F8, F2, F6

DIV.D F10, F0, F6

ADD.D F6, F8, F2



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo算法举例

- 当采用Tomasulo算法时，在上述给定的时刻，保留站、load缓冲器以及寄存器状态表中的内容。

指 令	指令状态表		
	流出	执行	写结果
L.D F6 , 34(R2)	√	√	√
L.D F2 , 45(R3)	√	√	
MUL.D F0 , F2 , F4	√		
SUB.D F8 , F6 , F2	√		
DIV.D F10 , F0 , F6			
ADD.D F6 , F8 , F2			



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo算法举例

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	LD					45+Regs[R3]
Add1	yes	SUB		Mem[34+Regs[R2]]	Load2		
Add2	no						
Add3	no						
Mult1	yes	MUL		Reg[F4]	Load2		
Mult2	no						

	寄存器状态表							
	F0	F2	F4	F6	F8	F10	...	F30
Qi	Mult1	Load2			Add1		...	



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Tomasulo算法的主要优点

- 冲突检测逻辑是分布的（通过保留站和CDB实现）
 - 如果有多条指令已经获得了一个操作数，并同时在等待同一运算结果，那么这个结果一产生，就可以通过CDB同时播送给所有这些指令，使它们可以同时执行。
- 消除了WAW冲突和WAR冲突导致的停顿
 - 使用保留站进行寄存器换名，并且操作数一旦就绪就将之放入保留站。



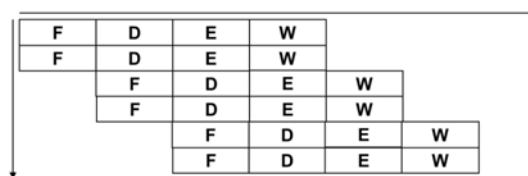
邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态分支预测技术

- 所开发的ILP越多，控制相关的制约就越大，分支预测就要有更高的准确度。
- 对于每个时钟周期流出多条指令（若为n条，就称为n流出）的处理机来说非常重要。因为：
 - 在n-流出的处理机中，遇到分支指令的可能性增加了n倍。要给处理器连续提供指令，就需要预测分支的结果；
 - 机器的CPI越小，控制停顿的相对影响就更大。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心



动态分支预测技术

- 动态分支预测：在程序运行时，**根据分支指令过去的表现来预测其将来的行为。**
- 分支预测的有效性取决于：
 - 预测的准确性
 - 预测正确和不正确两种情况下的分支开销
 - 决定分支开销的因素：
 - 流水线的结构
 - 预测的方法
 - 预测错误时的恢复策略等



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态分支预测技术

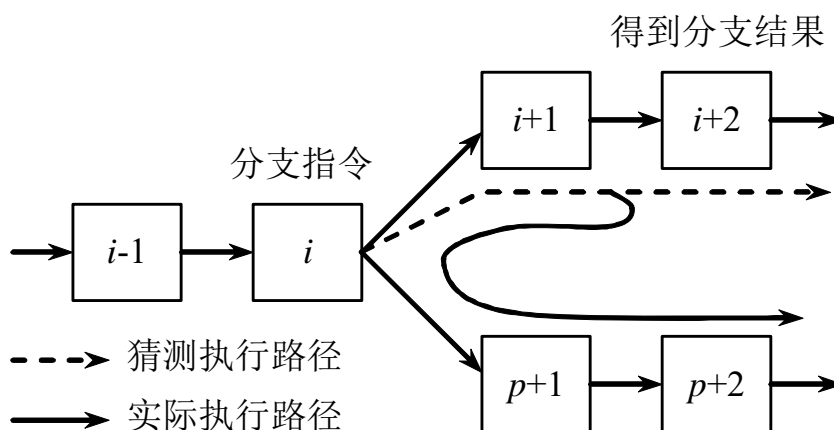
- 采用动态分支预测技术的目的
 - 预测分支是否成功
 - 尽快找到分支目标地址（或指令），避免控制相关造成流水线停顿
- 需要解决的关键问题
 - 如何记录分支的历史信息；
 - 如何根据这些信息来预测分支的去向（甚至取到指令）。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

动态分支预测技术

- 在预测错误时，要作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支历史表 BHT

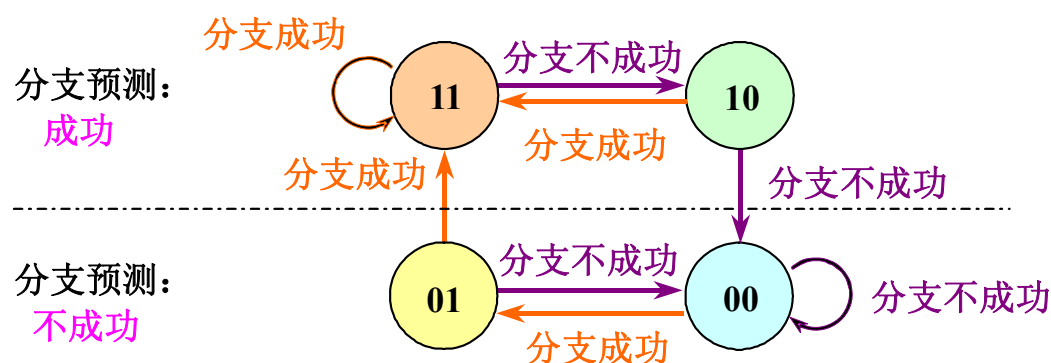
- 分支历史表BHT（Branch History Table）或分支预测缓冲器（Branch Prediction Buffer）
 - 最简单的动态分支预测方法。
 - 用BHT来记录分支指令最近一次或几次的执行情况（成功或不成功），并据此进行预测。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支历史表 BHT

- 采用两位二进制位来记录历史
 - 提高预测的准确度(与1位二进制历史比较)
 - 研究表明：两位分支预测的性能与n位（ $n>2$ ）分支预测的性能差不多。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支历史表 BHT

- 两位分支预测中的操作有两个步骤：
 - 分支预测
 - 当分支指令到达译码段（ID）时，根据从BHT读出的信息进行分支预测。
 - 若预测正确，就继续处理后续的指令，流水线没有断流。否则，就要作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。
 - 状态修改
- 研究表明：对于SPEC89测试程序来说，具有大小为4K的BHT的预测准确率为82%~99%。
- BHT可以跟分支指令一起存放在指令Cache中，也可以用一个专门的硬件来实现。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支历史表 BHT

- BHT方法只在以下情况下才有用：
 - BHT只对分支是否成功进行预测，不提供分支的目标地址。因此在判定分支是否成功所需的时间大于确定分支目标地址所需的时间时才有效。
 - 对于前述5段经典流水线：
 - 由于判定分支是否成功和计算分支目标地址都是在ID段完成，所以BHT方法不会给该流水线带来好处



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支目标缓冲器BTB

- 对于高性能流水线，如**多流出**类的处理机，在准确预测分支的基础上，还要求能快速提供足够的指令流（**4-8条**）。
- 对于前述**5段经典流水线**
 - 如果能提前**1拍**，在**IF**段得到足够信息，就可能将分支的开销降为**0**。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支目标缓冲器BTB

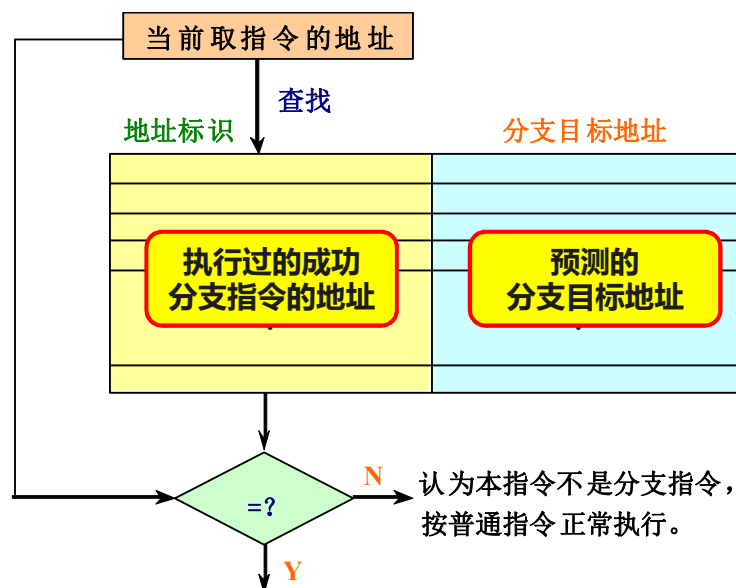
- 方法：**分支目标缓冲**
 - 将分支成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存起来，缓冲区以分支指令的地址作为标识。
 - 这个缓冲区就是分支目标缓冲器（Branch-Target Buffer，简记为BTB，或者Branch-Target Cache）。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

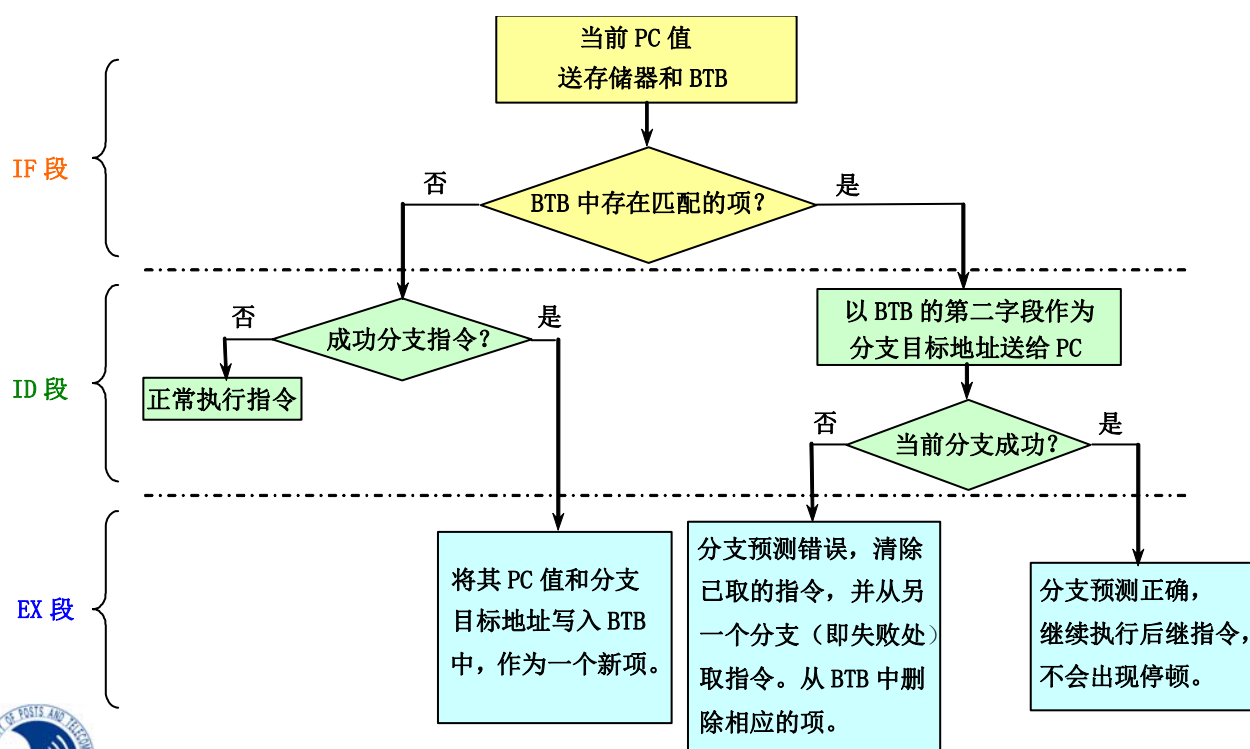
采用分支目标缓冲器BTB

- 指令地址若存在匹配，是分支指令且上一次分支成功



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支目标缓冲器BTB



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支目标缓冲器BTB

- 采用BTB后，各种可能情况下的延迟

指令在BTB中？	预测	实际情况	延迟周期
是	成功	成功	0
是	成功	不成功	2
不是		成功	2
不是		不成功	0



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

采用分支目标缓冲器BTB

- BTB的变体

- 增设2位“分支历史” = BTB+BHT
- 在分支目标缓冲器中存放一条或者多条分支目标处的指令

当前取指令的地址 PC

相联查找

分支指令地址	分支历史表 BHT	分支目标处的若干条指令
A0	T0	$I_{0,0}, I_{0,1}, \dots, I_{0,n-1}$
A1	T1	$I_{1,0}, I_{1,1}, \dots, I_{1,n-1}$
\vdots	\vdots	\vdots
A _{k-1}	T _{k-1}	$I_{k-1,0}, I_{k-1,1}, \dots, I_{k-1,n-1}$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于硬件的前瞻执行

- 前瞻执行（Speculation）的基本思想：
 - 对分支指令的结果进行猜测，并假设这个猜测总是对的
 - 然后按这个猜测结果继续取、流出和执行后续的指令
 - 只是执行指令的结果不是写回到寄存器或存储器，而是放到一个称为**ROB**（Re Order Buffer）的缓冲器（**队列**）中。等到相应的指令得到“确认”（Commit）（即确实是应该执行的）之后，才将结果写入寄存器或存储器。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

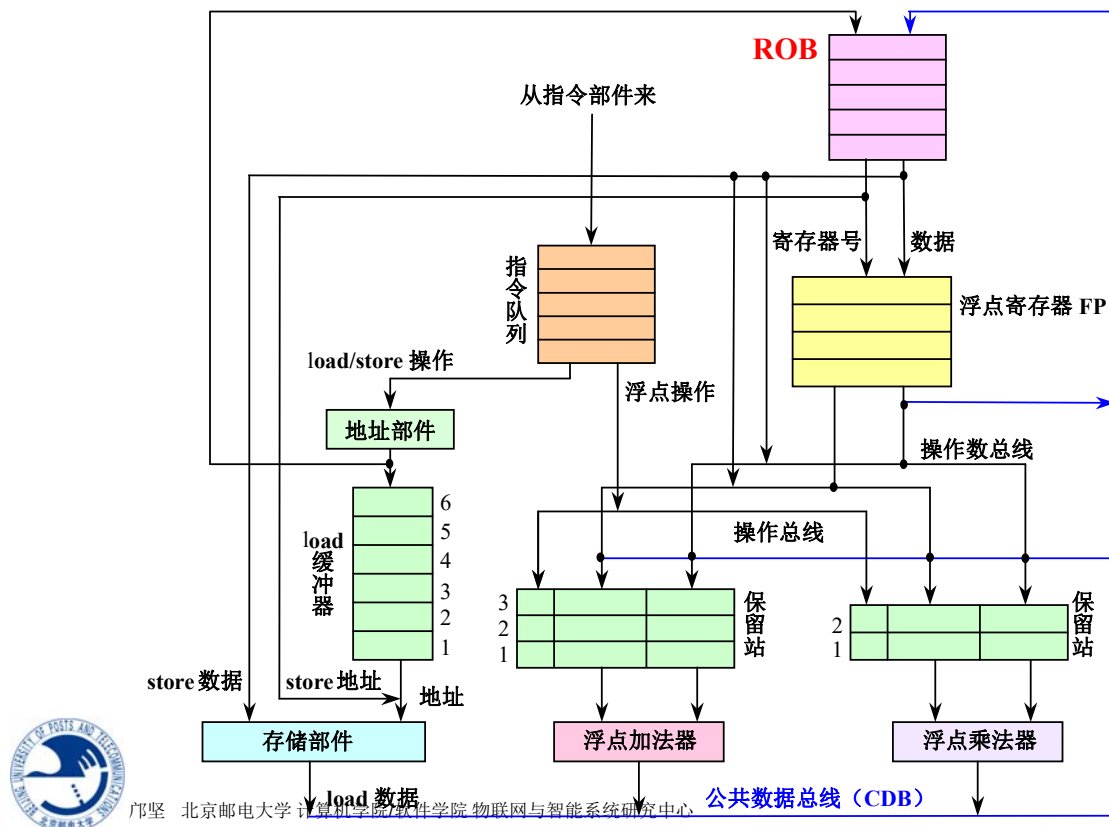
基于硬件的前瞻执行

- 基于硬件的前瞻执行结合了三种思想：
 - 动态分支预测。用来选择后续执行的指令。
 - 在控制相关的结果尚未出来之前，前瞻地执行后续指令。
 - 用动态调度对基本块的各种组合进行跨基本块的调度。
- 关键思想：**允许指令乱序执行，但必须顺序确认。**
- 对Tomasulo算法加以扩充，就可以支持前瞻执行。
 - 把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：**写结果，指令确认**



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

支持前瞻执行的浮点部件结构



基于硬件的前瞻执行

■ 写结果段

- 把前瞻执行的结果写到ROB中；
- 通过CDB在指令之间传送结果，供需要用到这些结果的指令使用。

■ 指令确认段

- 在分支指令的结果出来后，对相应指令的前瞻执行给予确认。
- 如果前面所做的猜测是对的，把在ROB中的结果写到寄存器或存储器。
- 如果发现前面对分支结果的猜测是错误的，那就不予以确认，并从那条分支指令的另一条路径开始重新执行。



支持前瞻执行的浮点部件结构

- ROB中的每一项主要由以下4个字段组成：

ROB				
Busy	指令	状态	目的	Value

- **指令类型（指令）**
 - 指出该指令是分支指令、store指令或寄存器操作指令。
- **目标地址（目的）**
 - 给出指令执行结果应写入的目标寄存器号（如果是load和ALU指令）或存储器单元的地址（如果是store指令）。
- **数据值字段（Value）**
 - 用来保存指令前瞻执行的结果，直到指令得到确认。
- **就绪字段（状态）**
 - 指出指令是否已经完成执行并且数据已就绪。
- Tomasulo算法中保留站的换名功能是由ROB来完成的。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

支持前瞻执行的浮点部件结构

- 采用前瞻执行机制后，指令的执行步骤：

保留站							
Busy	Op	Vj	Vk	Qj	Qk	Dest	A

- **流出**
 - 从浮点指令队列的头部取一条指令。
 - 如果有空闲的保留站（设为r）**且有空闲的ROB项**（设为b），就流出该指令，并把相应的信息放入保留站r和ROB项b。
 - 如果保留站或ROB全满，便停止流出指令，直到它们都有空闲的项。
- **执行**
 - 如果有操作数尚未就绪，就等待，并不断地监测CDB。（检测RAW冲突）
 - 当两个操作数都已在保留站中就绪后，就可以执行该指令的操作。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

支持前瞻执行的浮点部件结构

■ 写结果

- 当结果产生后，将该结果连同本指令在流出段所分配到的ROB项的编号放到CDB上，经CDB写到ROB以及所有等待该结果的保留站。
- 释放产生该结果的保留站。
- store指令在本阶段完成，其操作为：
 - 如果要写入存储器的数据已经就绪，就把该数据写入分配给该store指令的ROB项。
 - 否则，就监测CDB，直到那个数据在CDB上播送出来，这时才将之写入分配给该store指令的ROB项。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

支持前瞻执行的浮点部件结构

- 指令确认 - 对分支指令、store指令以及其他指令的处理不同：
 - 其他指令（除分支指令和store指令）
 - 当该指令到达ROB队列的头部而且其结果已经就绪时，就把该结果写入该指令的目标寄存器，并从ROB中删除该指令。
 - store指令
 - 处理与上面类似，只是它把结果写入存储器。
 - 分支指令
 - 当预测错误的分支指令到达ROB队列的头部时，清空ROB，并从分支指令的另一个分支重新开始执行。（错误的前瞻执行）
 - 当预测正确的分支指令到达ROB队列的头部时，该指令执行完毕。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

前瞻执行举例

- 假设浮点功能部件的延迟时间为：加法2个时钟周期，乘法10个时钟周期，除法40个时钟周期。对于下面的代码段，给出当指令MUL.D即将确认时的状态表内容。

L.D F6, 34 (R2)

L.D F2, 45 (R3)

MUL.D F0, F2, F4

SUB.D F8, F6, F2

DIV.D F10, F0, F6

ADD.D F6, F8, F2



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

前瞻执行举例

- 前瞻执行中MUL.D确认前，保留站和ROB的状态

名称	保留站							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL	Mem[45+ Regs[R2]]	Regs[F4]			#3	
Mult2	yes	DIV		Mem[34+Regs[R2]]	#3		#5	



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

项号	ROB					
	Busy	指令		状态	目的	Value
1	no	L.D	F6, 34 (R2)	确认	F6	Mem[34+Regs[R2]]
2	no	L.D	F2, 45 (R3)	确认	F2	Mem[45+Regs[R3]]
3	yes	MUL.D	F0, F2, F4	写结果	F0	#2×Regs[F4]
4	yes	SUB.D	F8, F6, F2	写结果	F8	#1－#2
5	yes	DIV.D	F10, F0, F6	执行	F10	
6	yes	ADD.D	F6,F8,F2	写结果	F6	#4＋#2

字段	浮点寄存器状态							
	F0	F2	F4	F6	F8	F10	...	F30
ROB项编号	3			6	4	5		
Busy	yes	no	no	yes	yes	yes	...	no



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

前瞻执行的漏洞问题

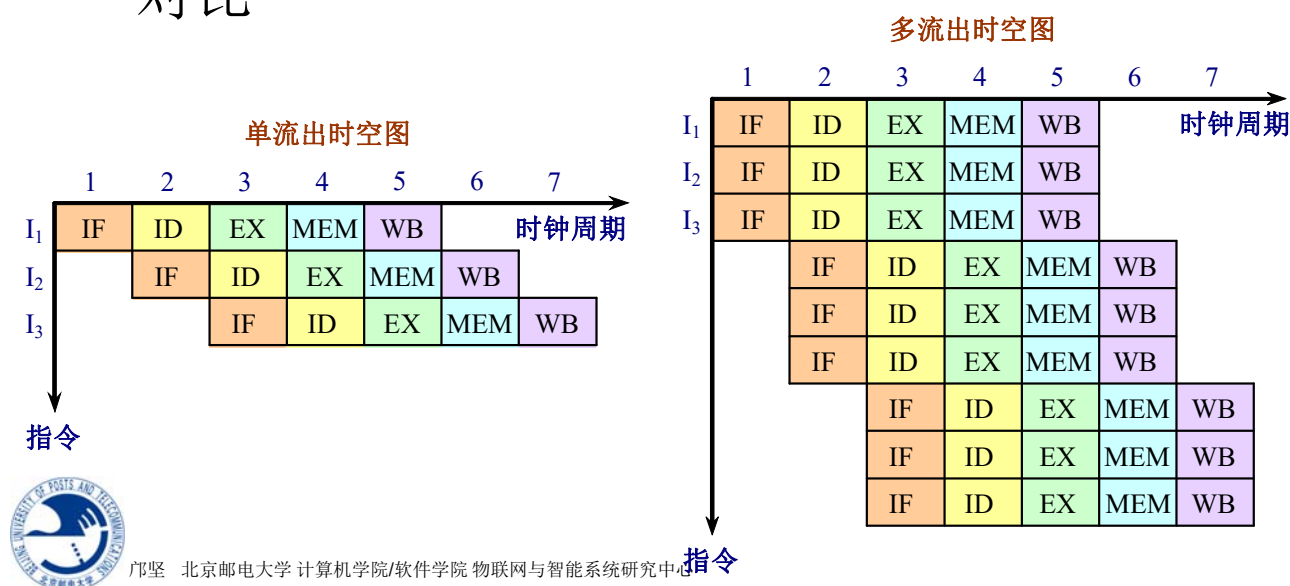
- 熔断(Meltdown)和幽灵(Spectre)漏洞
 - 也称为“旁路分析利用”漏洞
 - 利用乱序/前瞻执行技术，破坏了内存隔离机制，使恶意程序可越权访问操作系统内存数据，造成敏感信息泄露
 - 恶意程序从其它程序的内存中窃取信息，包括密码、账户信息、加密密钥和理论上存储在进程中的任何内容
 - 原因：如果处理器猜错了，那么被错误抓取的数据依旧会存储在Cache中...
 - ...



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

多指令流出技术

- 一个时钟周期内流出多条指令， $CPI < 1$ 。
- 单流出和多流出处理机执行指令的时空图对比



多指令流出技术

- 多流出处理机有两种基本风格
 - 超标量（Superscalar）
 - 在每个时钟周期流出的指令条数不固定，依代码的具体情况而定。（有上限）
 - 设这个上限为m，就称该处理机为m流出。
 - 可以通过编译器进行静态调度（按序执行），也可以基于Tomasulo算法进行动态调度（乱序执行）。
 - 超长指令字VLIW（Very Long Instruction Word）
 - 在每个时钟周期流出的指令条数是固定的，这些指令构成一条长指令或者一个指令包。
 - 指令包中，指令之间的并行性是通过指令显式地表示出来的。
 - 指令调度是由编译器静态完成的。



多指令流出技术

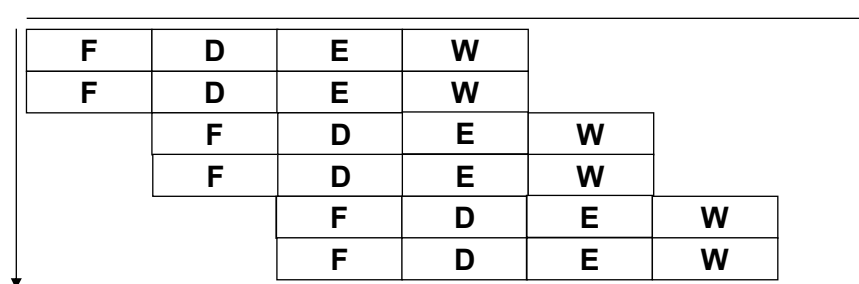
技 术	流出结构	冲突检测	调 度	主要特点	处理机实例
超标量 (静态)	动态	硬件	静态	顺序执行	Sun UltraSPARCII/III
超标量 (动态)	动态	硬件	动态	部分乱序执行	IBM Power2
超标量 (前瞻)	动态	硬件	带有前瞻的动态执行	带有前瞻的乱序执行	Pentium III/4, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64III
VLIW /LIW	静态	软件	静态	流出包之间没有冲突	Trimedia, i860
EPIC	主要是静态	主要是软件	主要是静态	相关性被编译器显式地标记出来	Itanium



EPIC: Explicitly Parallel Instruction Computer。扩展的VLIW，静态和动态方法的组合

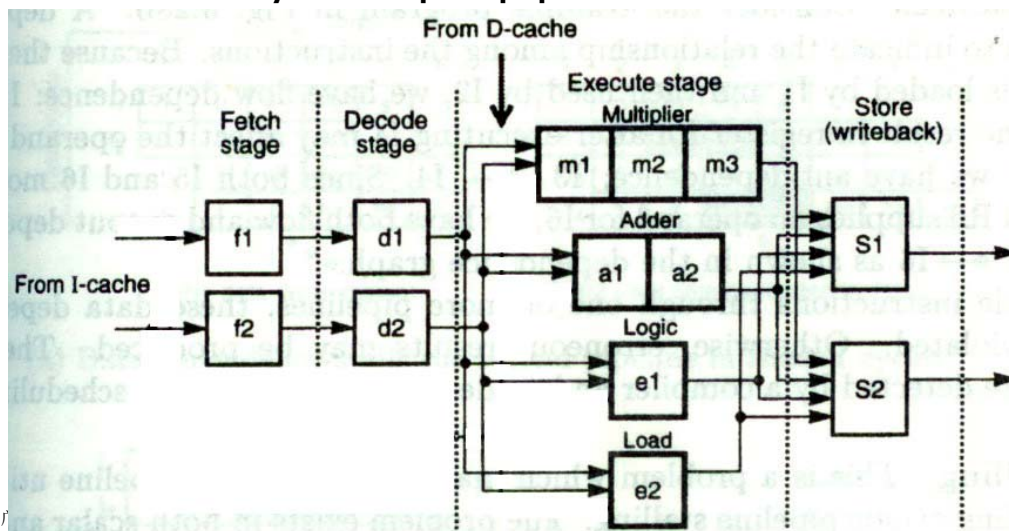
Superscalar

- A superscalar machine of degree m – m instructions are issued per cycle and ILP should be m in order to fully utilize the pipeline.



Superscalar Pipeline

- Superscalar Pipeline – In an m-issue superscalar machine, the decode and execution resources are increased to form essentially m pipelines operating concurrently, the functional units may be shared by multiple pipelines.



Multipipeline scheduling

- Issue & completion policies:
 - In-order issue with in-order-completion
 - In-order issue and out-of-order completion
 - Out-of-order issue and out-of-order completion



Demo code

```

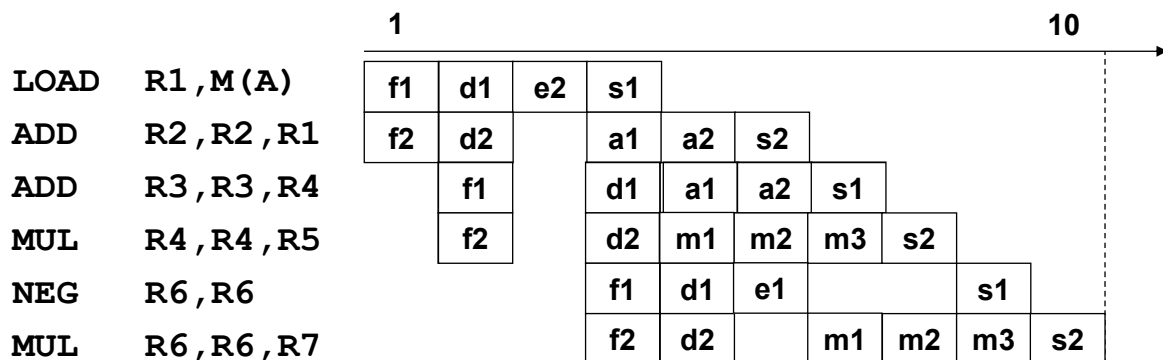
I1.    LOAD    R1 ,M (A)
I2.    ADD     R2 ,R2 ,R1
I3.    ADD     R3 ,R3 ,R4
I4.    MUL     R4 ,R4 ,R5
I5.    NEG     R6 ,R6
I6.    MUL     R6 ,R6 ,R7
    
```

相关性?



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Issue & completion

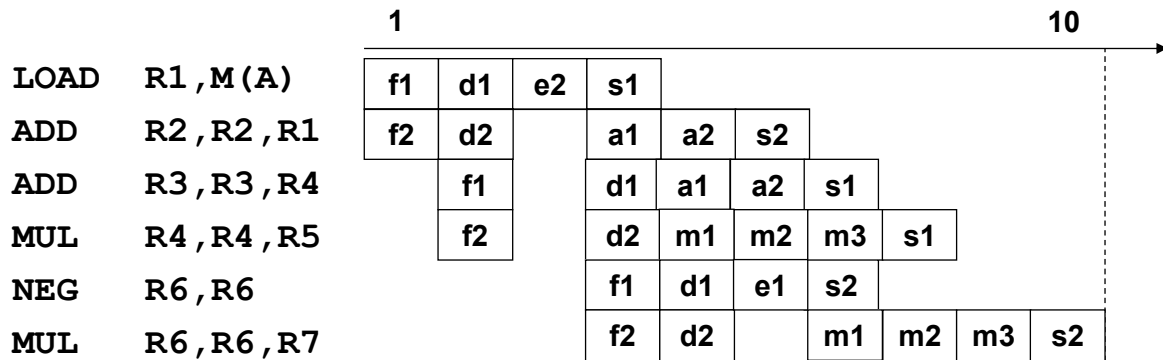


In-order issue with in-order-completion

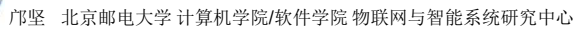


邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

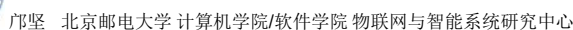
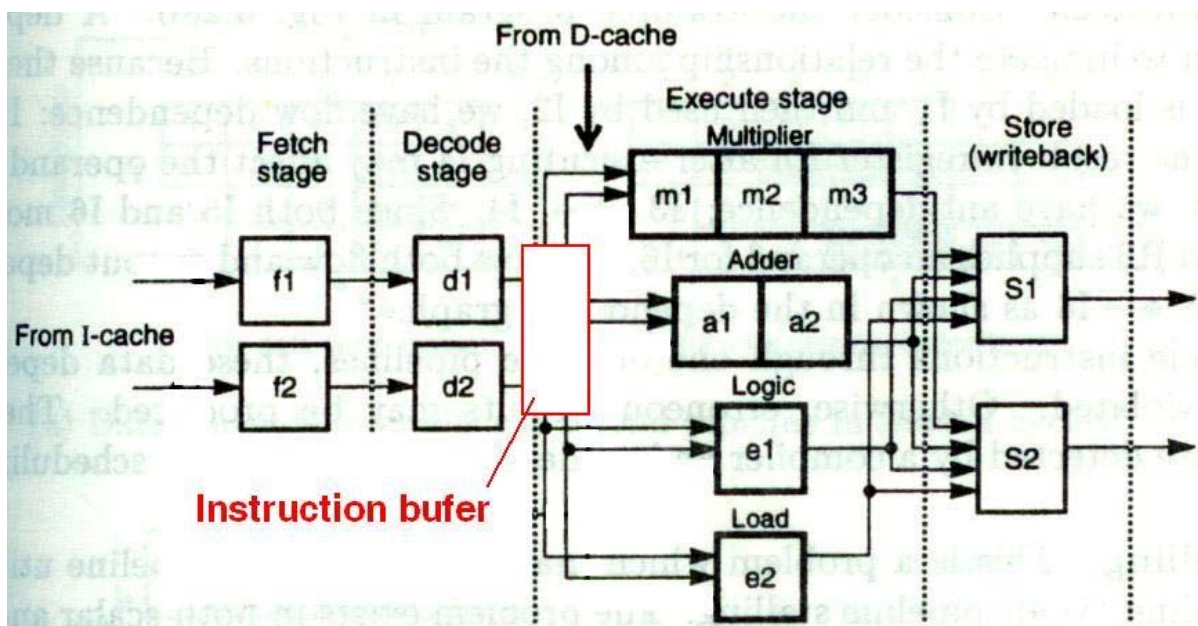
Issue & completion



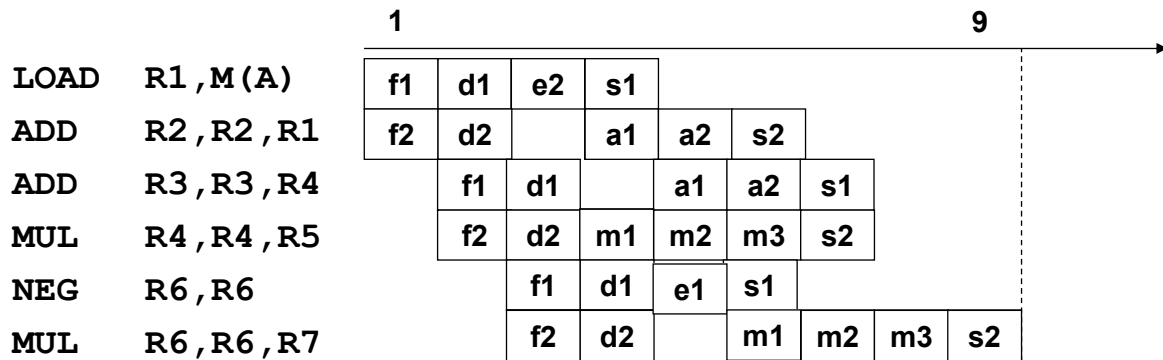
In-order issue and out-of-order completion



Issue & completion



Issue & completion

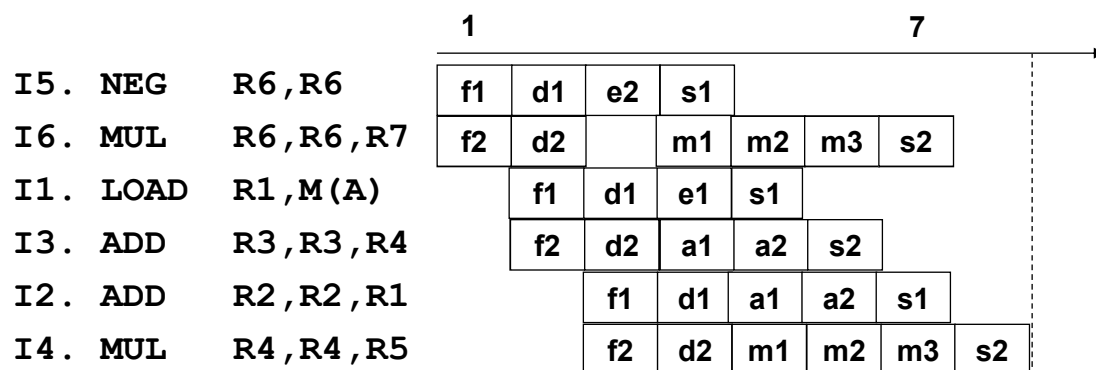


Out-of-order issue and out-of-order completion



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Issue & completion



Out-of-order issue and out-of-order completion
(with SW support)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心



How about PowerPC ?

基于静态调度的多流出技术

- 基于静态调度的多流出技术
 - 指令按序流出，在流出时硬件进行冲突检测。如果有，只流出该指令之前的指令。
- 假设一种每个时钟周期流出两条指令的MIPS处理器：1条整数型指令+1条浮点操作指令
 - 同时取两条指令，译码两条指令

指令类型	流水线工作情况							
整数指令	IF	ID	EX	MEM	WB			
浮点指令	IF	ID	EX	EX	MEM	WB		
整数指令		IF	ID	EX	MEM	WB		
浮点指令		IF	ID	EX	EX	MEM	WB	
整数指令			IF	ID	EX	MEM	WB	
浮点指令			IF	ID	EX	EX	MEM	WB
整数指令				IF	ID	EX	MEM	WB
浮点指令				IF	ID	EX	EX	MEM

基于静态调度的多流出技术

- 需要增加的硬件不多。
 - 增加冲突逻辑检测电路
 - 如果浮点load或浮点store指令使用整数部件，会增加对浮点寄存器的访问冲突。
 - 增设一个浮点寄存器的读/写端口
 - 由于流水线中的指令多了一倍，定向路径也要增加
- 限制超标量流水线的性能发挥的障碍。
 - load后续3条指令都不能使用其结果，否则就会引起停顿。
 - 分支延迟
 - 如果分支指令是流出包中的第一条指令，则其延迟是3条指令
 - 否则就是流出包中的第二条指令，其延迟就是两条指令



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

- 扩展Tomasulo算法：支持双流出超标量
 - 每个时钟周期流出两条指令；
 - 一条是整数指令，另一条是浮点指令。
- 采用一种比较简单的方法
 - 指令按顺序流向保留站，否则会破坏程序语义
 - 将整数所用的表结构与浮点用的表结构分离开，分别进行处理，这样就可以同时地流出一条浮点指令和一条整数指令到各自的保留站



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

- 有两种不同的方法可以实现多流出。
 - 在半个时钟周期里完成流出步骤，这样一个时钟周期就能处理两条指令。
 - 设置一次能同时处理两条指令的逻辑电路。
- 现代的流出4条或4条以上指令的超标量处理机经常是两种方法都采用。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

- 例：对于采用了Tomasulo算法和多流出技术的MIPS流水线，考虑以下简单循环的执行。该程序把F2中的标量加到一个向量的每个元素上。

```
Loop:  L.D      F0, 0 (R1)    // 取一个数组元素放入F0
        ADD.D   F4, F0, F2    // 加上在F2中的标量
        S.D     F4, 0 (R1)    // 存结果
        DADDIU  R1, R1, #-8    // 将指针减少8（每个数据占8个字节）
        BNE     R1, R2, Loop  // 若R1不等于R2，转移到Loop继续
```

- 列出该程序前面3遍循环中各条指令的流出、开始执行和将结果写到CDB上的时间。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

■ 假设：

- 每个时钟周期能流出一条整数指令和一条浮点指令，即使它们相关也是如此。
- 有一个整数部件，用于整数ALU运算和地址计算；并且对于每一种浮点操作类型都有一个独立的流水化了的浮点功能部件。
- 指令流出和写结果各占用一个时钟周期。
- 具有动态分支预测部件和一个独立的计算分支条件的功能部件。
- 分支指令单独流出，没有采用延迟分支，但分支预测是完美的。分支指令完成前，其后续指令只能被取出和流出，但不能执行。
- 因为写结果占用一个时钟周期，所以产生结果的延迟为：整数运算一个周期，load两个周期，浮点加法运算3个周期。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

遍数	指 令	流出	执行	访存	写CDB	说明
1	L. D F0, 0 (R1)	1	2	3	4	流出第一条指令
1	ADD. D F4, F0, F2	1	5		8	等待L. D的结果
1	S. D F4, 0 (R1)	2	3	9		等待ADD. D的结果
1	DADDIU R1, R1, #-8	2	4		5	等待ALU
1	BNE R1, R2, Loop	3	6			等待DADDIU的结果
2	L. D F0, 0 (R1)	4	7	8	9	等待BNE完成
2	ADD. D F4, F0, F2	4	10		13	等待L. D的结果
2	S. D F4, 0 (R1)	5	8	14		等待ADD. D的结果
2	DADDIU R1, R1, #-8	5	9		10	等待ALU
2	BNE R1, R2, Loop	6	11			等待DADDIU的结果
3	L. D F0, 0 (R1)	7	12	13	14	等待BNE完成
3	ADD. D F4, F0, F2	7	15		18	等待L. D的结果
3	S. D F4, 0 (R1)	8	13	19		等待ADD. D的结果
3	DADDIU R1, R1, #-8	8	14		15	等待ALU
3	BNE R1, R2, Loop	9	16			等待DADDIU的结果

基于动态调度的多流出技术

- 虽然指令的流出率比较高，但是执行效率并不是很高。
 - 16拍共执行15条指令，
 - 平均指令执行速度为 $15/16=0.94$ 条/拍。
- 原因是浮点运算少，ALU部件成了瓶颈。
 - 解决方法：增加一个加法器，把ALU功能和地址运算功能分开。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

基于动态调度的多流出技术

- 上述双流出动态调度流水线的性能受限于以下3个因素：
 - 整数部件和浮点部件的工作负载不平衡，没有充分发挥出浮点部件的作用。
 - 应该设法减少循环中整数型指令的数量。
 - 每个循环迭代中的控制开销太大。
 - 5条指令中有两条指令是辅助指令。
 - 应该设法减少或消除这些指令。
 - 控制相关使得处理机必须等到分支指令的结果出来后才能开始下一条L.D指令的执行。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

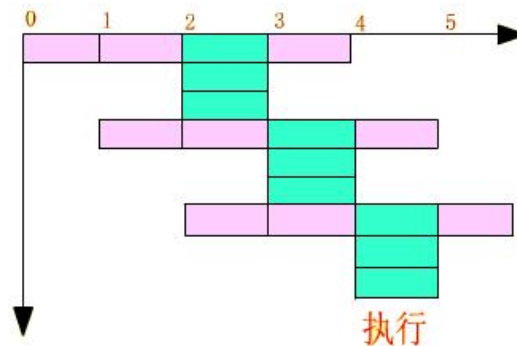
VLIW – static multiple issue

- VLIW (very lone instruction word) – packages the multiple operation into one very long instruction.
- Multi-issue – VLIW machine use **multiple, independent functional unit**.
- The burden for choosing the instructions to be issued simultaneously fall on the **compiler**.



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

VLIW

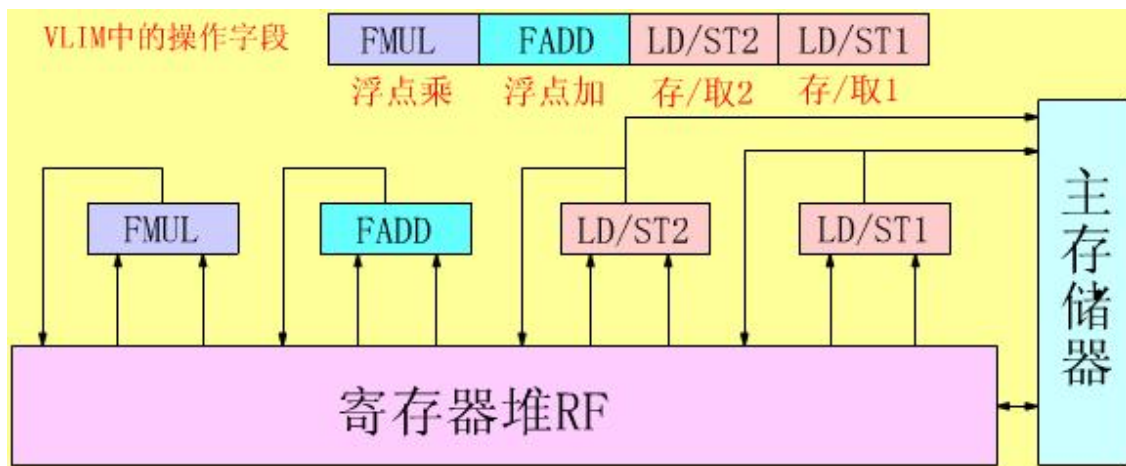


- 一种单指令多操作码多数据的系统结构；并行操作主要是在流水的执行阶段进行；
- 设置多个功能部件。
- 指令字被分割成一些字段，每个字段称为一个**操作槽**，直接独立地控制一个功能部件。
- 字长与机器中的执行部件数有关，指令字长度约在100-1000bit之间；
- 典型的机器：Cydrome公司Cydra 5(1989年)、飞利浦公司TM-1(1996年)。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

VLIW example



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

VLIW example

源代码	操 作 性 质	所需周期
C=A+B	Load A	1
	Load B	1
	C=A+B	1
	Store C	1
K=I+J	Load I	1
	Load J	1
	K=I+J	1
	Store K	1
L=M-N	Load M	1
	L=M-N	1
	Store L	1
Q=C*K	Q=C*K	2
	Store Q	1



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

VLIW example

1	Load A	Load B		
2	Load I	Load J	$C = A+B$	
3	Load M	Store C	$K = I+J$	
4		Store K	$L = M-K$	$Q = C*K$
5		Store L		
6	Store Q			



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

超长指令字技术 (VLIW)

■ VLIW存在的一些问题

- 程序代码长度增加了
 - 提高并行性而进行的大量的循环展开。
 - 指令字中的操作槽并非总能填满。
- 采用了锁步机制
 - 任何一个操作部件出现停顿时，整个处理机都要停顿。
- 机器代码的不兼容性



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

多流出处理器受到的限制

- 主要受以下三个方面的影响：
 - 程序所固有的指令级并行性。
 - 硬件实现上的困难。
 - 超标量和超长指令字处理器固有的技术限制。
- 超标量处理机与VLIW处理机相比有两个优点
 - 超标量结构对程序员是透明的，因为处理机能自己检测下一条指令能否流出，从而不需要重新排列指令来满足指令的流出。
 - 即使是没有经过编译器针对超标量结构进行调度优化的代码或是旧的编译器生成的代码也可以运行。要想达到很好的效果，使用动态超标量调度技术。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

超流水线处理机

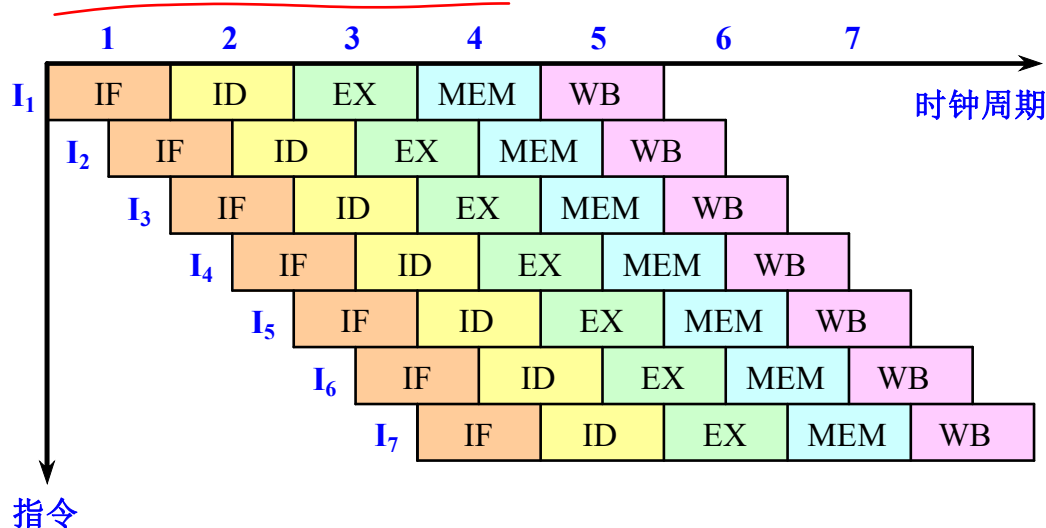
- Superpipelining
 - 将每个流水段进一步细分，这样在一个时钟周期内能够分时流出多条指令。这种处理机称为超流水线处理机。
 - 对于一台每个时钟周期能流出 n 条指令的超流水线计算机来说，这 n 条指令不是同时流出的，而是每隔 $1/n$ 个时钟周期流出一条指令。
 - 实际上该超流水线计算机的流水线周期为 $1/n$ 个时钟周期。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

超流水线处理机

- 一台每个时钟周期分时流出两条指令的超流水线计算机的时空图



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

超流水线处理机

- 超标量和超流水线
 - 超标量采用空间并行性。通过重复设置多份硬件来提高性能；
 - 超流水线采用时间并行性。只需增加少量硬件，通过各部分硬件的充分重叠工作来提高性能。
- 另一种超流水线处理机 - 指令流水线级数为8或8以上的流水线处理机。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

典型的超流水线处理器

■ SGI公司的MIPS系列R4000

■ R4000微处理器有2个Cache:

- 指令Cache和数据Cache，容量都是8 KB，每个Cache的数据宽度为64 b

■ R4000的核心处理部件:

- 整数部件：一个 32×32 位的通用寄存器组，一个算术逻辑部件（ALU），一个专用的乘法/除法部件
- 浮点部件(可以并行工作)
 - 一个执行部件，浮点乘法部件，浮点除法部件，浮点加法/转换/求平方根部件
 - 一个 16×64 位的浮点通用寄存器组。浮点通用寄存器组也可以设置成32个32位的浮点寄存器。

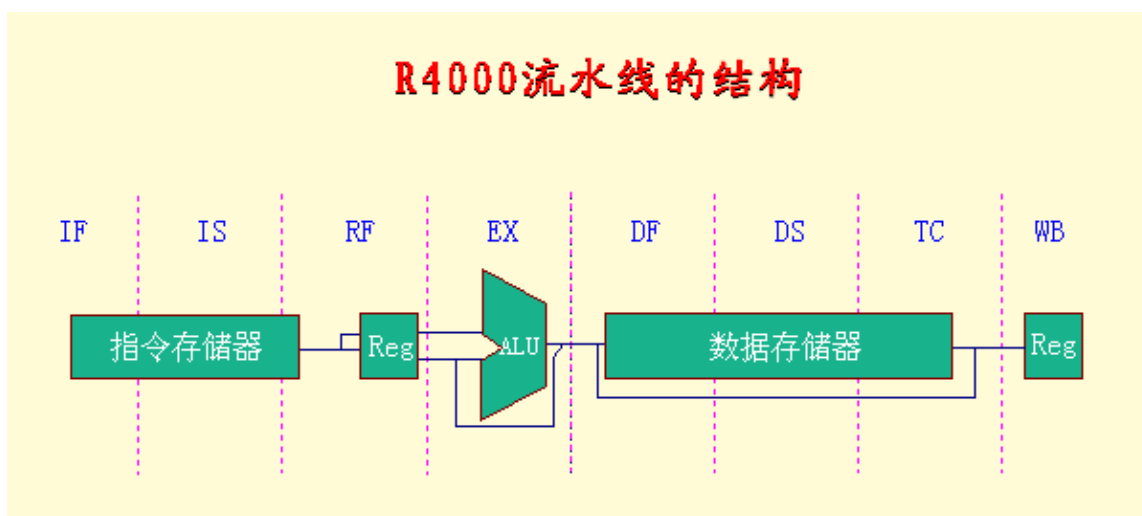


邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

典型的超流水线处理器

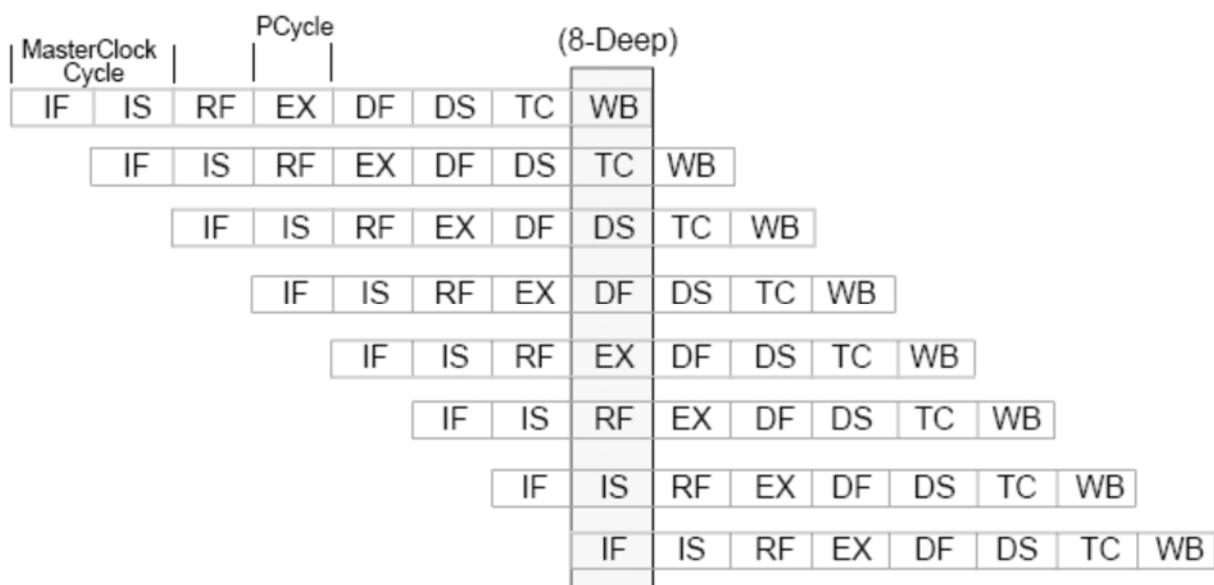
■ R4000的指令流水线有8级

R4000流水线的结构



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS R4000 Pipeline Stages



P.44 MIPS R4000 Microprocessor User's Manual

邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS R4000 Pipeline Stages

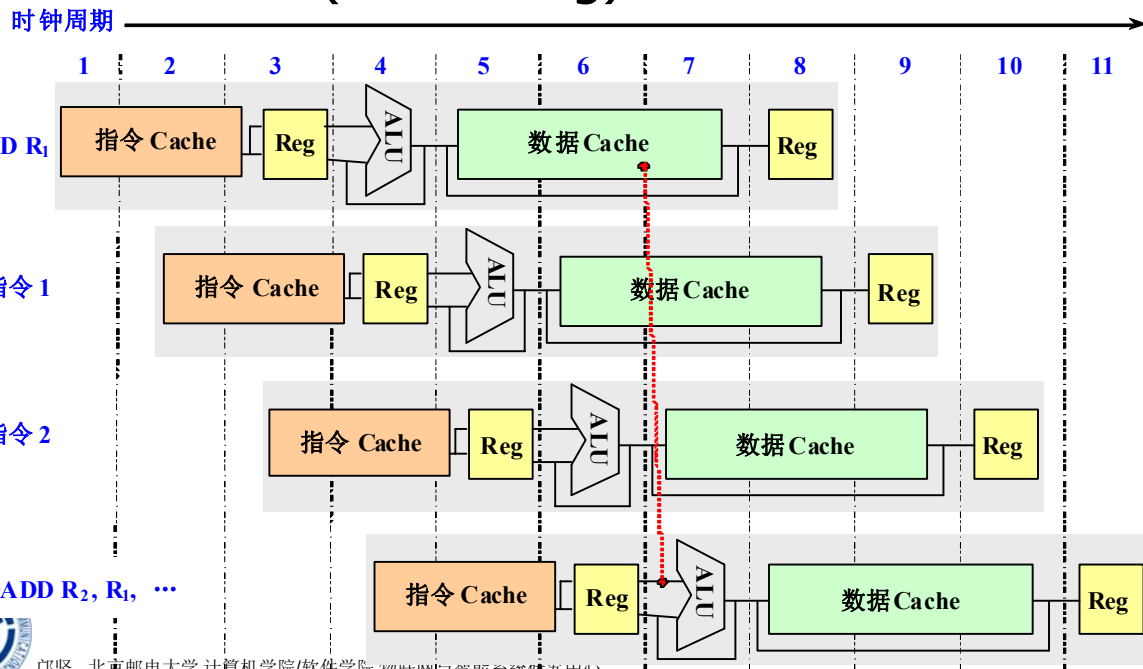
- **IF**: 取指令的前半步，根据PC值去启动对指令Cache的访问。
- **IS**: 取指令的后半步，在这一级完成对指令Cache的访问。
- **RF**: 指令译码，访问寄存器组读取操作数，冲突检测，并判断指令Cache是否命中。
- **EX**: 指令执行。包括有效地址计算，ALU操作，分支目标地址计算，条件码测试。
- **DF**: 取数据的前半步，启动对数据Cache的访问。
- **DS**: 取数据的后半步，在这一级完成对数据Cache的访问。
- **TC**: 标识比较，判断对数据Cache的访问是否命中。
- **WB**: load指令或运算型指令把结果写回寄存器组。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

MIPS R4000

- Load数据在DS末尾才准备好，载入延迟为两个时钟周期(forwarding)



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

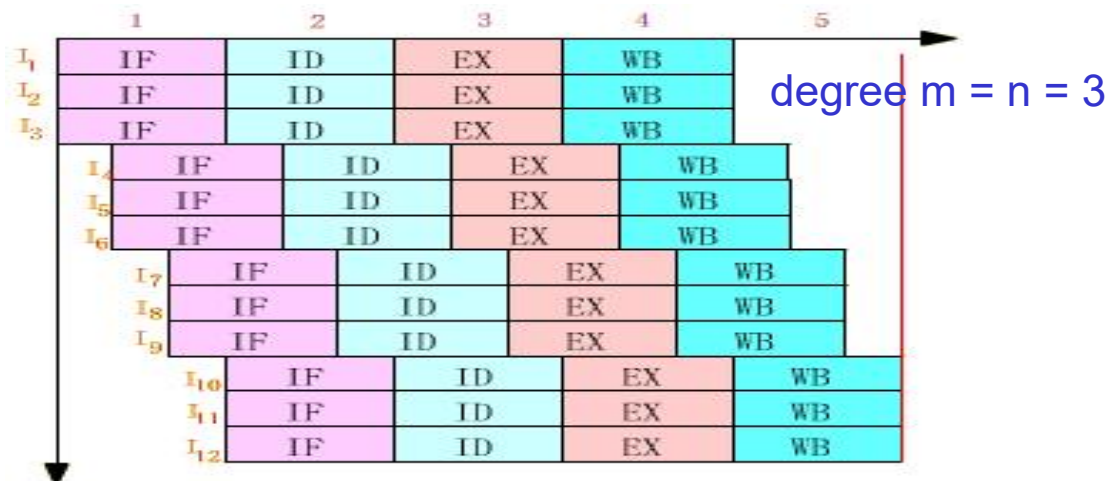
Machine Parallelism

Machine type	Scalar base machine of k pipe stage	Superscalar machine of degree m	Superpipelined machine of degree n	Superpipelined Superscalar machine of degree (m,n)
Machine pipeline cycle	1	1	1/n	1/n
Instruction issue rate	1	m	1	m
Instruction issue latency	1	1	1/n	1/n
ILP to fully utilize the pipeline	1	m	n	mn



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Superpipelined superscaler



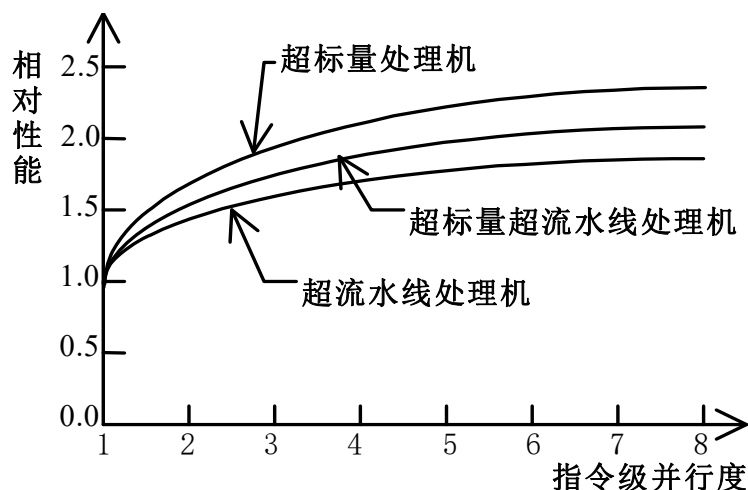
- The machine executes m instructions every cycle with a pipeline cycle $1/n$ of base cycle.

DEC Alpha processor – $(m, n) = (2, 6)$



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Relative performance



- ILP因素：
 - 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。



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

Relative performance

- Startup delay – superpipelined machine has longer startup delay than superscaler machine.
- Branch – may create more damage on the superpipelined machine than on the superscaler machine.
- Resource conflict – superpipeline vs superscaler ?



邱坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

流水的异常处理

- 精确(Precise)异常：如果发生异常时，处理机的现场跟严格按程序顺序执行时指令*i*的现场相同 -立即响应，后续指令作废；
- 不精确(Imprecise)异常：当执行指令*i*导致发生异常时，处理机的现场（状态）与严格按程序顺序执行时指令*i*的现场不同。
 - 流水线可能已经执行完按程序顺序是位于指令*i*之后的指令；
 - 流水线可能还没完成按程序顺序是指令*i*之前的指令。

Precise/Imprecise	Exception Type
Imprecise	Machine check System reset
Precise	External interrupt Decrementer System management interrupt
Precise	Instruction-caused exceptions

