

Ch3--

ILP及其探索

Ch3-2

- 动态调度——Tomasulo算法
- 记分板+寄存器重命名

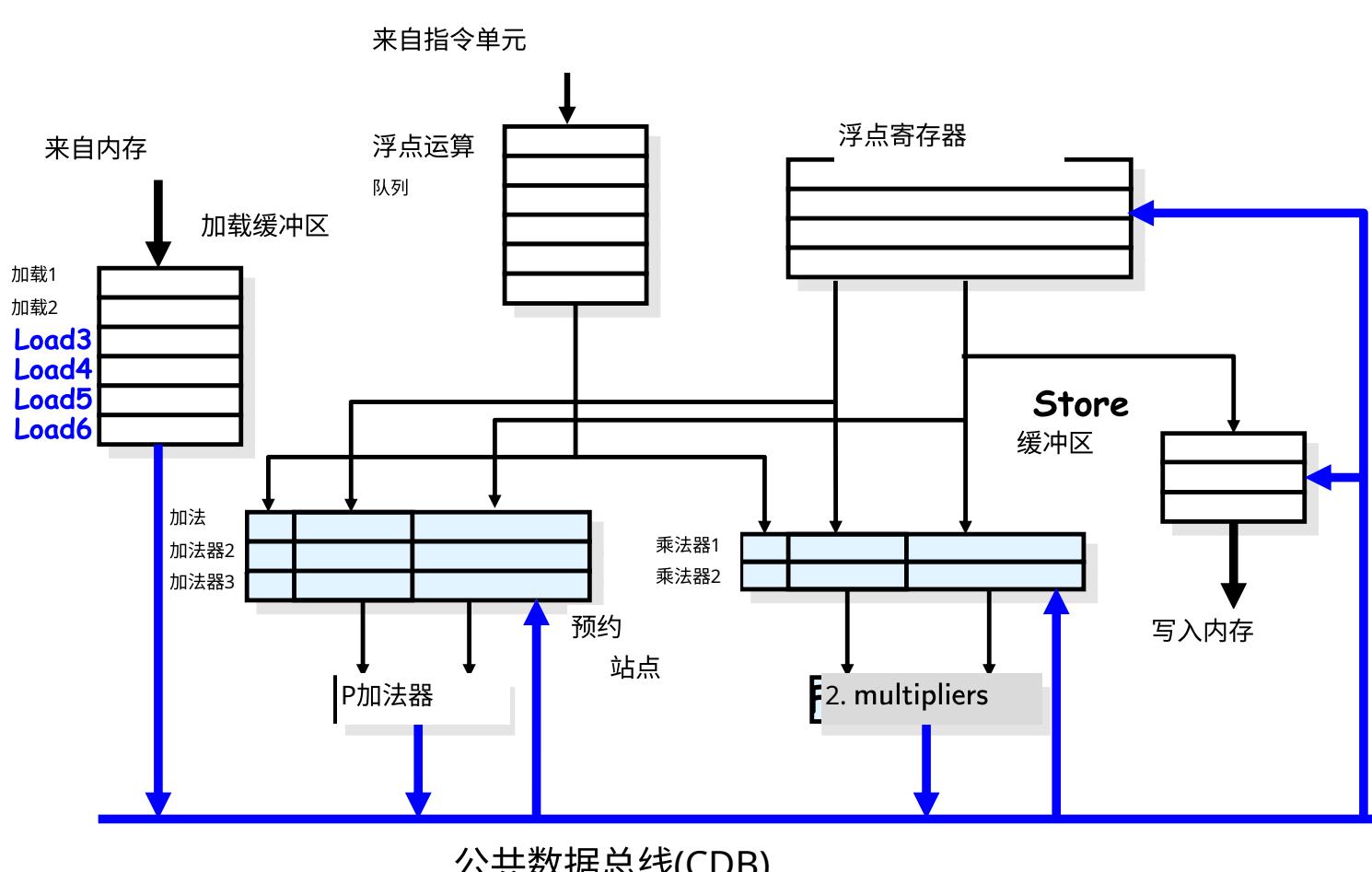
3.4,3.5

基于Tomasulo算法的动态调度

- 适用于IBM 360/91（缓存技术出现前）
- 目标：无需特殊编译器实现高性能
- 浮点寄存器数量过少（360机型仅4个）制约了编译器对操作调度的优化空间
 - > 这促使Tomasulo探索如何通过硬件重命名机制实现更高效的寄存器
- 为何研究1966年的计算机？
- 其技术衍生品已蔚然成林！
 - » Alpha 21264、HP 8000、MIPS 10000、奔腾III、PowerPC604...

·3

Tomasulo体系结构



·5

Tomasulo算法的三个阶段

- 发射阶段——从浮点操作队列获取指令。若保留站空闲（无结构冲突），控制器发射指令并发送操作数（寄存器重命名）。
- 执行阶段——操作数运算(EX)。当两个操作数就绪时执行；若未就绪，则监听公共数据总线等待结果。
- 写回阶段——完成执行(WB)
 - 将数据写入公共总线供所有等待单元读取；标记保留站为可用状态

特点

- > 乘法器等运算功能单元
- > 顺序发射指令，乱序完成
- > 取指阶段 → 发射阶段，Ro
- > 4 级流水线
- > 集中式记分牌控制
- 缺点
 - > WAW/WAR冲突时停顿

- 功能单元少，非流水线
- 顺序发射，乱序完成
- 浮点操作队列、保留站、加载/存储缓冲区、公共数据总线
- 寄存器重命名 → 消除写后写、读后写冲突
- 减少结构冒险
- 写后读检测分散式保留
- CDB → 转发路径

·2

托马斯洛算法

- 控制逻辑与缓冲器分布在功能单元(FU)中
 - > 功能单元缓冲区称为“保留站”，存储待处理操作数
- 指令中的寄存器被值或保留站(RS)指针替代，称为寄存器重命名
 - > 避免写后读、写后写冲突
 - > 保留站数量超过寄存器，可实现编译器无法完成的优化
- 结果通过公共数据总线从保留站直接传输至功能单元（不经过寄存器），该总线向所有功能单元广播结果
- 加载/存储操作同样视为配备保留站的功能单元
- 整数指令可跨越分支执行，允许浮点队列中的浮点操作突破基本块限制

·4

保留站组成部件

保留站：

- Op：该单元待执行的操作
- V_j, V_k: 源操作数值
 - > 存储缓冲区包含 V 字段，用于暂存待写入结果
- Q_j, Q_k：保留站生成源寄存器值(待写入数据)
- > 注: Q_j, Q_k = 0 ⇒ 就绪
- > 存储缓冲区仅保存生成结果的RS站Q_i标识
- A : 存储内存地址计算信息
- 忙碌: 表示保留站或功能单元处于工作状态
- 寄存器结果状态 - 指示哪个功能单元将写入每个寄存器（若存在）。若无待处理指令需写入该寄存器，则留空。

·6

数据通路

- 常规数据总线：数据 + 目标地址（“去向”总线）
 - 公共数据总线：数据 + 源地址（“来源”总线）
 - > 64 位数据 + 4位功能单元源地址
 - > 若符合预期功能单元则写入（产生结果）
 - > 广播是否
 - 示例速度：
 - 浮点加减法需3时钟周期；乘法需10周期；除法需40周期

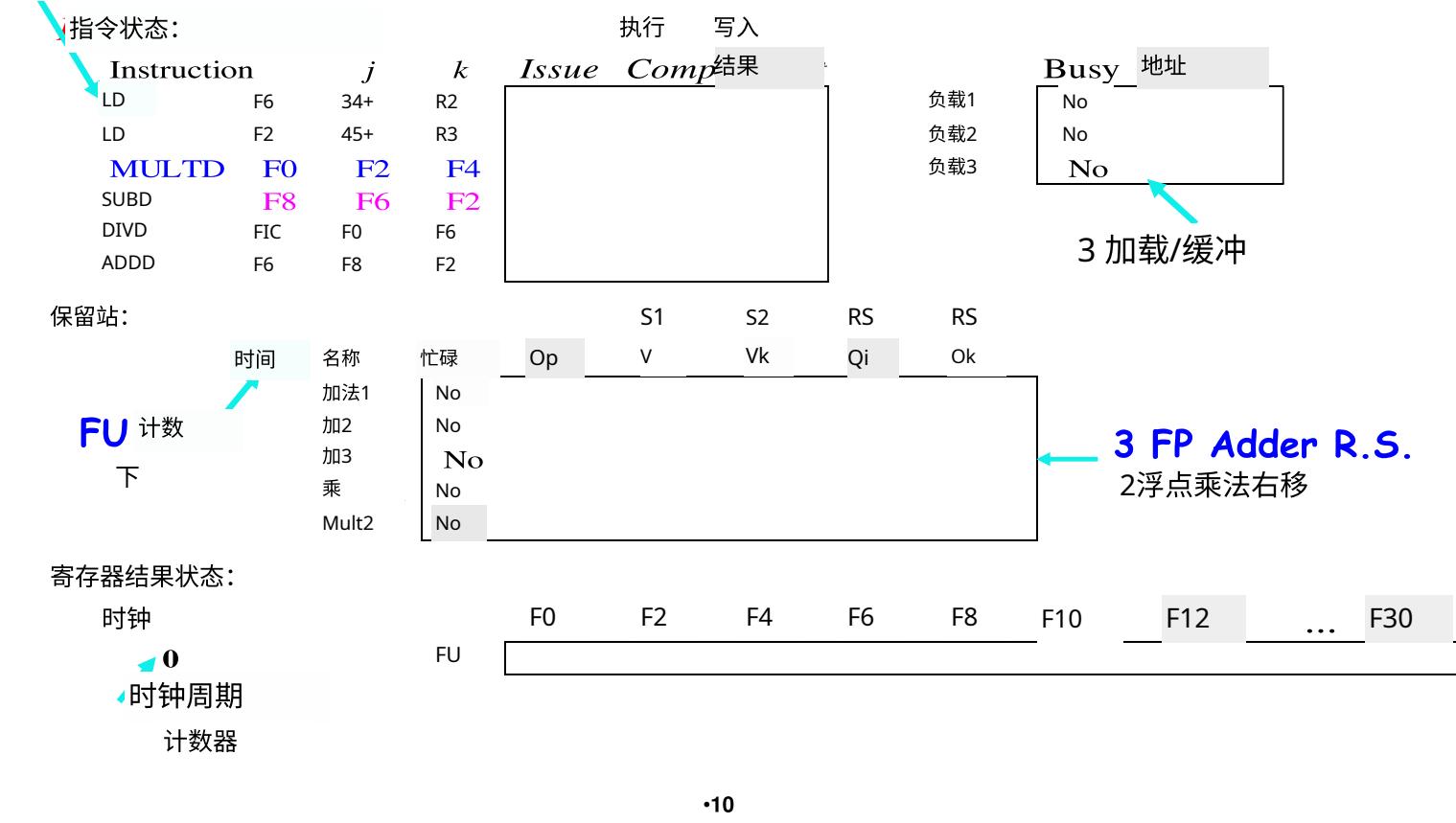
·7

·8

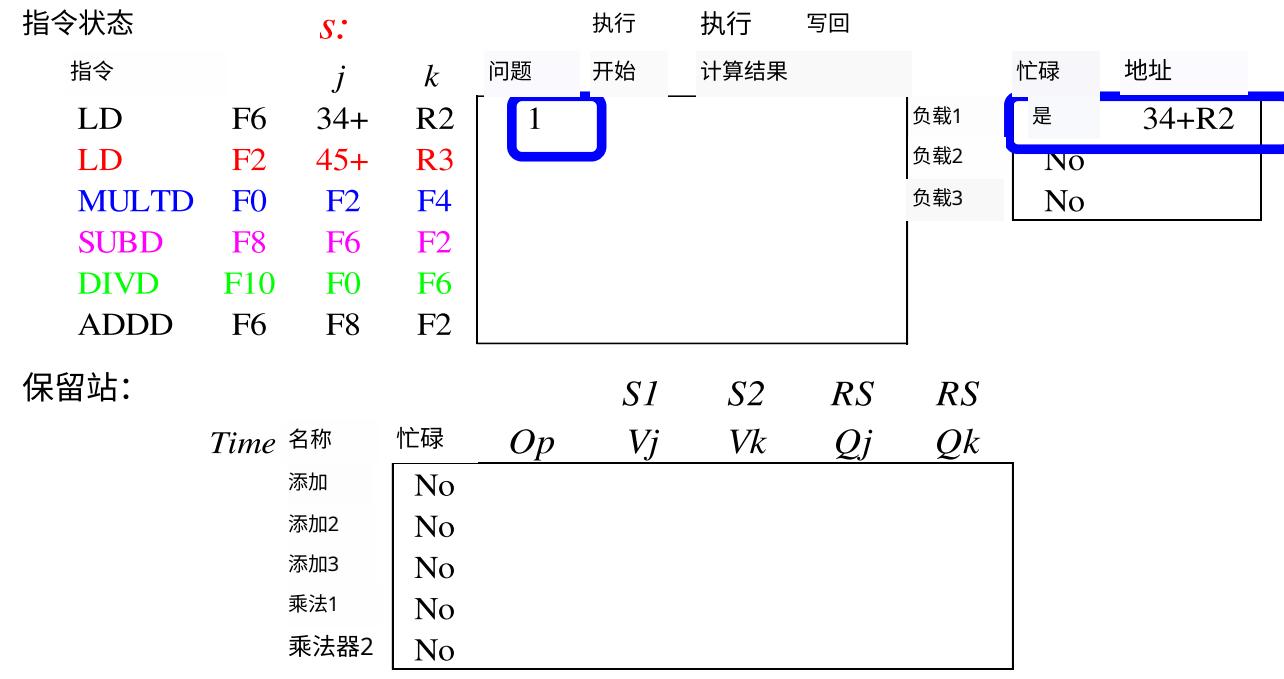
指令状态	等待直至	操作或登记
发布	站点 r 为空	若[寄存器状态[r].Q1=0] { 则 RS[r].Qi ← 寄存器状态 [rs].Qi } 否则 { RS[r].Vj ← 寄存器[r]; RS[r].Qj ← 0 }; 若[寄存器状态[r].Q1=0] { 则 RS[r].Qk ← 寄存器状态[r].Q1 } 否则 { RS[r].Vj ← 寄存器[r]; RS[r].Qk ← 0 }; RS[r].Busy ← 是;
加载或存储	缓冲区 r 为空	若[寄存器状态[r].Q1=0] { RS[r].Q1 ← 寄存器状态 [rs].Q1 } 否则 { RS[r].Vj ← 寄存器[r]; RS[r].Qj ← 0 }; RS[r].A ← 立即数; RS[r].Busy ← 是;
仅加载仅存储		寄存器状态[r].Qi=r;
		若[寄存器状态[r].Q1=0] { RS[r].Qk ← 寄存器状态 [rs].Qi } 否则 { RS[r].Vj ← 寄存器[r]; RS[r].Qk ← 0 };
执行	(RS[r].Oj=0) 且 浮点操作/加载存储 步骤1	计算结果: 操作数位于 Vj 和 Vk 中 RS[r].A ← RS[r].Vj + RS[r].A;
	加载步骤2	RS[r].A=0 从 Mem[RS[r].A] 读取
写入结果	浮点运算负载 or	执行完成于 r 且公共数据总线就绪 $\forall x \text{ (若 } (\text{RegisterStat}[x].Qi=r) \left\{ \text{Regs}[x] \leftarrow \text{结果}; \text{寄存器状态}[X].Q1 \leftarrow 0 \right. \right); \forall x \text{ (如 } (RS[x].Q1=r) \left\{ RS[x].V1 \leftarrow \text{result}; RS[x].Q1 \leftarrow 0 \right. \right) \forall x \text{ (若 } (RS[x].Qk=r) \left\{ RS[x].Vk \leftarrow \text{result}; RS[x].Qk \leftarrow 0 \right. \right); \text{寄存器堆}[r].忙碌标志} \leftarrow \text{否};$
存储		执行完成于 r & RS[r].Qk=0 存储器[RS[r].地址] ← RS[r].值; RS[r].忙 ← 否;

Tomasulo 算法示例

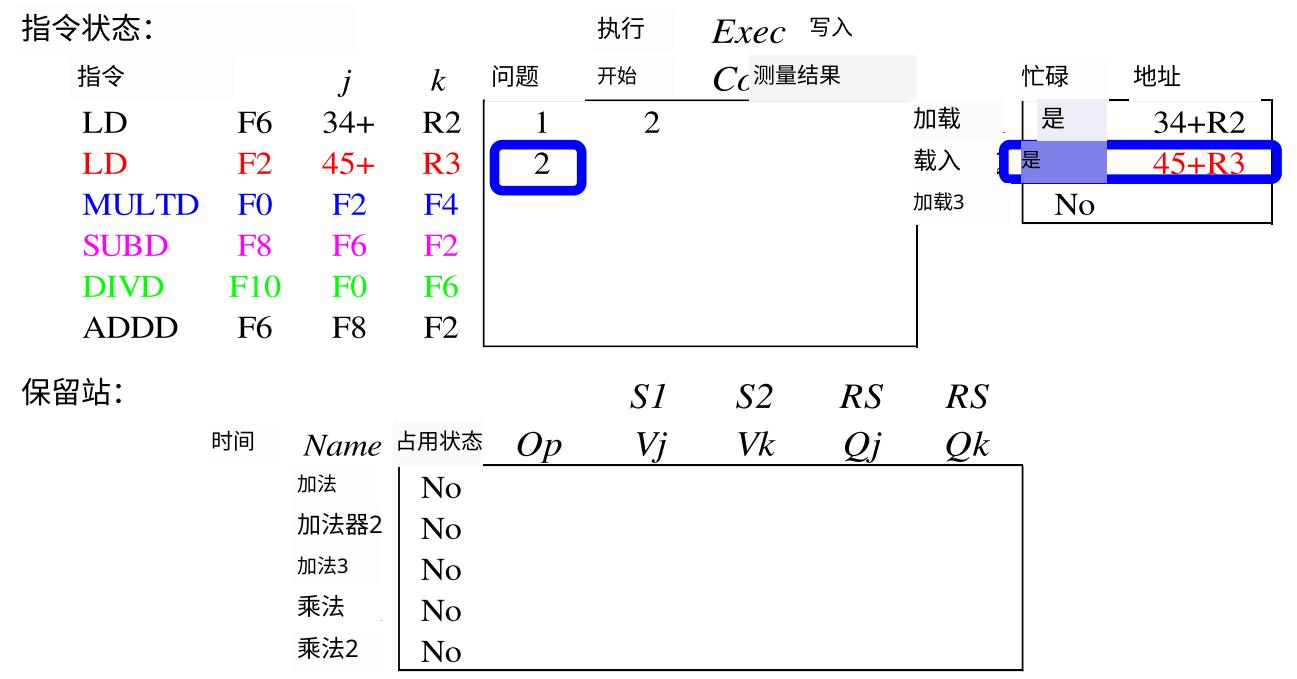
指令流



Tomasulo 算法示例周期1

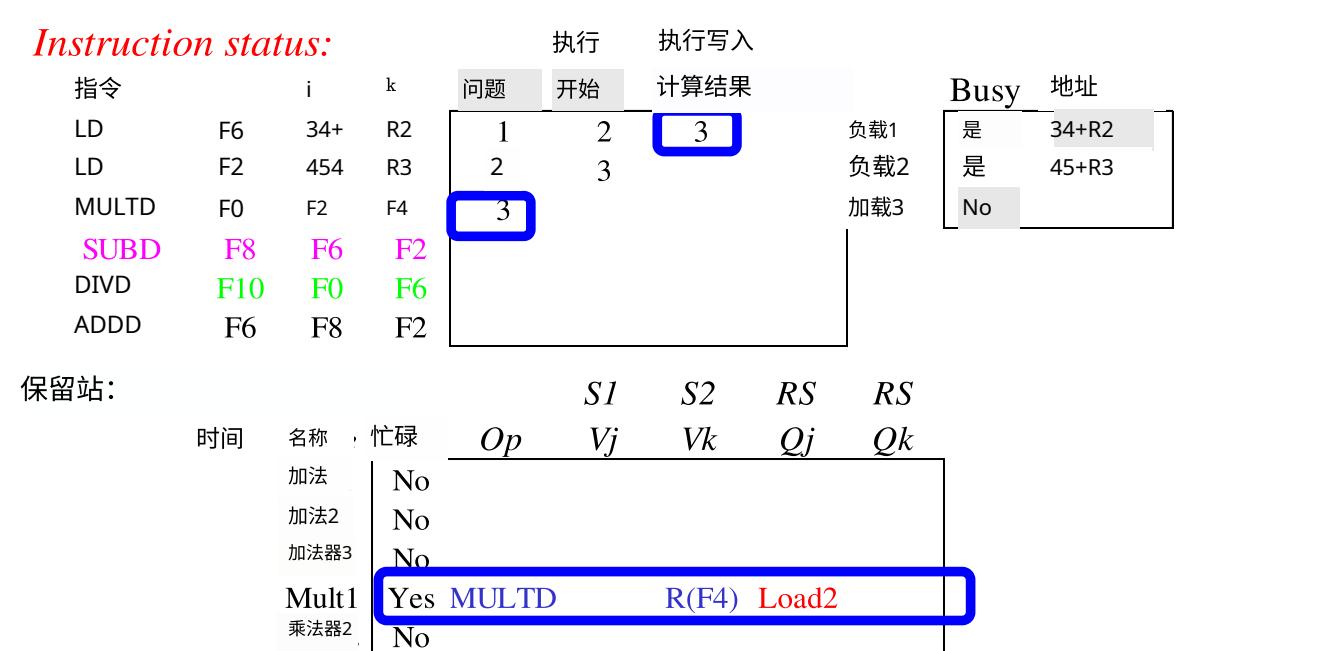


Tomasulo 算法示例周期2



注: 可存在多个未完成加载

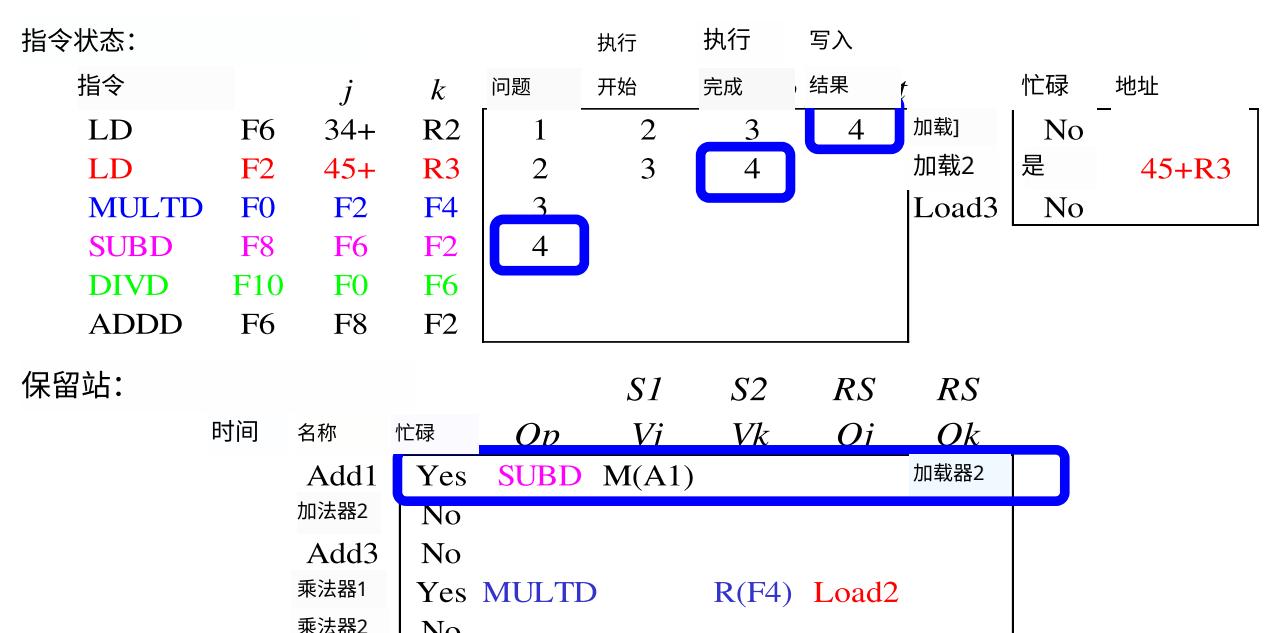
Tomasulo 算法示例周期3



•注: 寄存器名称在
保留站; 乘法指令已发射

•加载1完成; 哪些操作在等待加载1?

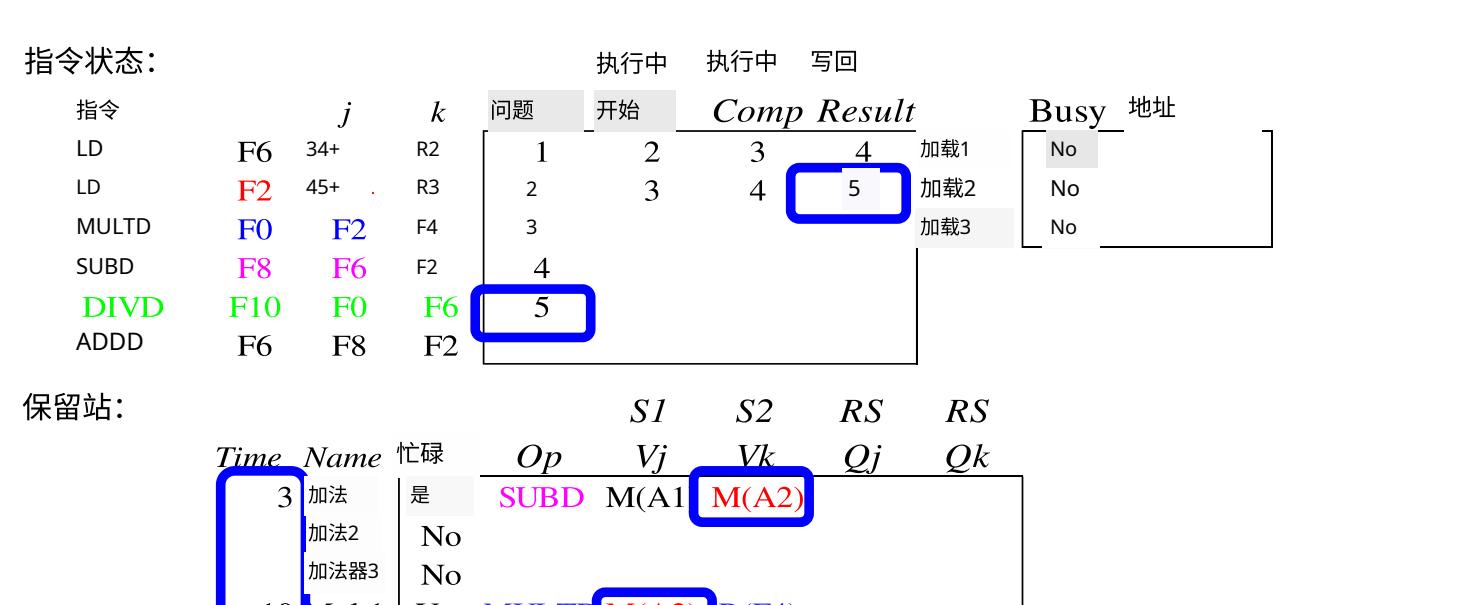
Tomasulo 算法示例周期4



•加载2完成; 哪些操作在等待加载2?

•14

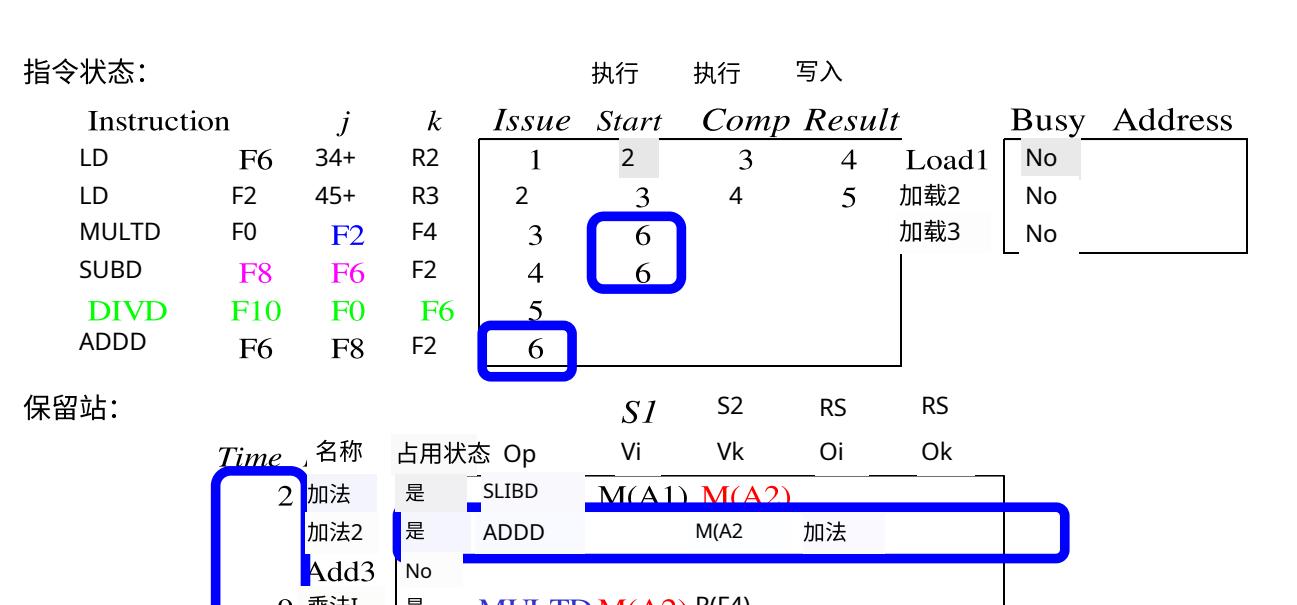
Tomasulo 算法示例周期5



•Add1和Mult1的计时器开始倒计时

•15

Tomasulo 算法示例周期6



•尽管F6存在名称依赖, 是否仍在此处发布ADDD指令?

•16

Tomasulo算法示例周期7

指令状态:	指令	j	k	执行开始				执行完成		写回结果	
				问题	开始	计算结果	结束	忙碌	地址		
	LD	F6	34+	R2	1	2	3	4	负载1	No	
	LD	F2	45+	R3	2	3	4	5	负载2	No	
	MULTD	F0	F2	F4	3	6			负载3	No	
	SUBD	F8	F6	F2	4	6					
	DIVD	F10	F0	F6	5						
	ADDD	F6	F8	F2	6						

保留站:

时间	名称	S1	S2	RS	RS
		忙碌	Op	Vj	Vk
1	加法器1	是	SUBD	M(A1)	M(A2)
2	加法器2	是	ADDD	M(A2)	添加
3	Add3	No			
4	乘数1	是	MULTD	M(A2)	R(F4)
5	乘数2	Yes	DIVD	M(A1)	乘数1

寄存器结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	乘法器1	M(A2)	加法器2	加法器1	乘法器2			

*17

Tomasulo算法示例周期9

指令状态:	Instruction	j	k	执行中				执行中		写回	
				Issue	start	Comp	Result	Busy	Address		
	LD	F6	34+	R2	1	2	3	4	Load1	No	
	LD	F2	45+	R3	2	3	4	5	加载2	No	
	MULTD	F0	F2	F4	3	6			加载3	No	
	SUBD	F8	F6	F2	4	6	8	9			
	DIVD	F10	F0	F6	5						
	ADDD	F6	F8	F2	6						

保留站:

Time	Name	S1	S2	RS	RS
		忙碌	Op	Vj	Vk
3 加2	加法	No			
4 加3	是 ADDD	(M-M)	M(A2)		
5 乘1	No				
6 乘2	Yes MULTD	M(A2)	R(F4)		
Mult2	Yes DIVD	M(A1)	Mult1		

注册结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(A2)	加法2	(M-M)	乘法2			

*19

Tomasulo算法示例周期11

指令状态:	指令	j	k	执行				执行		写入	
				下发	开始	计算结果	结束	忙碌	地址		
	LD	F6	34+	R2	1	2	3	4	负载1	No	
	LD	F2	45+	R3	2	3	4	5	负载2	No	
	MULTD	F0	F2	F4	3	6			负载3	No	
	SUBD	F8	F6	F2	4	6	8	9			
	DIVD	F10	F0	F6	5						
	ADDD	F6	F8	F2	6	10					

保留站:

Time	Name	S1	S2	RS	RS
		忙碌	Op	Vj	Vk
1 添加1	添加	No			
2 添加2	是 ADDD	(M-M)	M(A2)		
3 Add3	No				
4 多选1	Yes MULTD	M(A2)	R(F4)		
5 多选2	Yes DIVD	M(A1)	多任务1		

注册结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	乘数1	M(A2)	(M-M+N(M-M))	乘法单元2				

*21

Tomasulo算法示例周期13

指令状态:	指令	j	k	执行中				执行中		写入	
				问题	开始	计算结果	结束	忙碌	地址		
	LD	F6	34+	R2	1	2	3	4	负载1	No	
	LD	F2	45+	R3	2	3	4	5	负载2	No	
	MULTD	F0	F2	F4	3	6			负载3	No	
	SUBD	F8	F6	F2	4	6	8	9			
	DIVD	F10	F0	F6	5						
	ADDD	F6	F8	F2	6	10	12	13			

保留站:

Time	Name	S1	S2	RS	RS
		忙碌	Op	Vj	Vk
Add1	Add2	No			
加3	No				
2乘1	Yes MULTD	M(A2)	R(F4)		

Tomasulo算法示例周期15

指令状态:	指令	j	k	执行				写回	忙碌	地址
				开始	组件	结果	问题			
	LD	F6	34+	R2	1	2	3	4	负载1	No
	LD	F2	45+	R3	2	3	4	5	负载2	No
	MULTD	F0	F2	F4	3	6	15	16	负载3	No
	SUBD	F8	F6	F2	4	6	8	9		
	DIVD	F10	F0	F6	5					
	ADD	F6	F8	F2	6	10	12	13		

Reservation Stations:

时间	名称	繁忙	S1		S2		RS		RS	
			Op	Vj	Vk	Qj	Qk			
	加法器1	No								
	Add2	No								
	Add3	No								
0	Mult1	Yes	MULTD	M(A2)	R(F4)					
	乘法器2	是	DIVD		M(A1)	乘法器1				

寄存器结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	乘法器1	M(A2)	(M-M+N(M-M))			?		

•Mult1 (MULTD)运算完成; 后续等待什么?

-25

Tomasulo算法示例第17周期

指令状态:	指令	j	k	执行中				写入	忙碌	地址
				发布	启动	完成	结果			
	LD	F6	34+	R2	1	2	3	4	Load1	No
	LD	F2	454	R3	2	3	4	5	加载2	No
	MULTD	F0	F2	F4	3	6	15	16	Load3	No
	SUBD	F8	F6	F2	4	6	8	9		
	DIVD	F10	F0	F6	5	17	56			
	ADD	F6	F8	F2	6	10	12	13		

保留站:

Time	Name	Busy	S1		S2		RS		RS	
			Op	Vj	Vk	Qj	Qk			
	加法	No								
	加法2	No								
	加法3	No								
	乘法1	No								
39	乘法2	Yes	DIVD	M*F4	M(A1)					

寄存器结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
17	FU	M*F4	M(A2)	(M-M+V(M-M))	乘法器2				

-27

Tomasulo算法示例周期56

指令状态:	指令	j	k	执行开				写回	忙碌	地址
				开始	组件	结果	问题			
	LD	F6	34+	R2	1	2	3	4	负载1	No
	LD	F2	45+	R3	2	3	4	5	负载2	No
	MULTD	F0	F2	F4	3	6	15	16	负载3	No
	SUBD	F8	F6	F2	4	6	8	9		
	DIVD	F10	F0	F6	5	17	56			
	ADD	F6	F8	F2	6	10	12	13		

保留站:

时间	名称	忙碌	Op	Vj	Vk	Qj	Qk
	Add1	No					
	添加2	No					
	添加3	No					
	乘1	No					
	乘2	是	DIVD	M*F4	M(A1)		

寄存器结果状态:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)	(M-M+M(M-M))	乘法器2				

•乘法器2 (DIVD) 正在完成; 它在等待什么?

-29

Tomasulo算法具有三大优势

一、风险检测逻辑的分布

- 分布式保留站与公共数据总线
- 若多条指令等待同一结果且各指令均具备另一操作数，则可通过公共数据总线广播实现指令同时释放
- 若采用集中式寄存器堆，功能单元须在寄存器总线可用时从中读取结果

二、消除WAW与WAR风险的停顿

Tomasulo算法示例周期16

指令状态:	指令	j	k	执行				写回	忙碌	地址
				开始	组件	结果	计算结果			
	LD	F6	34+	R2	1	2	3	4	负载1	No
	LD	F2	45+	R3						

为何Tomasulo算法能实现循环迭代重叠?

□ 寄存器重命名

- > 多轮迭代使用不同的物理寄存器目标 (动态循环展开)

□ 保留站机制

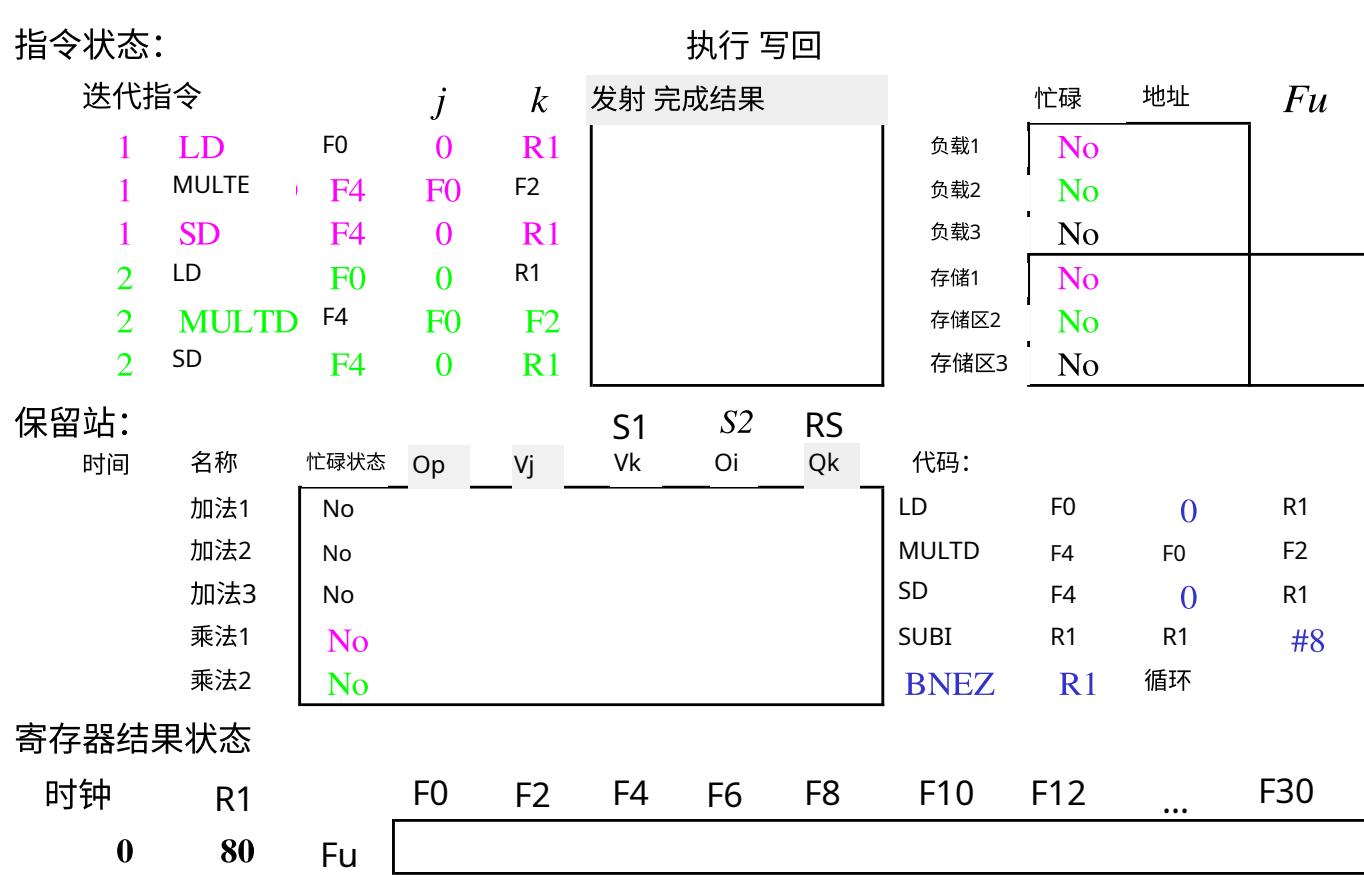
- > 允许指令发射越过整数控制流操作

> 同时缓冲寄存器的旧值——完全规避了记分牌中的WAR冲突停顿

□ 另一种视角: Tomasulo算法动态构建数据流依赖图。

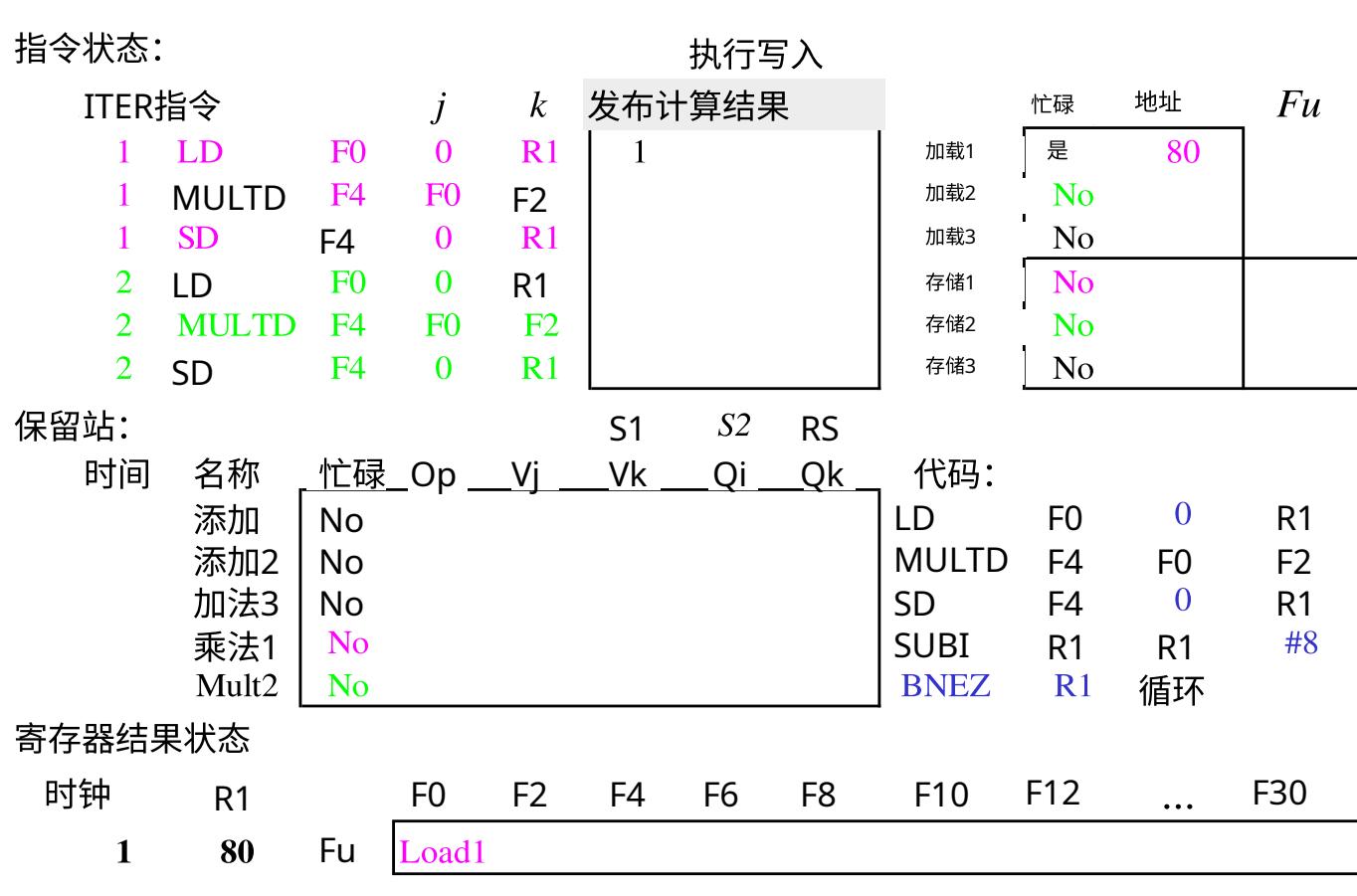
.33

循环示例



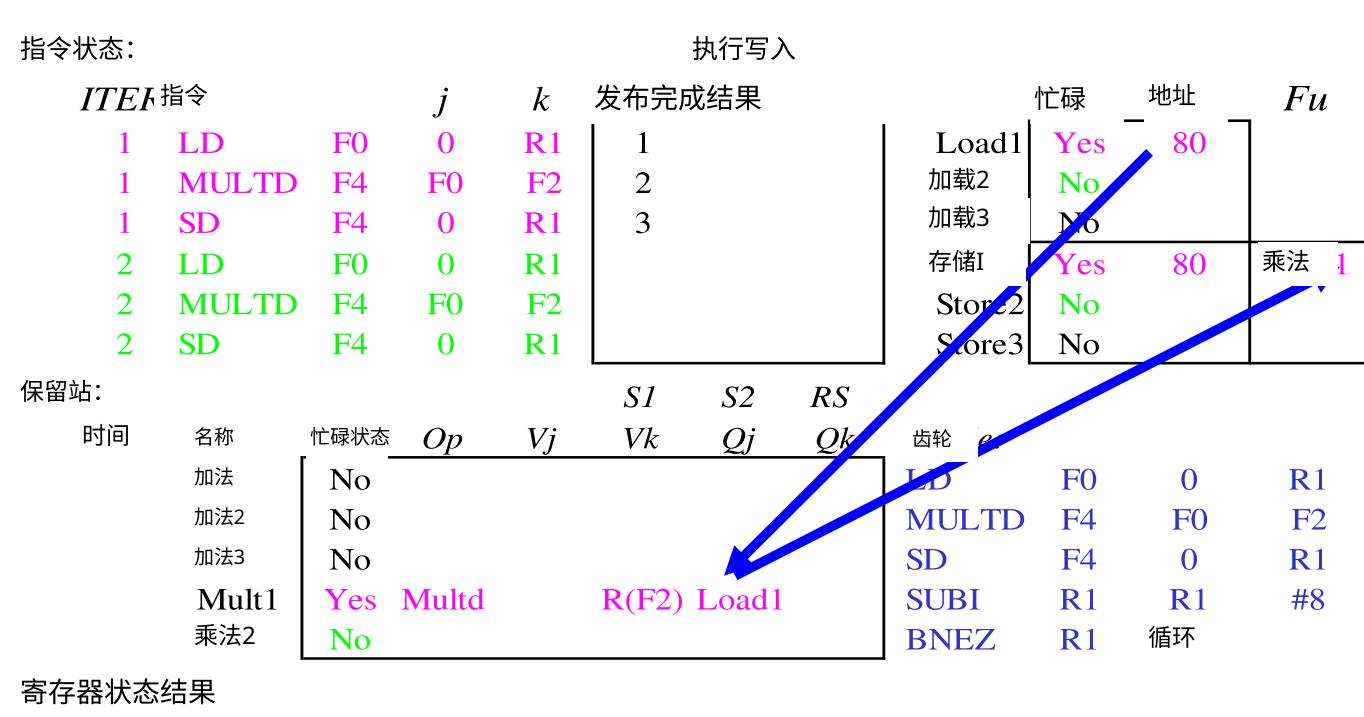
.35

循环示例周期1



.37

循环示例周期3



.39

□ Implicit renaming sets up ‘数据流’ 图表

Tomasulo算法实现循环迭代重叠

□ 寄存器重命名

- > 多轮迭代使用不同的物理寄存器目标 (动态循环展开)

□ 保留站

- > 允许指令发射越过整数控制流操作 > 同时缓冲寄存器的旧值——彻底消除记分牌中出现的写后读冲突停顿

□ 另一视角: Tomasulo算法动态构建数据流依赖图。

.34

Tomasulo循环示例

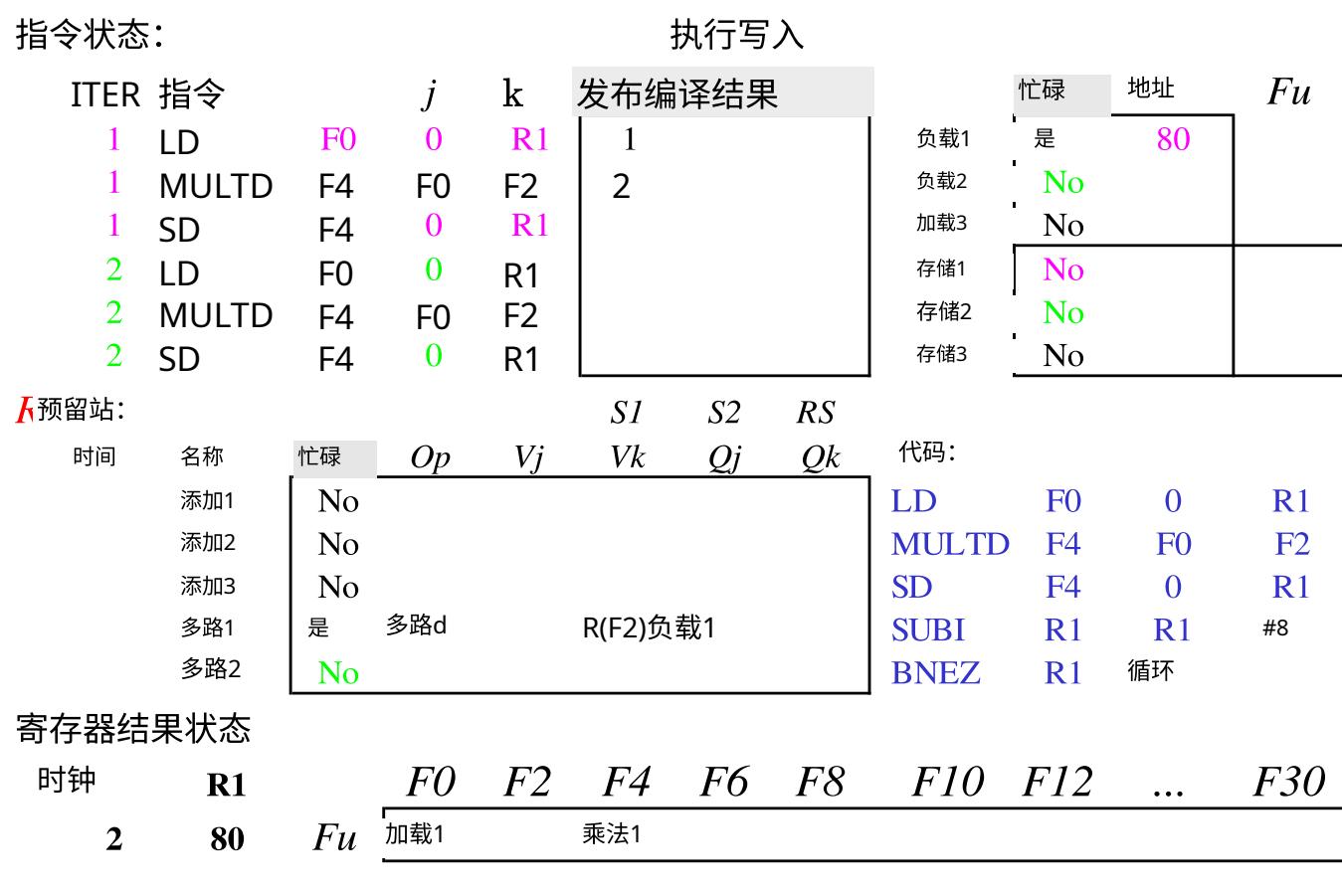
循环:	LD	F0	0	R1
	MULTD	F4	F0	F2
	SD	F4	0	R1
	SUBI	R1	R1	#8
	BNEZ	R1	循环	

□ 假设乘法运算占用4个时钟周期

□ 假设首次加载耗时8个时钟周期 (缓存未命中), 二次加载耗时1个时钟周期 (命中) □ 需明确说明的是, 将展示SUBI和BNEZ指令的时钟周期 □ 实际情况: 前方存在整数指令

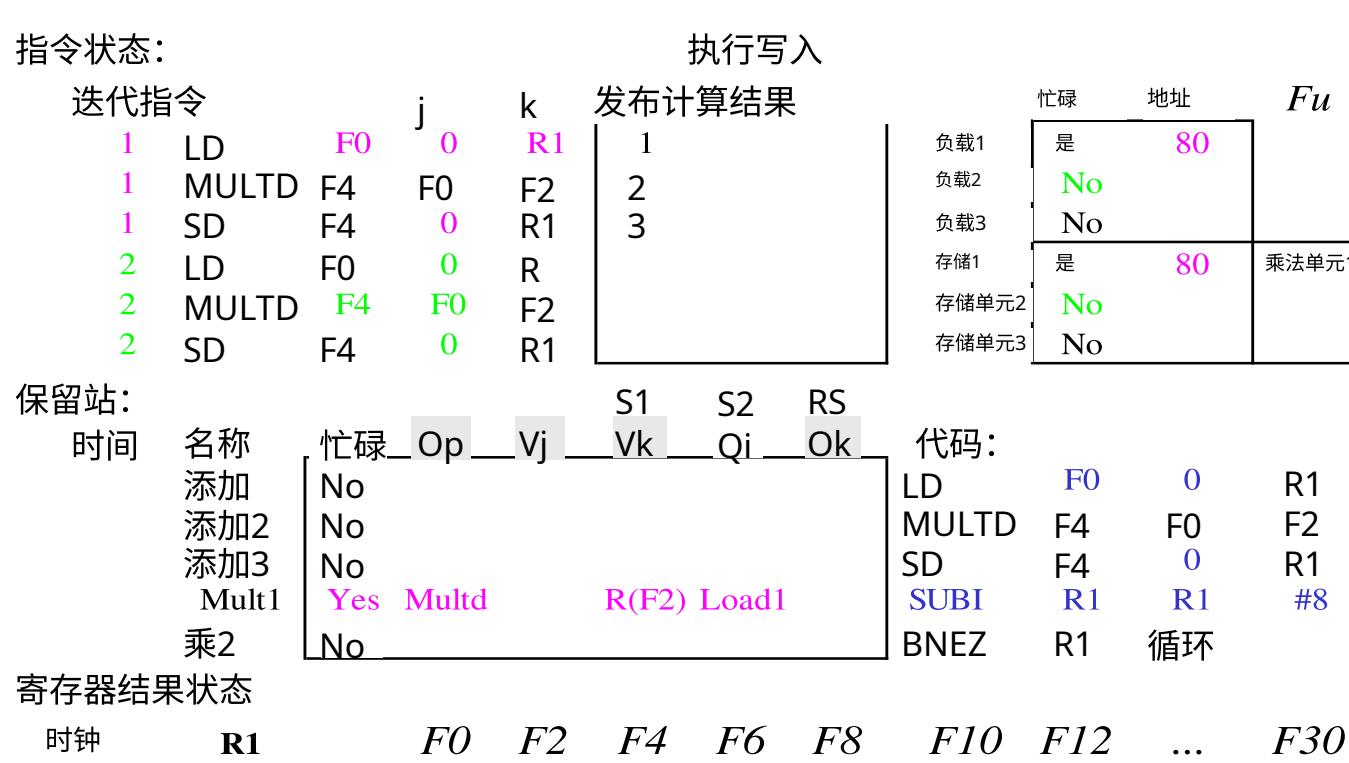
.36

循环示例周期2



.38

循环示例周期4



□ 正在分派SUBI指令

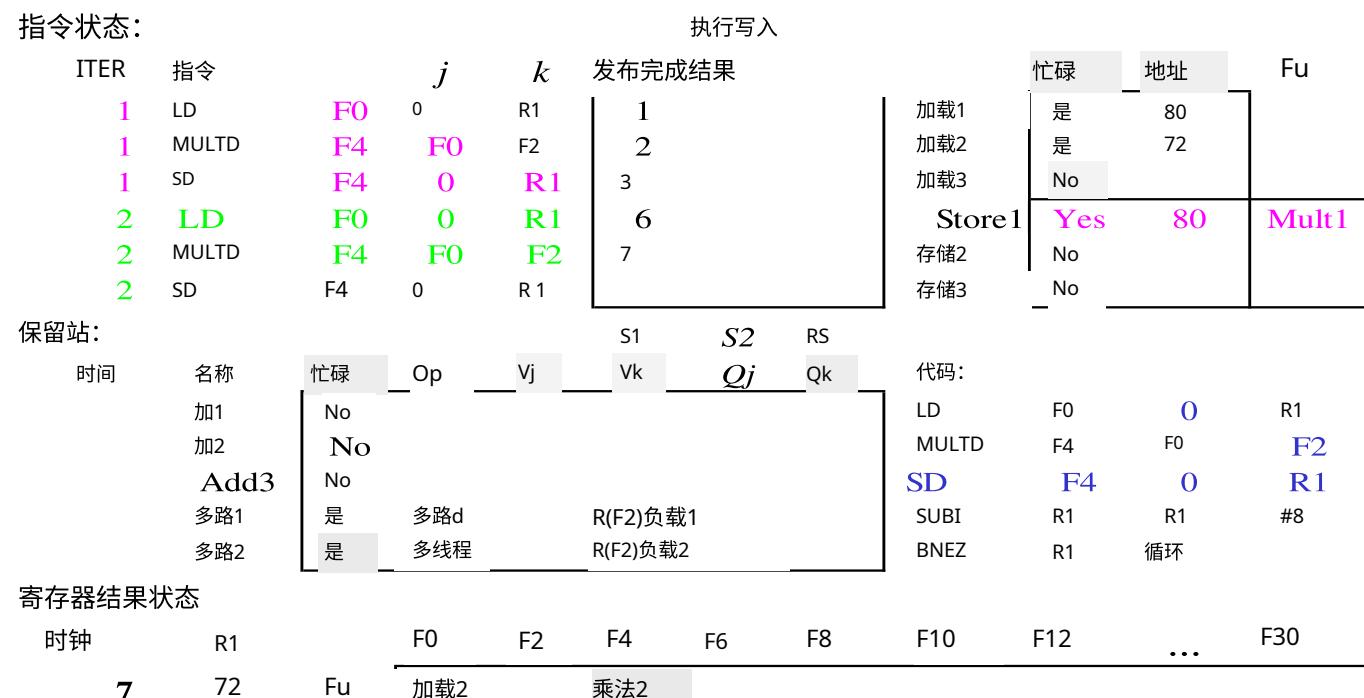
循环示例周期5



□ 并且, BNEZ指令

*41

循环示例周期7

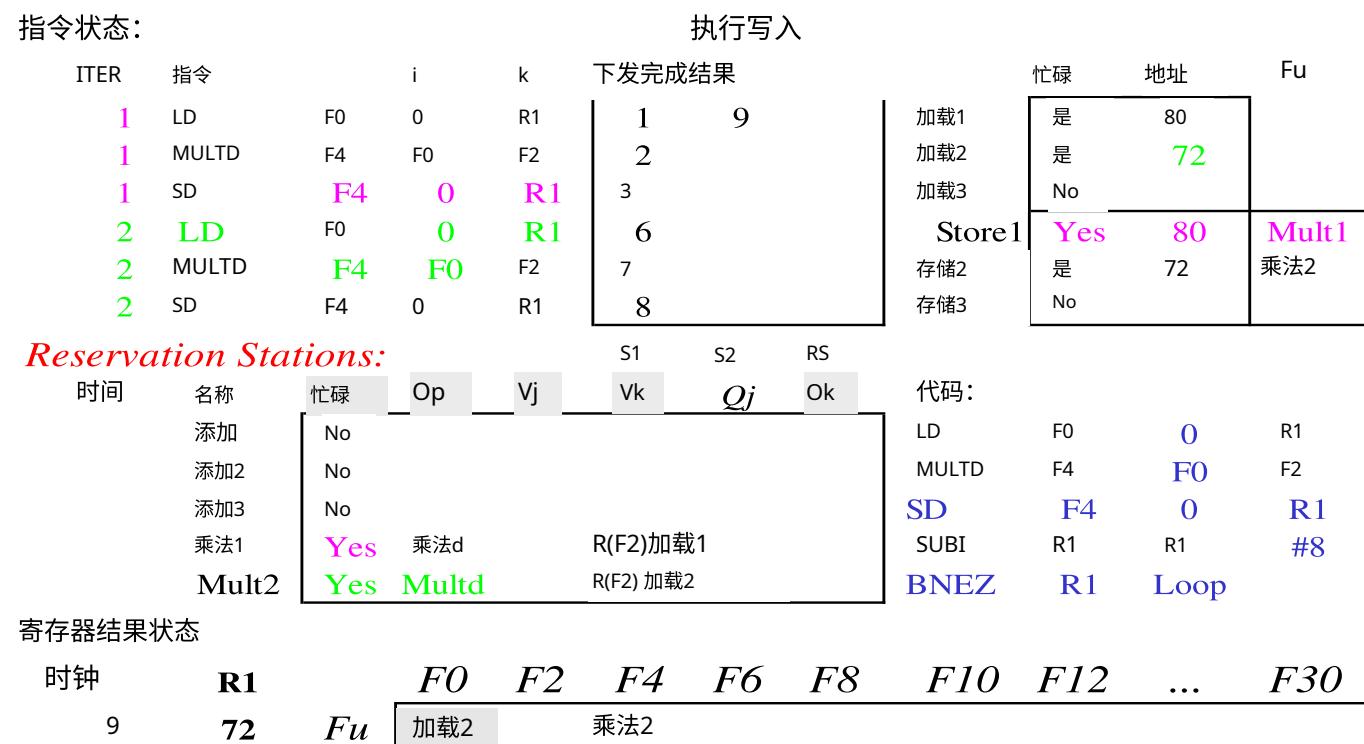


□ 寄存器文件完全脱离计算

□ 第一次与第二次迭代完全重叠

*43

循环示例周期9

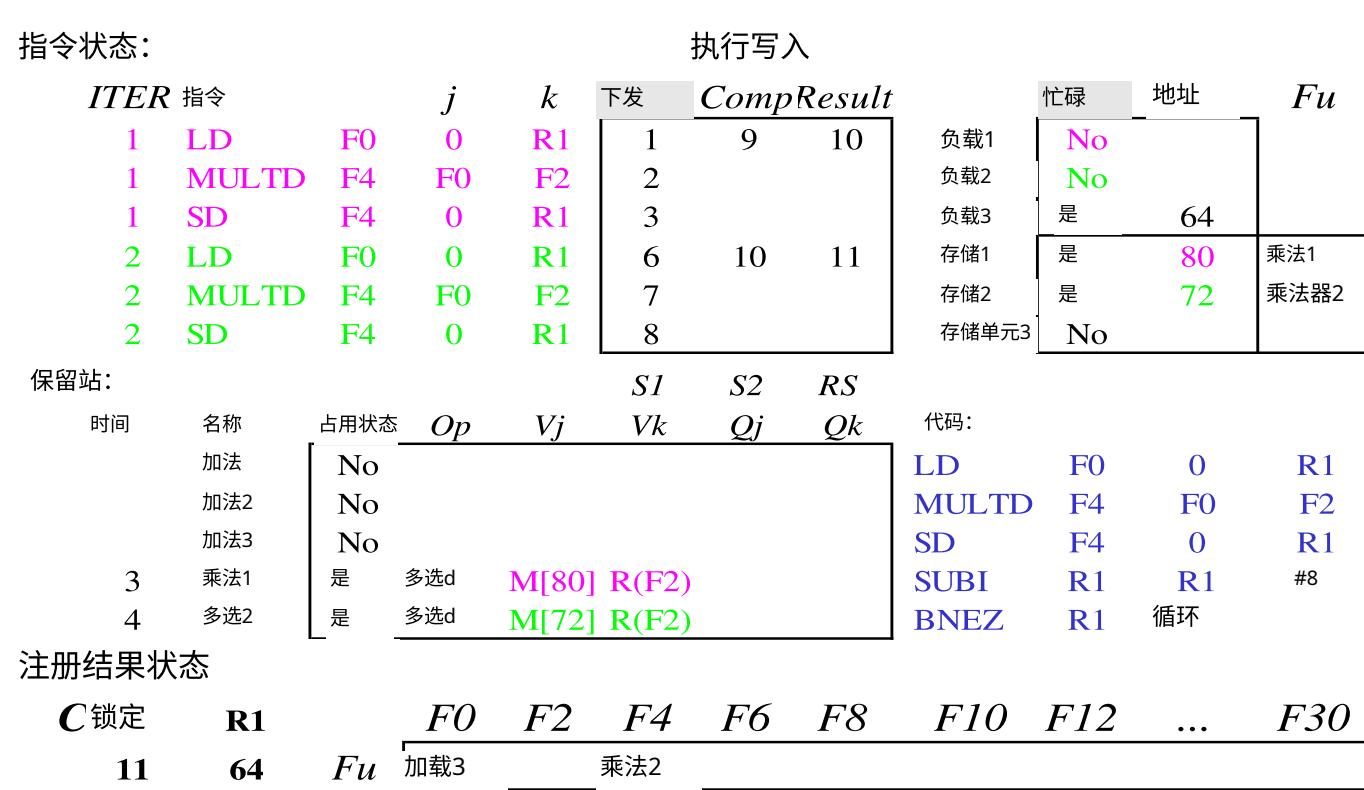


□ Moad1完成: 谁在等待?

□ 注: 正在分派SUBI指令

*45

循环示例周期11



□ 序列中的下一个加载项

*47

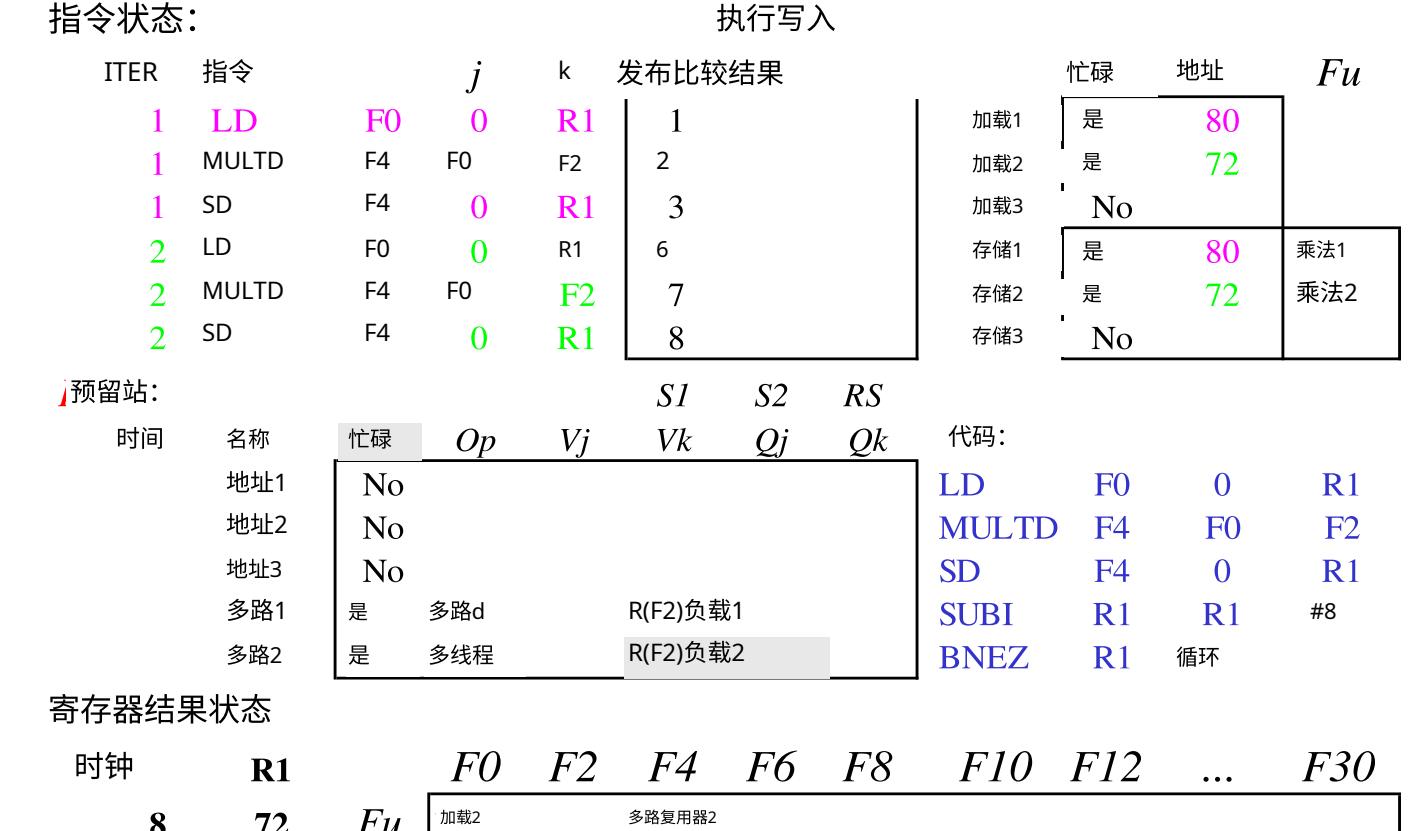
循环示例周期6



□ 注意F0从未看到从地址80加载的数据

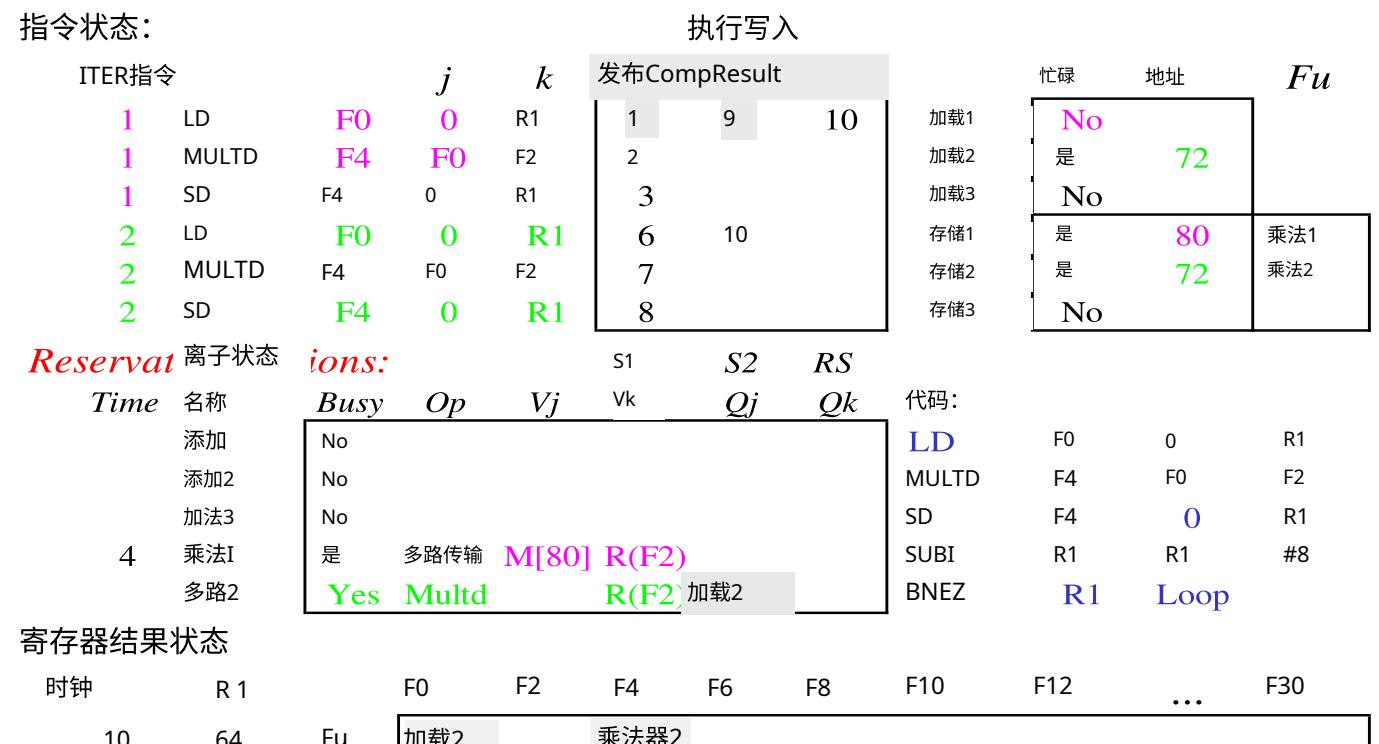
*42

循环示例周期8



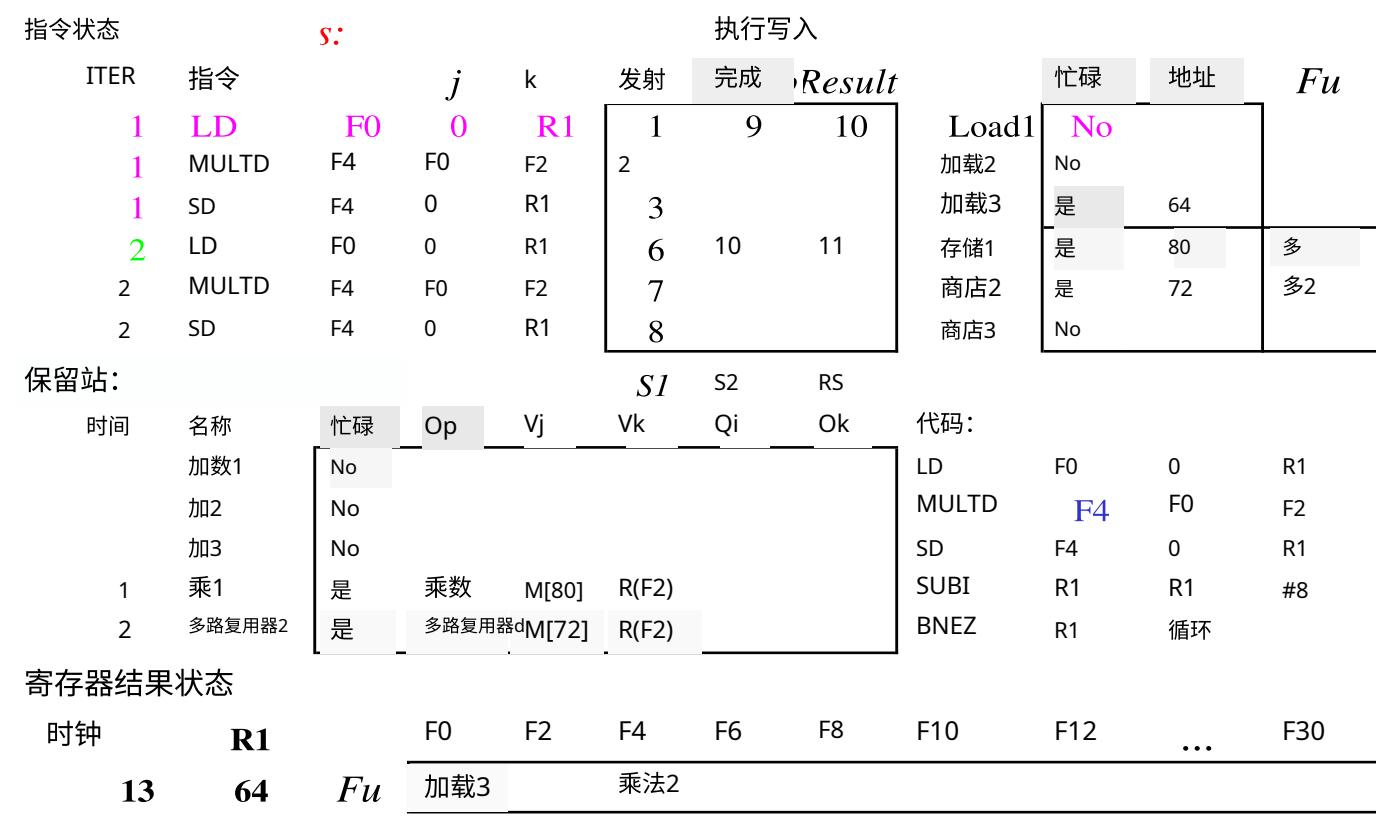
*44

循环示例周期10

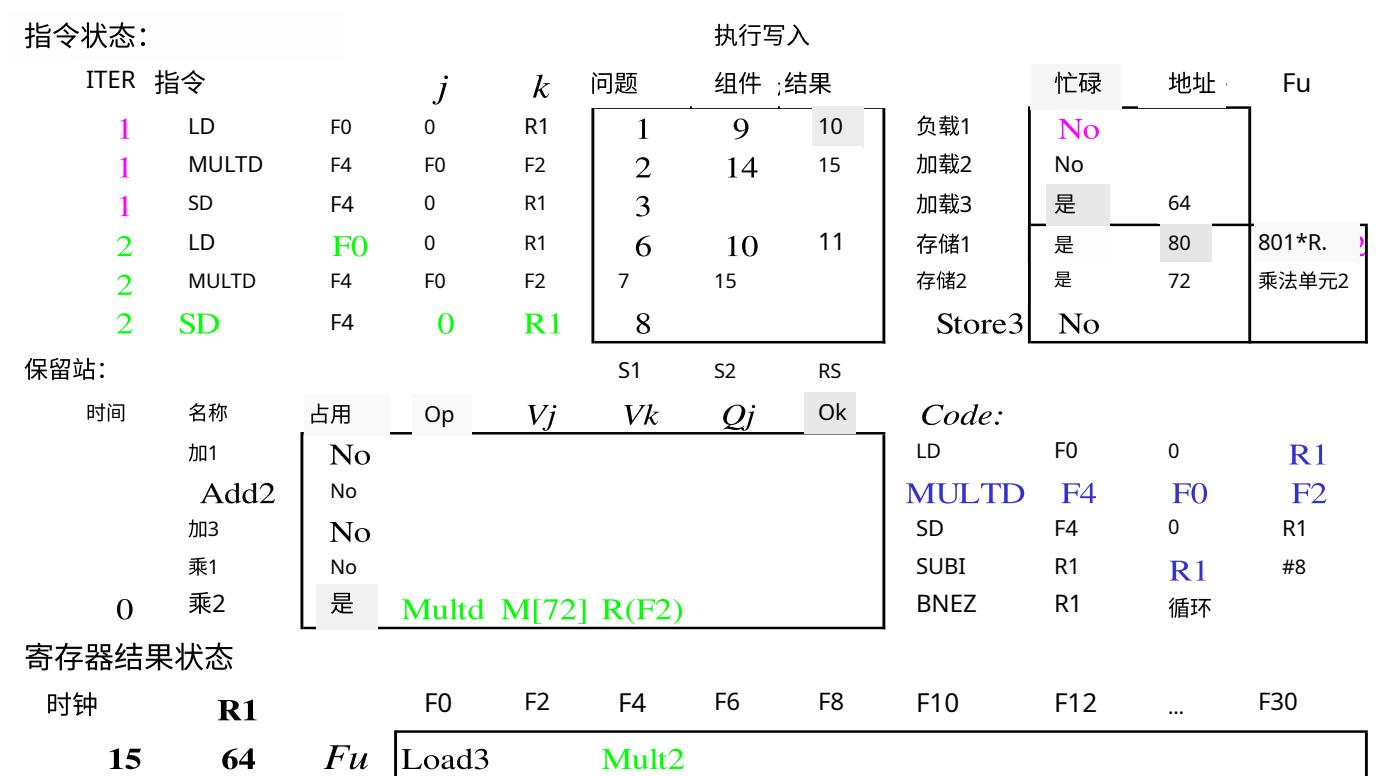


□ Moad2完成: 谁在等待?

循环示例周期13



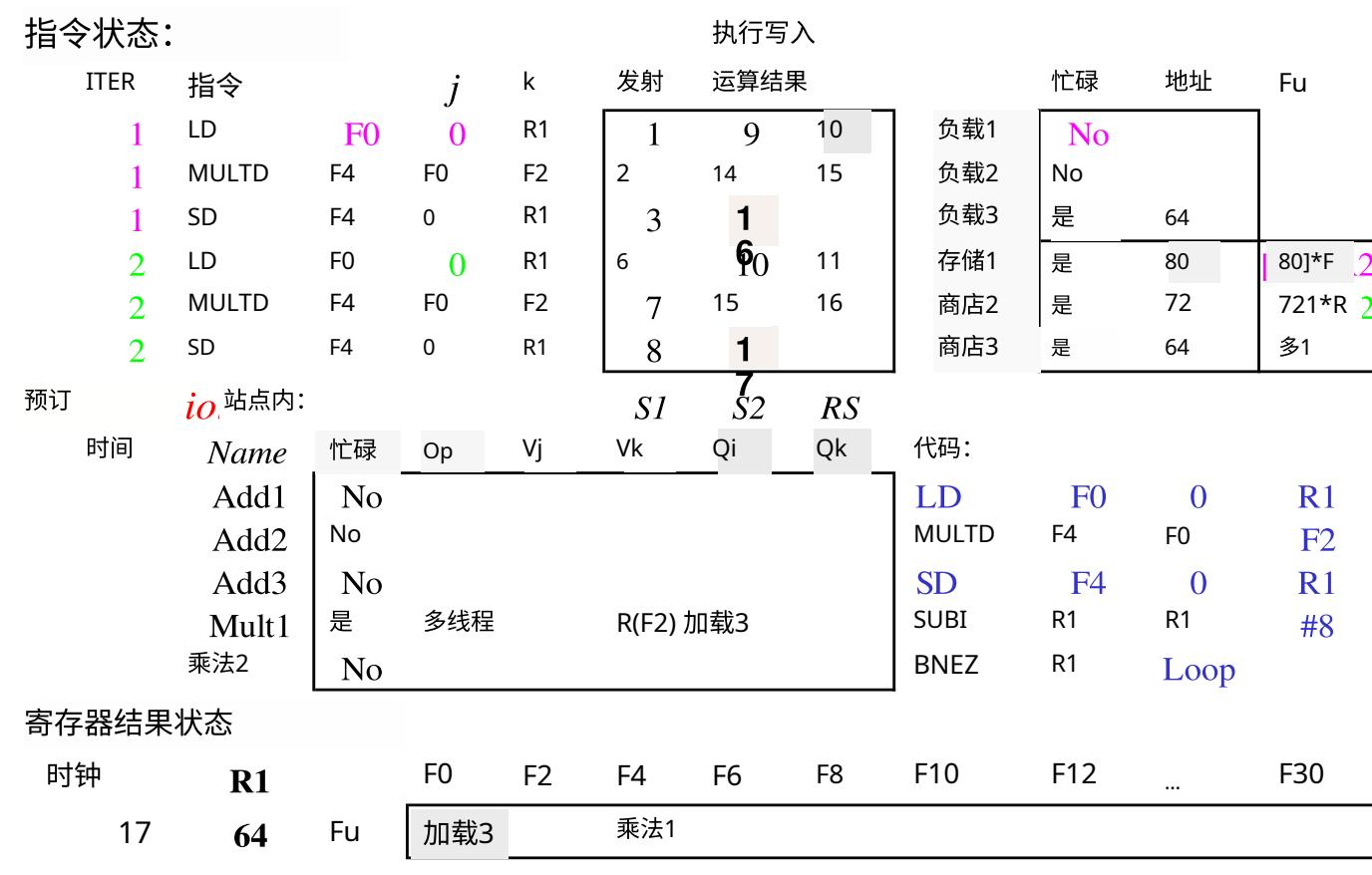
循环示例周期15



□ Mult2运算完成。谁在等待?

-51

循环示例周期17



精确中断如何实现?

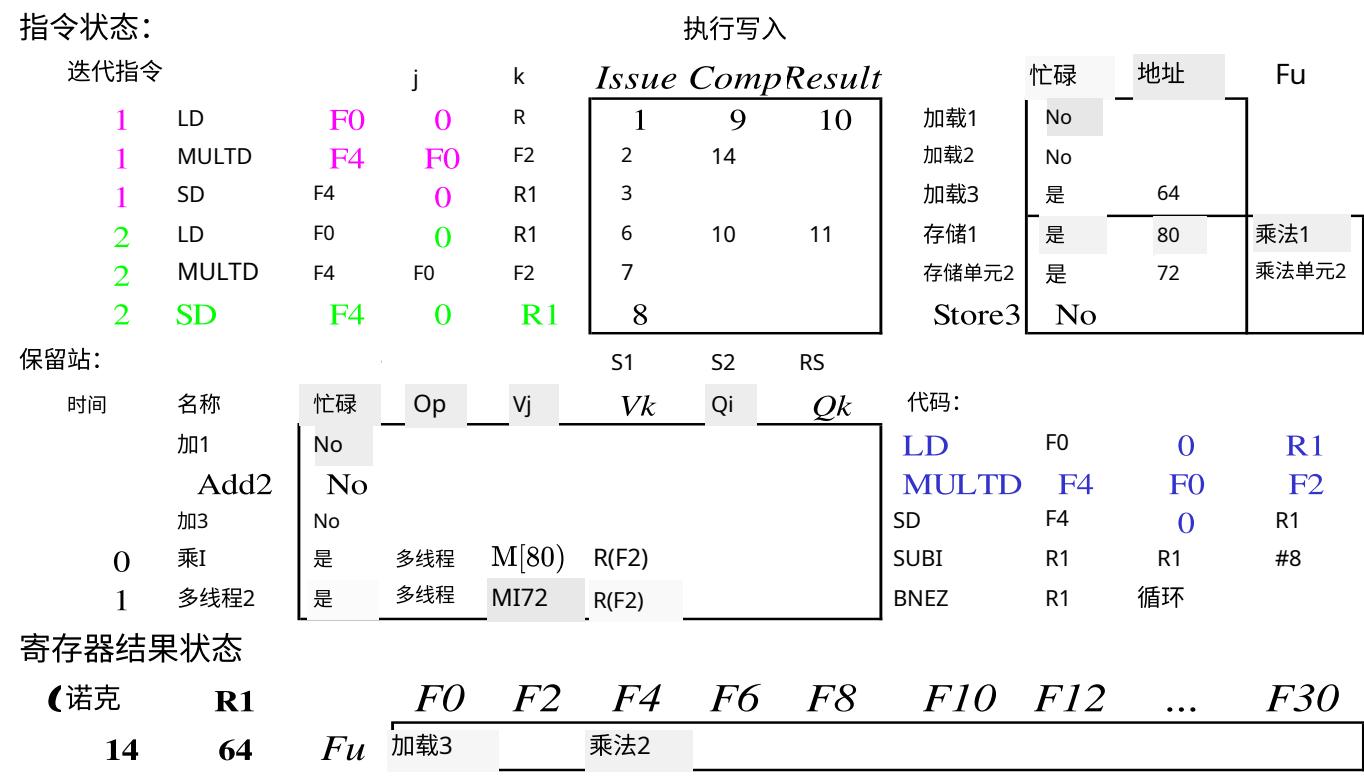
□ Tomasulo算法包含:

顺序发射、乱序执行与乱序完成

└ 需修正乱序完成机制，以便在指令流中定位精确断点。

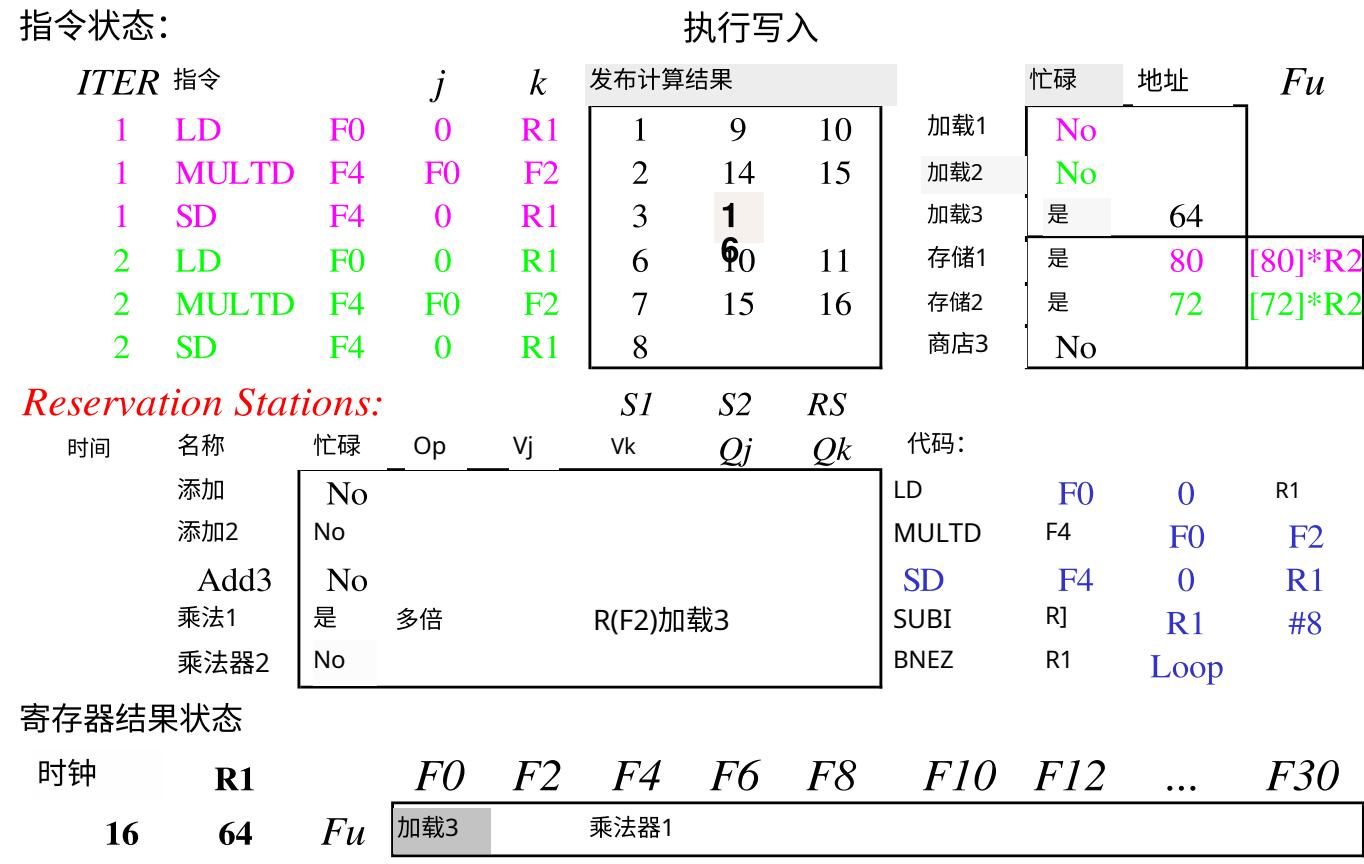
→ 推测执行与重排序缓冲区！（后续详解）

循环示例周期14



□ 乘法1完成中。等待者是谁? 50

循环示例周期16



Tomasulo算法概要

□ 保留站: 通过隐式寄存器重命名扩展寄存器组 + 缓冲源操作数
 > 消除寄存器瓶颈
 > 规避记分牌的写后读、写后写冲突
 > 支持硬件级循环展开
 □ 不限于基本块
 > (整数运算单元可超前执行, 跨越分支)
 混合贡献
 > 动态调度
 > 寄存器重命名
 > 加载/存储消歧
 J360/91的后裔包括奔腾III、PowerPC 604、MIPS R10000、HP-PA 8000和Alpha 21264

-52

记分板与Tomasulo算法对比

□ 特点
 > 乘法器等函数
 > 按序发射, 完成OOO
 > 若 → 发射, Ro
 > 4 级流水线
 > 记分牌集中控制
 缺点
 > WAW, WAR冲突时停顿
 > 功能单元少, 非流水线
 > 顺序发射, 乱序完成
 > 浮点操作队列、保留站、加载/存储缓冲器、公共数据总线
 > 三级流水线
 > 寄存器重命名 → 无写后写、读后写冲突
 > 降低结构冒险
 > 真数据冒险检测分散式预留
 > CDB → 前递路径

-计分板能否通过寄存器重命名避免WAW和WAR冲突?

-53

53

计分板流水线阶段描述

□ 发射阶段：当满足以下条件时指令被发射

- > 功能单元可用且
- > 无其他活跃指令占用相同目标寄存器
- > 避免结构冲突与写后写冲突

□ 读取操作数阶段(RD)

- > 读取操作将延迟至两个操作数均就绪时执行
- > 这意味着先前已发射但未完成的指令未将该操作数作为目标地址
- > 此举可动态解决RAW数据冲突

□ 执行阶段(EX)

- > 完成后通知记分板以释放功能单元
- 重复使用

□ 写入结果(WB)

- > 记分牌检测WAR冲突并在必要时暂停完成中的指令
- 指令

-57

显式寄存器重命名

□ 使用比ISA规定寄存器数量更大的物理寄存器文件

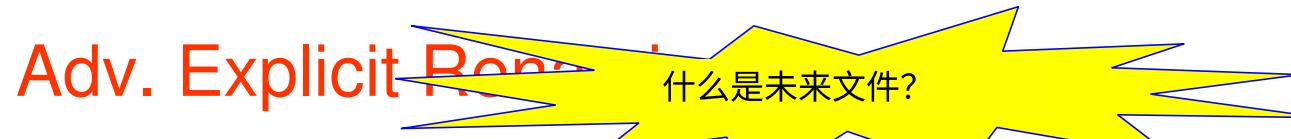
□ 关键洞见：为每条写入指令分配新的物理目标寄存器

- > 与静态单赋值形式(SSA)的编译器转换极其相似——但这是在硬件层面实现！
- > 彻底消除WAR或WAW冒险的可能性
- > 类似Tomasulo算法，能实现完全的乱序完成
- > 类似基于硬件的动态编译？

□ 机制？维护转换表：

- > ISA寄存器 \Rightarrow 物理寄存器映射
- > 寄存器写入时，用空闲列表的新寄存器替换条目
- > 当物理寄存器不被任何活跃指令使用时即释放

-59



[多款处理器采用此技术变体：>R10000、Alpha 21264、HP PA8000]

□ 获取精确中断点的另一种方法：

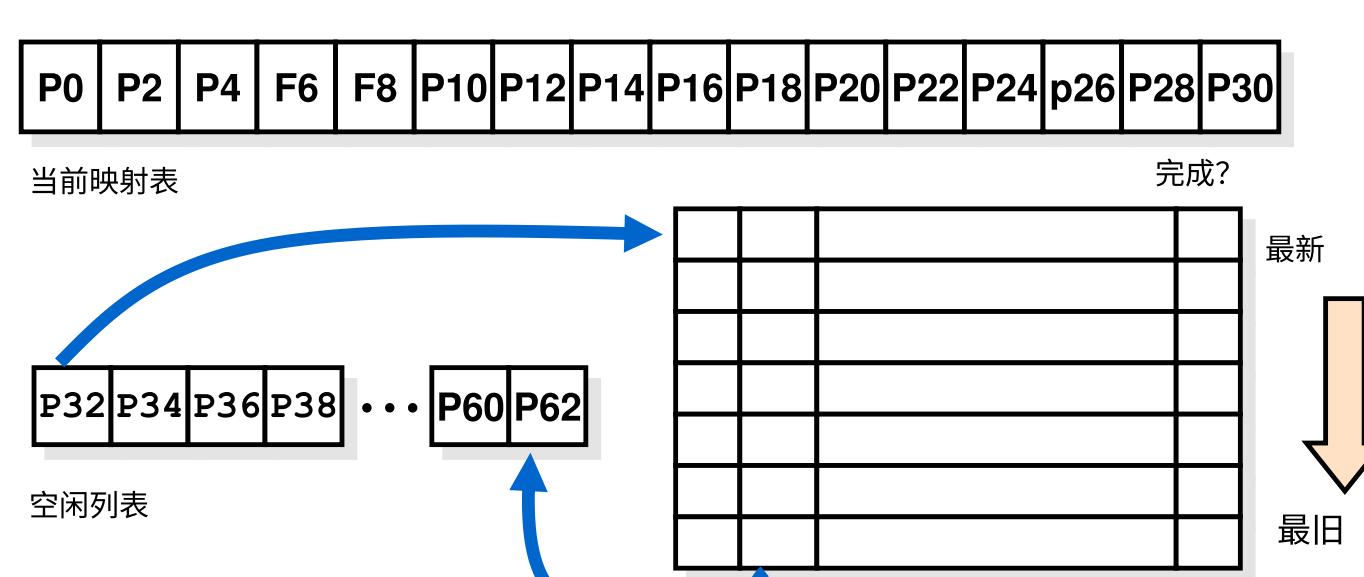
要实现精确断点，只需撤销表映射即可“回退”所有操作

> 在重排序缓冲区和未来文件之间提供了有趣的混合机制

- 结果会立即写回寄存器文件
- 寄存器名称按程序顺序（由ROB）“释放”

-61

显式寄存器重命名： (R1000风格)



物理寄存器文件大于ISA寄存器文件
□ 在指令发射时，每条修改寄存器的指令都会从空闲列表中分配新的物理寄存器

记分板算法

□ 记分板全面负责指令的

发射与执行

> 创建依赖记录

> 决定何时获取操作数

> 决定何时进入执行阶段

> 决定何时可将结果写入寄存器文件

□ 三种数据结构

> 指令状态：

- 指令当前处于四个步骤中的哪一步

>> 功能单元状态：忙碌、操作码、 F_i 、 F_j 、 F_k 、 Q_j 、 Q_k 、 R_j 、 R_k

> 寄存器结果状态：

- 由哪个功能单元写入该寄存器

-58

显式重命名的优势

□ 将重命名与调度解耦：

» 流水线可完全仿效“标准”MIPS架构（可能支持每周期多发操作）

> 亦或采用类Tomasulo算法或记分牌等架构

> 可采用标准转发或旁路机制

□ 允许从单一寄存器文件读取数据

> 无需从保留站或重排序缓冲区旁路数值

> 这对流水线平衡至关重要

-60

显式重命名支持包括：

□ 快速访问转译表

□ A 物理寄存器文件，其寄存器数量超出指令集架构(ISA)规定

□ 能够识别哪些物理寄存器处于空闲状态

> 无可用寄存器 \Rightarrow 发射阶段停顿

□ 因此，寄存器重命名无需保留站。但需注意：

> 许多现代架构采用显式寄存器重命名+类Tomasulo保留站来控制执行

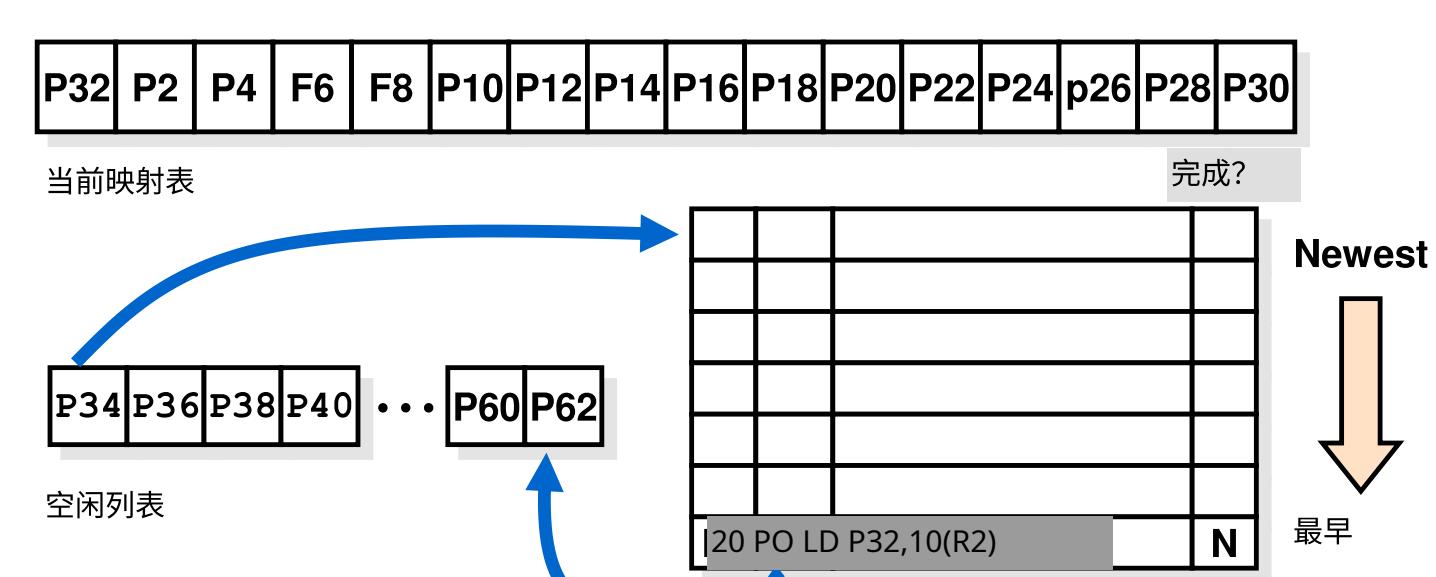
□ 两个关键问题：

> 如何管理“空闲列表”？

> 显式寄存器重命名如何与精确中断相结合？

-62

显式寄存器重命名： (R1000风格)



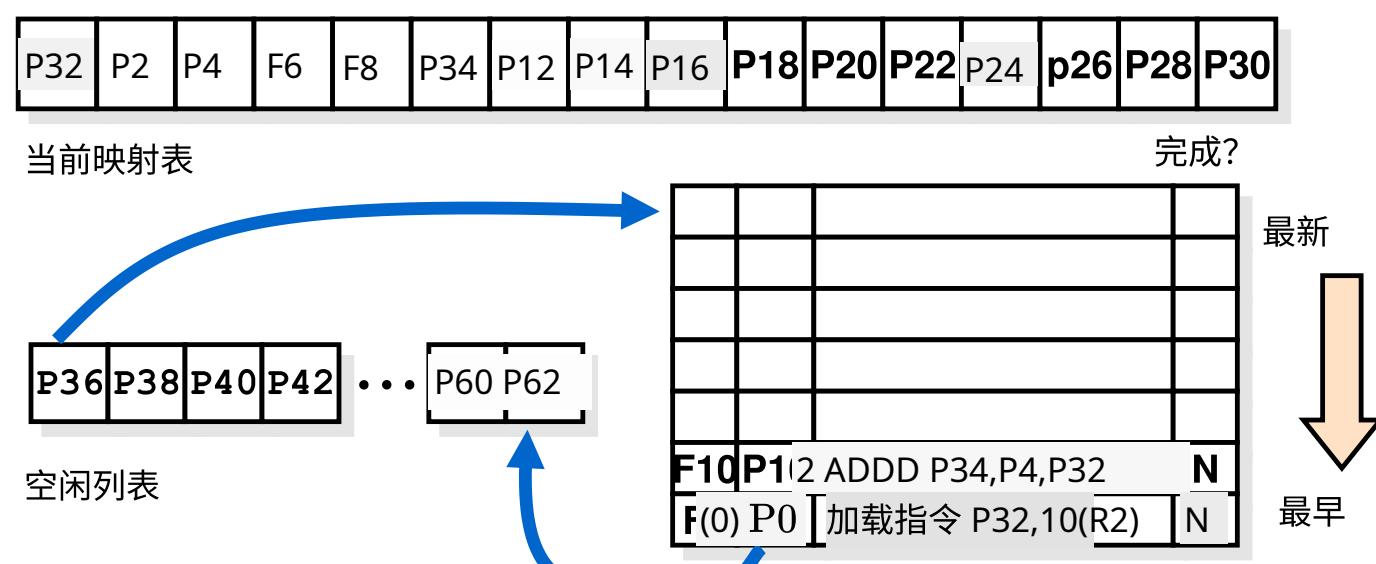
□ 注意物理寄存器P0在此加载点之后处于“死亡”（非“活跃”）状态

> 当提交加载时，我们将释放

-63

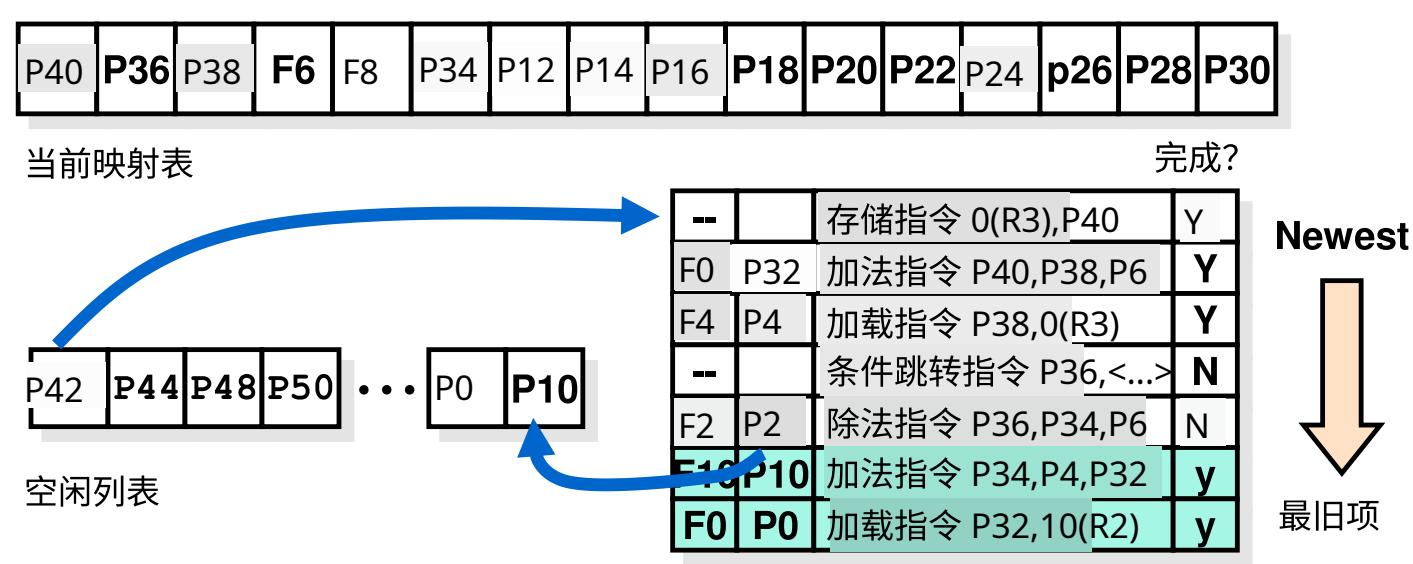
-64

显式寄存器重命名：(R1000架构)



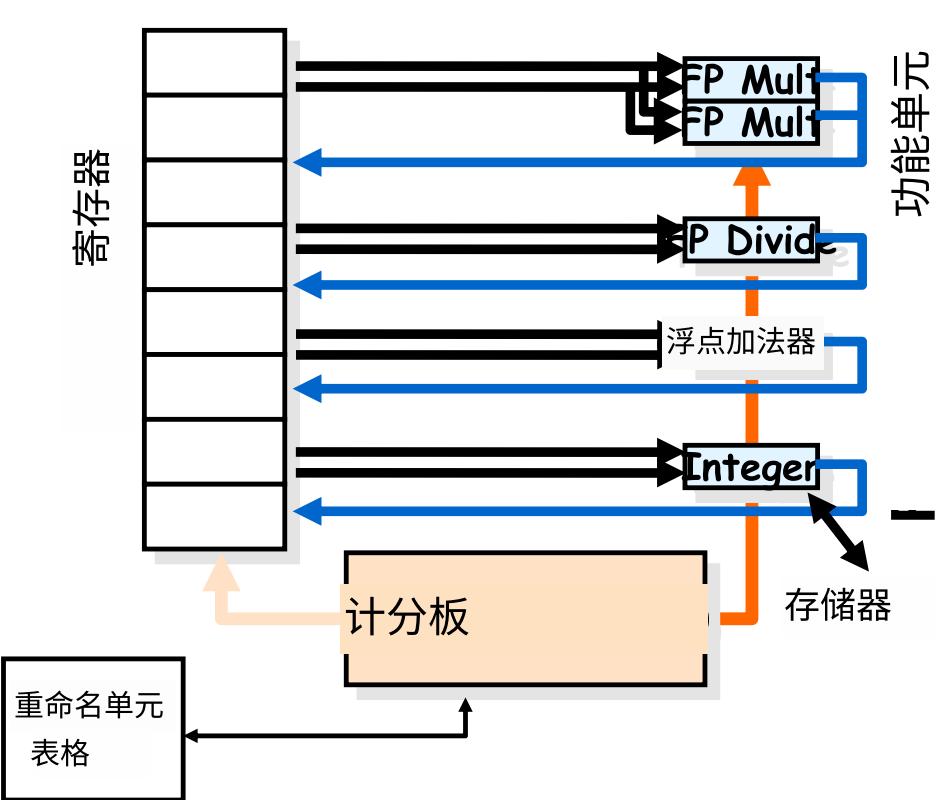
•65

显式寄存器重命名： (R1000 架构风格)



•67

能否在记分板中使用显式寄存器重命名？



•69

带显式重命名的计分板

指令状态:

指令	j	k	读取	执行	写入
LD	F6	34+	R2	操作	组件
LD	F2	45+	R3		
MULTD	F0	F2	F4		
SUBD	F8	F6	F2		
DIVD	F10	F0	F6		
ADDD	F6	F8	F2		

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
国际	No								
国际2	No								
多1	No								
加法	No								
除法	No								

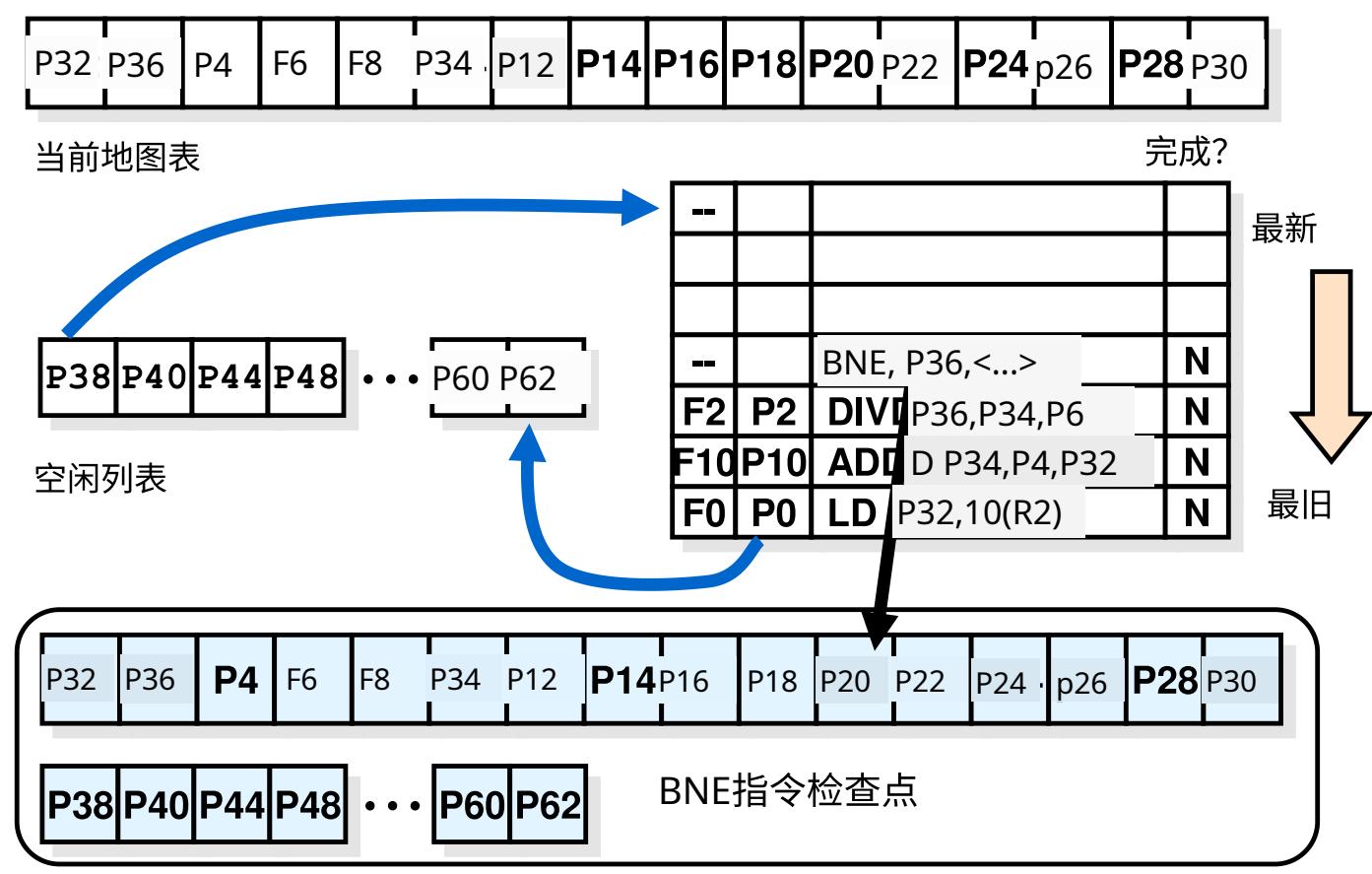
寄存器重命名与结果:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	P0	P2	P4	P6	P8	P10	P12		P30

•初始化重命名表

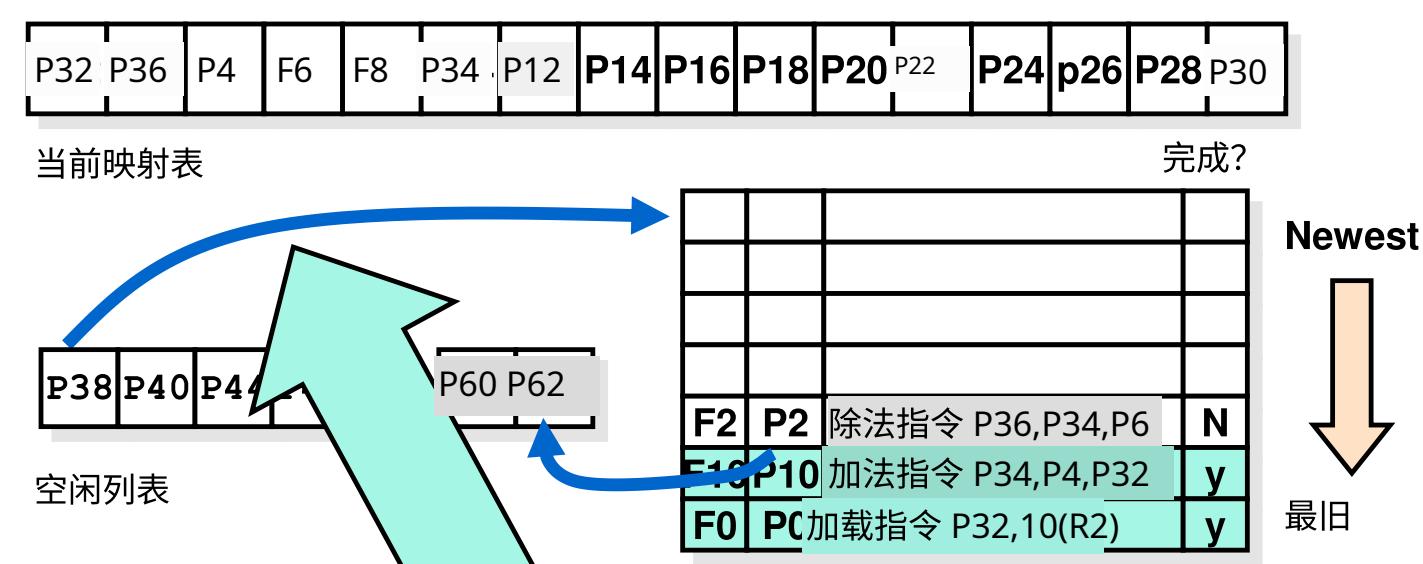
•71

显式寄存器重命名：(R1000架构)

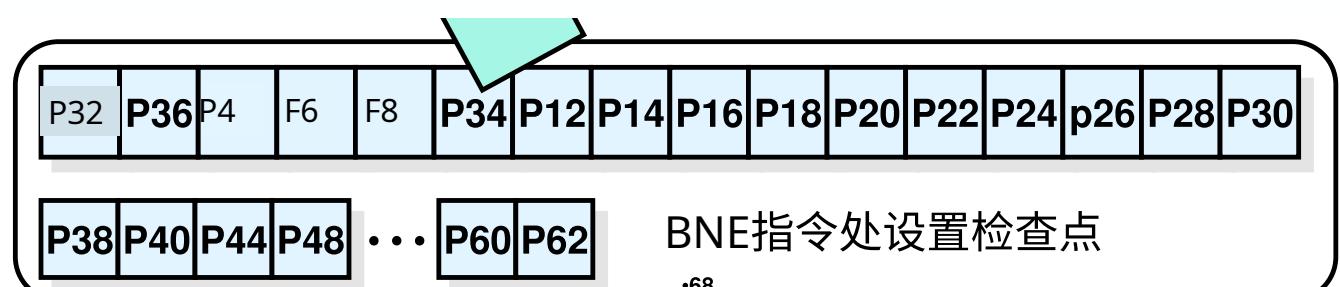


•66

显式寄存器重命名： (R1000风格)



通过恢复映射表和空闲列表修复推测错误



记分牌四阶段

显式重命名控制

□发射阶段——解码指令并检查结构冲突

为结果分配新物理寄存器

> 按程序顺序发出的指令 (用于冒险检测)

> 若无空闲物理寄存器则禁止发射

> 若存在结构冒险则禁止发射

□读取操作数——等待至无冒险时读取操作数

> 此阶段已解决所有真实依赖 (RAW冒险)，因我们需等待指令回写数据

□执行阶段——对操作数进行运算

> 功能单元在接收操作数后开始执行。当结果就绪时，将通知记分板

□写回结果——完成执行

> 注意：不检查WAR或WAW冒险！

•70

•每条指令分配空闲寄存器

•类似于单赋值编译器转换

指令状态:

指令	j	k	读取	执行	执行	写入
LD	F6	34+	R2	操作	开始	组件
LD	F2	45+	R3		1	
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
国际	No								
整数2	No								
乘法1	No								
加法	No								
除法	No								

寄存器重命名与结果:

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	P0	P2	P4	P6	P8	P10	P12		P30

•72

记分板2已重命名

指令状态:

指令	j	k	发射	操作	开始	组件	结果
LD	F6	34+	R2	1	2		
LD	F2	45+	R3	2			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
中断1	是	加载	P32		R2			No	
中断2	是	加载	P34		R3			是	
乘法1	No								
加法	No								
除法	No								

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	P0	P34	P4	P32	P8	P10	P12	P30

-73

•下一步Int1将写入结果，需要该值的位置?

指令	状态:			读取	执行	执行	写入
指令	j	k	发射	操作	启动	组件	结果
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
接口1	是	加载	P32		R2			No	
接口2	是	加载	P34		R3			No	
多1	是	多d	P36	P34	P4	整2		No	是
加	是	子项	P38	P32	P34	国际	整数2	No	No
除法	No								

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	P36	P34	P4	P32	P38	P10	P12	P30

-75

•ADDD为何未发射? 结构冲突! 加法器正被SUBD指令占用

指令状态:	j	k	读取	执行	执行	写入	
指令	j	k	问题	操作	开始	组件	结果
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
中断1	No								
整数2	No								
乘数1	是	多维乘数	P36	P34v	P4			是	是
添加	是	减去	P38	P32v	P34v			是	是
除以	是	除法	P40	P36	P32v	乘法1		No	Yes

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	P36	P34	P4	P32	P38	P40	P12	P30

-77

•NEXT STEP ADDER WITH COMPLETE EXECUTION

重命名记分板8

指令状态:

指令	j	k	下发	操作	开始	完成结果	t
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	7	8	
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

功能单元状态:

时间名称	忙碌	Op	目标	S1	S2	FU	FU	Fj?	Fk?
中断1	No								
中断2	No								
9乘1	是	乘数	P36	P34v	P4			No	No
1加	是	减	P38	P32v	P34v			No	No
除法	是	除	P40	P36	P32v	乘1		No	是

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	P36	P34	P4	P32	P38	P40	P12	P30

-79

重命名的记分牌3指令状态:

指令状态:

指令	j	k	发射	操作	开始	组件	结果
LD	F6	34+	R2				

指令	状态:	j	k	读取		执行		写入	
				问题	操作	开始	组件	结果	
LD	F6	34+	R2	1	2	3	4	5	
LD	F2	45+	R3	2	3	4	5	6	
MULTD	F0	F2	F4	3	7	8			
SUBD	F8	F6	F2	4	7	8	9	10	
DIVD	F10	F0	F6	5					
ADDD	F6	F8	F2	11	12	13	14	15	

功能单元状态:	时间名称	目的地		S1	S2	FU	FU	Fj?	Fk?
		忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj
国际	No								
国际2	No								
7号多1	是	多线程	P36	P34v	P4			No	No
添加	No								
除法	是	除线程	P40	P36	P32v	多路复用器1		No	是

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	P36	P34	P4	P32	P38	P40	P12	P30

•81

重命名记分牌12

指令	状态:	j	k	读取		执行		写入	
				问题	操作	开始	组件	结果	
LD	F6	34+	R2	1	2	3	4	5	
LD	F2	45+	R3	2	3	4	5	6	
MULTD	F0	F2	F4	3	7	8			
SUBD	F8	F6	F2	4	7	8	9	10	
DIVD	F10	F0	F6	5					
ADDD	F6	F8	F2	11	12	13	14	15	

功能单元状态:	时间名称	目的地		S1	S2	FU	FU	Fj?	Fk?
		忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj
接口1	No								
接口2	No								
5号复用器1	是	多选	P36	P34	P4		No	No	
2)添加	是	添加	P42	P38	P34		是	是	
除法	是	除法	P40	P36	P32	乘法1	No	是	

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	P36	P34	P4	P42	P38	P40	P12	P30

•82

重命名记分牌13

指令	状态:	j	k	读取操作		执行开始		执行组件		写入结果	
				问题	操作	开始	组件	结果			
LD	F6	34+	R2	1	2	3	4	5			
LD	F2	45+	R3	2	3	4	5	6			
MULTD	F0	F2	F4	3	7	8					
SUBD	F8	F6	F2	4	7	8	9	10			
DIVD	F10	F0	F6	5							
ADDD	F6	F8	F2	11	12	13	14	15			

功能单元状态:	时间名称	目标		S1	S2	FU	FU	Fj?	Fk?
		忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj
中断1	No								
整型2	No								
4乘1	是	多维	P36	P34	P4		No	No	
1加	是	添加	P42	P38	P34		是	是	
分割	是	分割	P40	P36	P32	多路1	No	是	

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	P36	P34	P4	P42	P38	P40	P12	P30

•83

重命名记分牌14

指令	状态:	j	k	读取		执行开始		执行		写入结果	
				发射	操作	开始	组件	结果			
LD	F6	34+	R2	1	2	3	4	5			
LD	F2	45+	R3	2	3	4	5	6			
MULTD	F0	F2	F4	3							

指令	状态:		读取	执行	执行	写入		
	j	k	问题	操作	开始	组件	结果	
LD	F6	34+	R2	1	2	3	4	5
LD	F2	45+	R3	2	3	4	5	6
MULTD	F0	F2	F4	3	7	8	17	18
SUBD	F8	F6	F2	4	7	8	9	10
DIVD	F10	F0	F6	5				
ADD	F6	F8	F2	11	12	13	14	15

功能单元状态:		目标	S1	S2	FU	FU	Fj?	Fk?
时间名称	忙碌	Op	Fi	Fj	Fk	Qj	Rj	Rk
国际	No							
国际2	No							
多路1	No							
添加	No							
除法	是	除	P40	P36v	P32	乘法器1	是	是

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
18	FU	P36	P34	P4	P42	P38	P40	P12	P30

•89

指令	状态:		读取	执行	执行	写入		
	j	k	问题	操作	开始	组件	结果	
LD	F6	34+	R2	1	2	3	4	5
LD	F2	45+	R3	2	3	4	5	6
MULTD	F0	F2	F4	3	7	8	17	18
SUBD	F8	F6	F2	4	7	8	9	10
DIVD	F10	F0	F6	5	19			
ADD	F6	F8	F2	11	12	13	14	15

功能单元状态:		目标	S1	S2	FU	FU	Fj?	Fk?
时间名称	忙碌	Op	Fi	Fj	Fk	Qj	Rj	Rk
国际	No							
整型2	No							
乘法1	No							
加法	No							
除法	是	—	P40	P36	P32	乘法器1	NO	NO

寄存器重命名与结果

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
19	FU	P36	P34	P4	P42	P38	P40	P12	P30

•90

摘要 #2

□ 显式重命名：物理寄存器数量超出指令集架构需求

> 将重命名与调度分离

-为解决RAW冒险开辟多种可能性

> 重命名表：追踪架构寄存器与物理寄存器的当前映射关系

> 可能涉及复杂的重命名表管理

•91