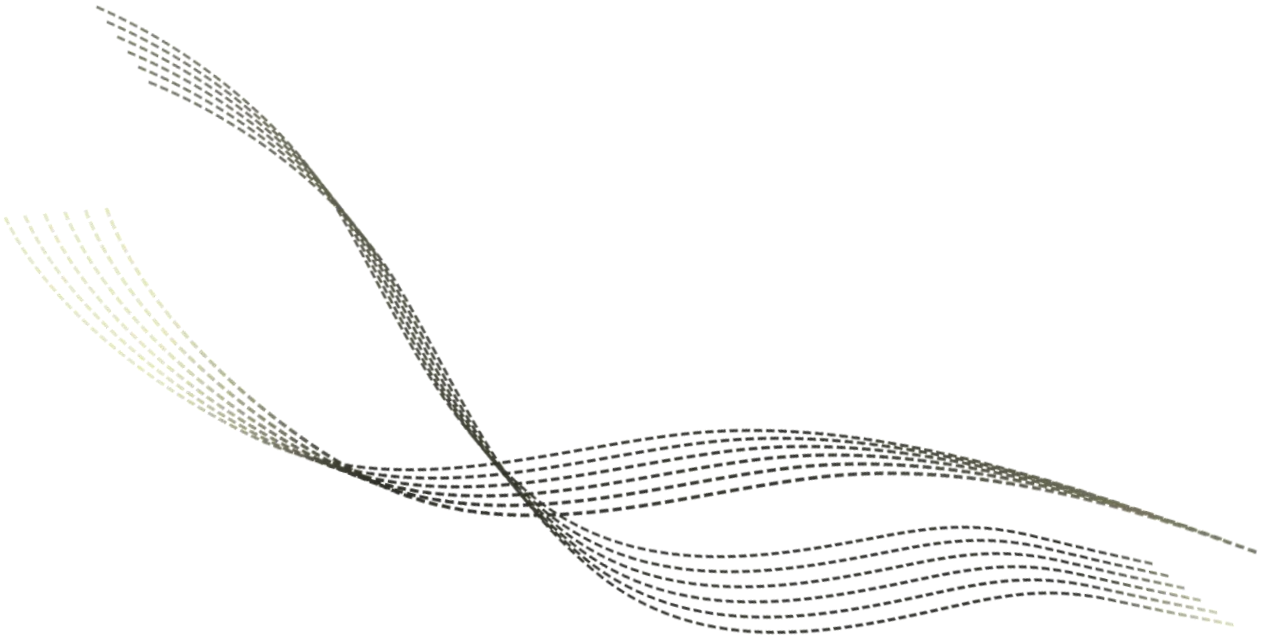
高级计算机体系结构

Advanced Computer Architecture



指令级并行 I

沈明华

目录

CONTENTS

PART 01 流水线冒险

■ 依赖(dependence)

- 三种依赖现象会导致流水线冒险(Hazards)问题
  - 数据依赖(data dependence)
  - 结构依赖(name dependence)
  - 控制依赖(control dependence)
- 流水线冒险会导致计算结果的错误
  - 通常来说，冒险是硬件架构的固有特性
    - 冒险源自于指令乱序执行、指令流水线等加速指令执行的技术
    - 当前指令执行时，逻辑上的上一条指令可能尚未执行完毕，导致数据读取、写入错误

■ 流水线冒险

□ 数据依赖

– 如下图所示指令1和指令2存在数据依赖

指令\时间	T0	T1	T2	T3	T4	T5	…
$r1 = (r0) + 10$	IF1	ID1	EX1	MA1	WB1		
$r2 = (r1) + 20$		IF2	ID2	EX2	MA2	WB2	
$r3 = r4 + r5$			IF3	ID3	EX3	MA3	WB3

– 因此指令2必须在指令1的WB阶段完成后才能取得正确的数据

➢ 指令3无依赖关系，但被指令2阻塞，也无法继续执行

指令\时间	T0	T1	T2	T3	T4	T5	…
$r1 = (r0) + 10$	IF1	ID1	EX1	MA1	WB1		
$r2 = (r1) + 20$		IF2	ID2	X	X	EX2	MA2
$r3 = r4 + r5$			IF3	X	X	ID3	EX3

■ 流水线冒险

□ 结构依赖

- 两条指令使用同一个寄存器或内存地址，实际上没有数据依赖关系
- 比如红框指令1读取r0，指令3写入r0，没有数据依赖，但在有可能同一时刻读取和写入同一个寄存器导致结构依赖
- 比如绿框指令4写入r3，指令5也写入r3，需要保证指令3和5的执行顺序，也是结构依赖

指令\时间	T0	T1	T2	T3	T4	T5	…
$r1 = (r0) * 10$	IF1	ID1	EX1	MA1	WB1		
$r2 = (r2) + 20$		IF2	ID2	EX2	MA2	WB2	
$r0 = r4 + r5$			IF3	ID3	EX3	MA3	WB3
$r3 = r4 * r5$				IF4	ID4	EX4	MA4
$r3 = r6 + r7$					IF5	ID5	EX5

■ 流水线冒险

□ 控制依赖

- 指令3是否执行取决于指令2是否跳转
- 在指令2的EX阶段执行完毕前，并不知道跳转是否执行
- 因此指令3和4的IF阶段必须在指令2的EX阶段执行完毕后

指令\时间	T0	T1	T2	T3	T4	T5	…
$r1 = (r0) + 10$	IF1	ID1	EX1	MA1	WB1		
$jeq\ (r1)\ (r2)\ 4$		IF2	ID2	EX2	MA2	WB2	
$r3 = r4 + r5$?					IF3	ID3	EX3
$r4 = r4 - r5$?					IF4	ID4	EX4

有可能跳转到第4条指令执行，因此这里起始位置一样

■ 流水线冒险

□ 流水线冒险又分为三类

- RAW(read after write)冒险
  - ➢ 数据依赖，导致指令2获得旧值
  - ➢ 又称数据冒险(data hazards)

$$1、r3 \leftarrow (r1)\ op\ (r2)$$
$$2、r5 \leftarrow (r3)\ op\ (r4)$$

- WAR(write after read)冒险
  - ➢ 结构依赖，导致指令1获得r1新值

$$1、r3 \leftarrow (r1)\ op\ (r2)$$
$$2、r1 \leftarrow (r5)\ op\ (r4)$$

- WAW(write after write)冒险
  - ➢ 结构依赖，导致r3写入旧值

$$1、r3 \leftarrow (r1)\ op\ (r2)$$
$$2、r3 \leftarrow (r5)\ op\ (r4)$$

- 思考：WAR和WAW冒险在普通单流水线架构上有可能发生吗?
- 注意：WAR通常译作“写后读”，但按英文理解的翻译应该是读后写

■ 克服流水线冒险

□ 通过**数据转发**缓解数据冒险问题

- 如果r3新数据已经存在于某个部件上，比如T2时刻r3的值已经计算出来，尚未存入存储器
- 增加一个数据转发通道
  - ➢ 指令2绕过存储器直接获得新r3的值

指令\时间	T0	T1	T2	T3	T4	T5	…
$r3 = (r0) * 10$	IF1	ID1	EX1	MA1	WB1		
$r5 = (r3) + 20$		IF2	ID2	EX2	MA2	WB2	

■ 克服流水线冒险

□ 通过**暂停指令**缓解数据冒险问题

- 既然是r3没有写入存储器，那停止指令直到r3写入存储器
- 通过硬件机制检测数据冒险风险，暂停流水线执行的机制被称为**流水线互锁**(pipeline interlock)

指令\时间	T0	T1	T2	T3	T4	T5	…
$r3 = (r0) * 10$	IF1	ID1	EX1	MA1	WB1		
$r5 = (r3) + 20$		IF2	ID2	ID2	ID2	ID2	

■ 克服流水线冒险

□ 通过重命名缓解结构依赖

- 结构依赖一般不存在数据依赖问题
- 因此给寄存器换名可以较好解决结构依赖问题

$$\boxed{\begin{array}{l}1、r3 \leftarrow (r1)\ op\ (r2)\\ 2、r1 \leftarrow (r5)\ op\ (r4)\end{array}} \Rightarrow \boxed{\begin{array}{l}1、r3 \leftarrow (r1)\ op\ (r2)\\ 2、r6 \leftarrow (r5)\ op\ (r4)\end{array}}$$

$$\boxed{\begin{array}{l}1、r3 \leftarrow (r1)\ op\ (r2)\\ 2、r3 \leftarrow (r5)\ op\ (r4)\end{array}} \Rightarrow \boxed{\begin{array}{l}1、r3 \leftarrow (r1)\ op\ (r2)\\ 2、r6 \leftarrow (r5)\ op\ (r4)\end{array}}$$

■ 克服流水线冒险

- 通过分支预测缓解控制依赖
  - 既然问题是不确定跳转指令是否执行，那么可以先斩后奏，预测跳转指令会执行(或者不执行)
  - 预测对了就是性能提升
  - 预测错了，销毁因为预测错误导致错误执行的指令

PART 02

Scoreboard算法

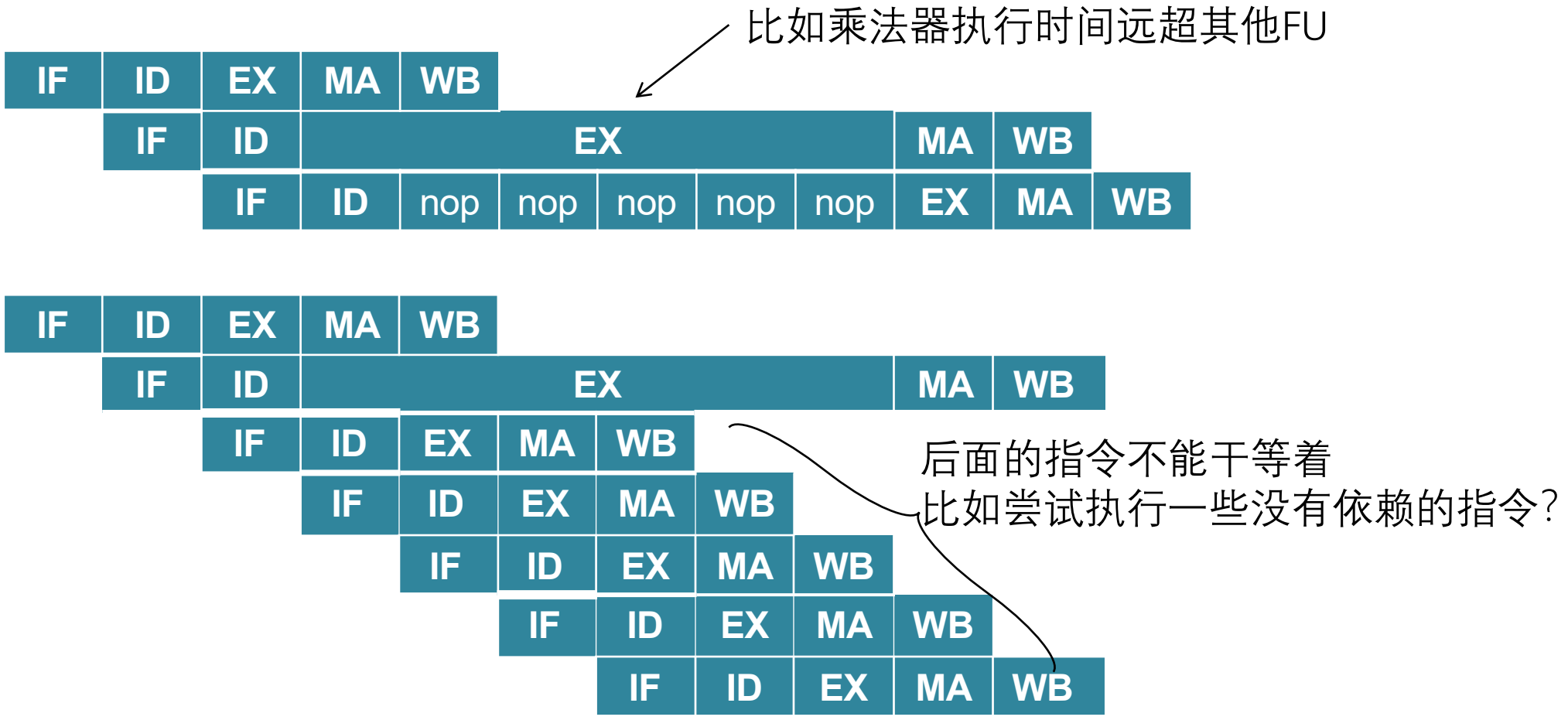
■ 起因

- FU(function unit)
  - 定义为一种根据自身输入进行基础计算，并产生结果的部件
  - 例如加法器、比较器、ALU等等
- FU按照执行时间大致分为两类
  - 执行时间只需少量时钟周期的，比如比较器
  - 执行时间需要大量时钟周期的，比如乘法器、除法器

■ 起因

□ 不同的FU执行时间不同

- 导致不同指令的EX阶段长短不同



■ 起因

□ 想法

– 利用硬件动态调度解决流水线冒险问题

□ 两种经典实现

– 控制为中心的ScoreBoard算法

– 数据为中心的Tomasulo算法

– 注意，无论乱序执行还是顺序执行，顺序指的是指令的完成顺序

顺序执行
(静态调度)

1. lw $3, 100($4)
2. add $2, $3, $4
3. sub $5, $6, $7

指令2必须等到$3数据准备好才能执行
指令3没有数据依赖，但必须等指令2执行完毕

乱序执行
(动态调度)

1. lw $3, 100($4)
3. sub $5, $6, $7
2. add $2, $3, $4

乱序执行把指令3调度到指令2之前执行，这样指令3不必等待

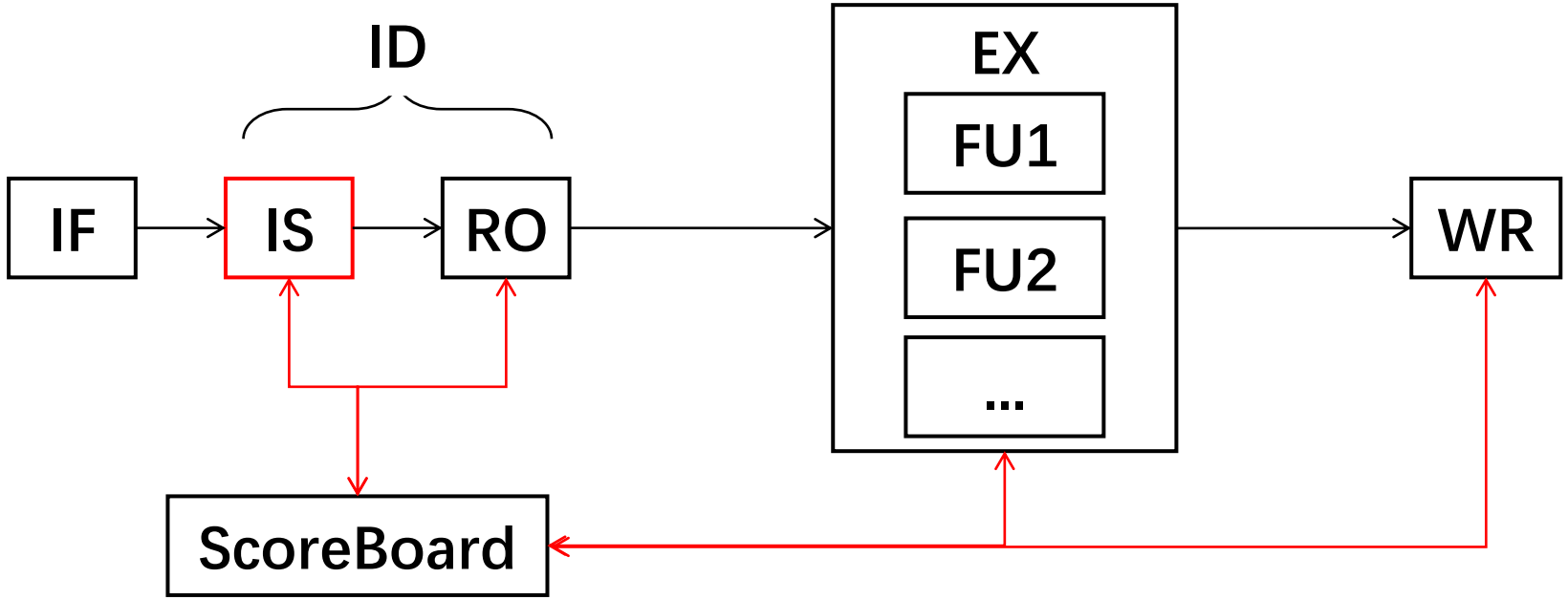
■ Scoreboard算法

- 世界上第一台支持乱序执行的设备是CDC 6600
  - 1964年的大型计算机，采用Scoreboard算法实现乱序执行
  - 包含10个可并行的FU
    - 其中有4个浮点运算器：2个浮点乘法器、1个浮点加法器和除法器

■ Scoreboard算法

□ Scoreboard方法控制的四个阶段

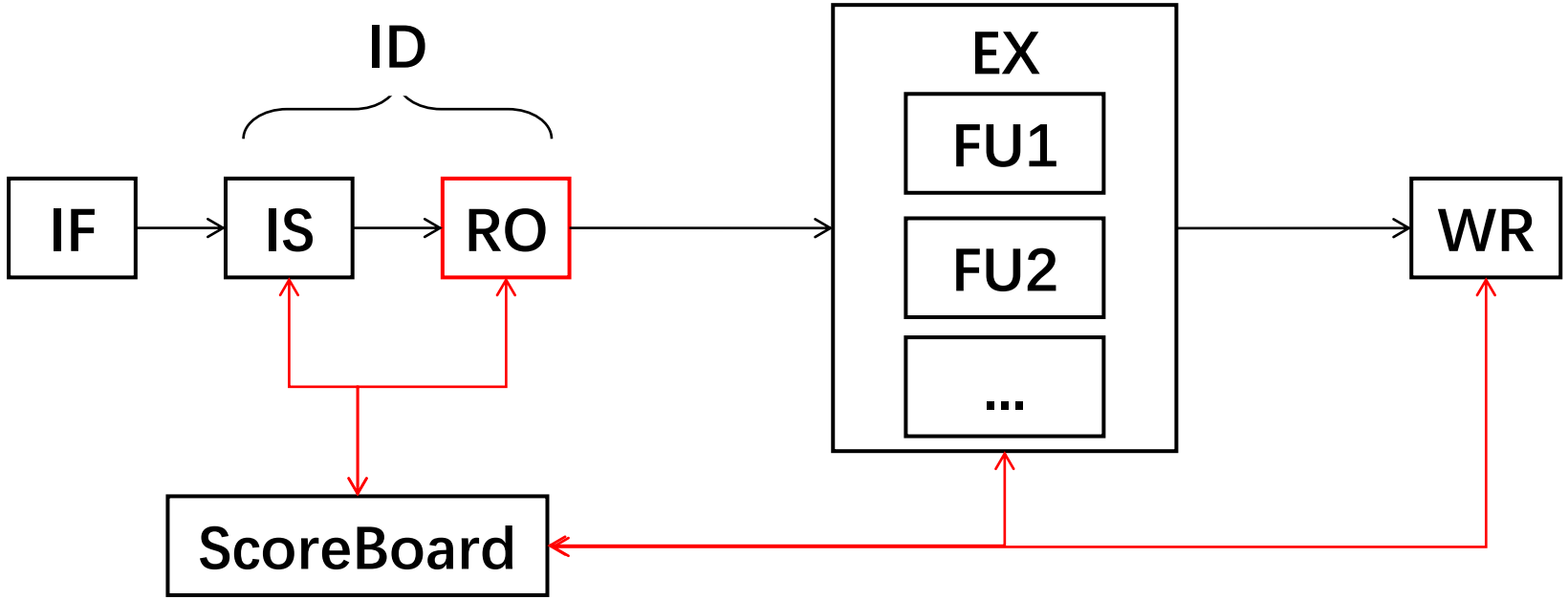
- 发送指令阶段(Issue)：分析指令并检测结构依赖。
  - ➢ 例如一条指令所需的FU空闲且没有其他正在运行的指令需要写入同一个寄存器，即没有WAW冒险。Scoreboard会发送这条指令，否则暂停当前指令的发送



■ Scoreboard算法

□ Scoreboard方法控制的四个阶段

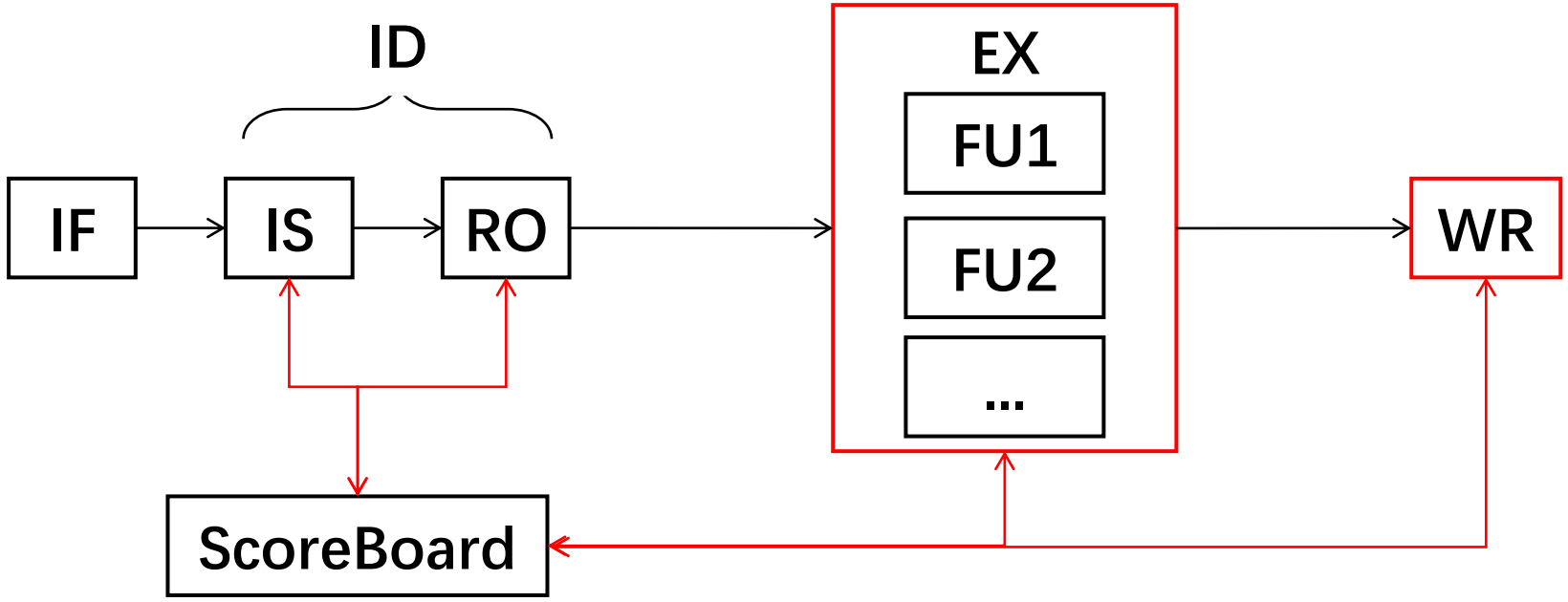
- 读操作数阶段(Read operands)：检测到没有数据依赖后，执行读取操作
  - ➢ 所有操作数就绪，Scoreboard会告诉FU可以执行指令了



■ Scoreboard算法

□ Scoreboard方法控制的四个阶段

- 执行阶段(Execution)：FU完成执行后会通知Scoreboard
- 写回阶段(Write Result)
  - ➢ Scoreboard感知到指令执行完毕，检测是否存在WAR冒险，如果不存在则执行写入，否则暂停当前指令的WR阶段



■ Scoreboard算法

□ 算法概览

- 核心思想：利用scoreboard组件记录操作数间的依赖关系，执行新指令前检查依赖是否满足
  - ➢ 发射指令时同时更新记录表
- 算法所需新结构：记录寄存器依赖关系的状态表和记录ALU操作数依赖关系的状态表

ALU操作数依赖关系

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
Integer									
MUL1									
MUL2									
ADD									
DIVIDE									

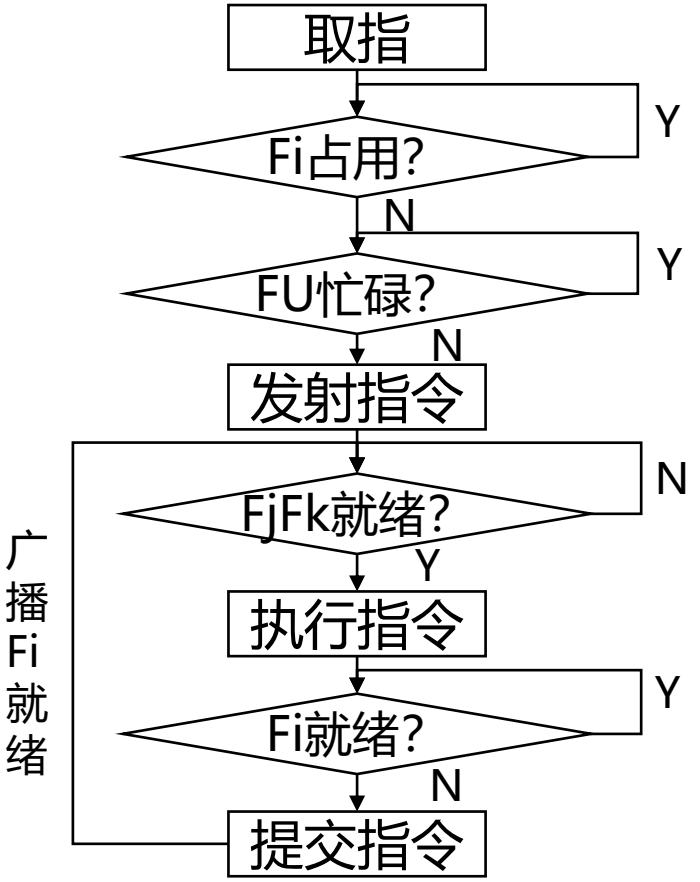
寄存器依赖关系

F1	
F2	
F3	
F4	
F5	
F6	

■ Scoreboard算法

□ 算法概览

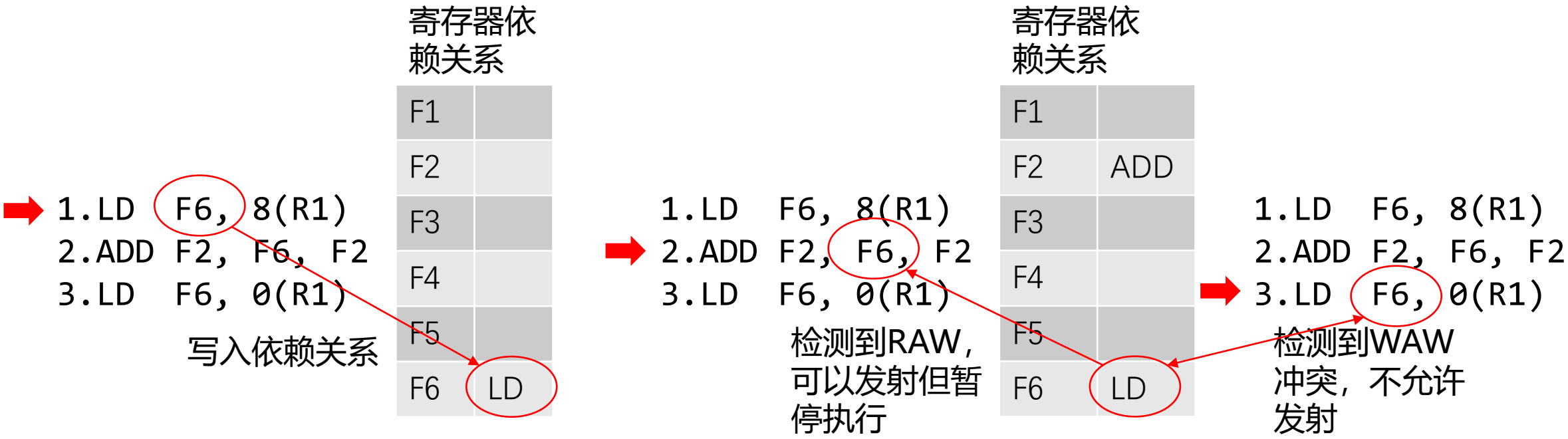
- 定义
  - Fi为目标寄存器
  - Fj和Fk为操作数寄存器
- 说明
  - 第一次Fi占用判断是否有其他指令要写入Fi，避免WAW冒险
  - 第二次FjFk操作数就绪判断是否等待前序指令的结果，避免RAW冒险
  - 第三次Fi就绪判断当前写入是否会覆盖前序指令的操作数读取，避免WAR冒险



■ Scoreboard算法

□ 寄存器状态表

- 指令发射时，一同确定这条指令结果写入的位置(Fi占用？)
  - ➢ 如左图所示：在寄存器状态表上记录当前寄存器的结果来源于哪条指令
  - ➢ ADD指令依赖F6寄存器，而F6寄存器结果未准备好，所以暂停执行



■ Scoreboard算法

□ 操作数状态表

- 寄存器状态表可以暂停指令发射，解决WAW冒险；操作数状态表可以暂停指令执行，解决RAW和WAR冒险
- 所有数据就绪，操作数状态表会通知对应指令可以开始执行(FjFk就绪？)
  - ➢ 这里简化状态表，假设只有一个ADD的FU和一个用于数据加载的integer的FU
  - ➢ 首先LD指令运行，操作数状态表记录LD指令，所有数据就绪，可以直接运行
  - ➢ 然后第二条指令ADD，操作数F6未就绪，因此Busy栏标记为NO，表示未执行
    - 解决RAW冒险

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer	YES	LD	F6	R1				YES	
ADD	NO	ADD	F4	F6	F2	integer		NO	YES

■ Scoreboard算法

□ 操作数状态表

- 第一条指令完成执行，向操作数状态表广播F6操作数就绪
- 状态表中每一项都会检查自己是否依赖F6，如果依赖F6，修改状态为就绪
- 此时ADD指令发现所有操作数都准备就绪，进入执行状态

```
1.LD   F6, 8(R1)
2.ADD  F4, F6, F2
3.LD   F6, 0(R1)
```

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer	YES	LD	F6	R1				YES	
ADD	YES	ADD	F4	F6	F2	integer		YES	YES

■ Scoreboard算法

□ 操作数状态表+寄存器状态表——指令发射

- 发射第一条指令先检查寄存器状态表，是否有操作数依赖前面的指令
  - ➢ 第一条指令发射时，寄存器状态表为空，不存在依赖，可以执行
  - ➢ 更新寄存器状态表，修改目标寄存器状态。F6修改为依赖于integer组件
  - ➢ 更新操作数状态表
- 发射第二条指令先检查寄存器状态表，发现F6依赖于integer FU
  - ➢ 更新寄存器状态表，F4修改为依赖于ADD组件
  - ➢ 更新操作数状态表，记录F6的依赖关系

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	integer

```
1.LD   F6, 8(R1)
2.ADD  F4, F6, F2
3.LD   F6, 0(R1)
```

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer	YES	LD	F6	R1				YES	
ADD	NO	ADD	F4	F6	F2	integer		NO	YES

■ Scoreboard算法

□ 操作数状态表+寄存器状态表——指令发射(续)

- 第三条指令无法发射
  - ➢ 检查寄存器状态表发现目标寄存器F6已经被占用
  - ➢ 为防止WAW冒险，暂停第三条指令的发射

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	integer

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
1.LD F6, 8(R1)	integer	YES	LD	F6	R1				YES	
2.ADD F4, F6, F2	ADD	NO	ADD	F4	F6	F2	integer		NO	YES
3.LD F6, 0(R1)										

■ Scoreboard算法

□ 操作数状态表+寄存器状态表——指令执行

- 第一条指令完成执行，向操作数状态表广播F6操作数就绪
  - ➢ 检查操作数状态表中存在Fj和Fk等于F6，如果存在，那么修改Rj或Rk为YES
- 同时寄存器状态表也会收到F6就绪的信息
  - ➢ 从状态表中抹去F6关于LD的数据依赖记录

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	

```
1.LD   F6, 8(R1)
2.ADD  F4, F6, F2
3.LD   F6, 0(R1)
```

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer									
ADD	YES	ADD	F4	F6	F2	integer		YES	YES

■ Scoreboard算法

□ 操作数状态表+寄存器状态表——指令执行

- 第三条指令发现F6已经空闲，允许发射指令并修改寄存器状态表
- 操作数状态表记录第三条指令，发现所有数据就绪，允许执行

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	integer

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	integer	YES	LD	F6	R1				YES	
	ADD	YES	ADD	F4	F6	F2	integer		YES	YES

■ Scoreboard算法

□ 操作数状态表+寄存器状态表——指令提交(Fi就绪？)

- 假定此时ADD指令的F2操作数一直未就绪，导致ADD指令一直未运行
- 此时第三条指令LD即使执行完毕，也不能写入结果
  - ➢ 通过检查操作数状态表发现F6为就绪态，不能覆盖F6的数据
  - ➢ 操作数状态表暂停LD指令写入结果，解决WAR冒险问题
    - 后面会学习到Tomasulo算法利用寄存器换名解决

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	Integer

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer		LD	F6	R1				YES	
ADD	NO	ADD	F4	F6	F2			YES	NO

■ Scoreboard算法示例

□ Scoreboard数据结构

- 对于一个FU，需要以下数据记录FU的状态
  - *Op*：当前FU执行的操作(比如加减乘除)
  - *Fi*：目标寄存器
  - *Fj*，*Fk*：两个源操作数所在寄存器
  - *Qj*，*Qk*：记录*Fj*和*Fk*由哪条指令产生
  - *Rj*，*Rk*：记录*Fj*和*Fk*是否就绪
  - *Busy*：记录当前FU是否忙碌
- 寄存器状态表
  - 标记当前寄存器被哪个FU占用
- 简化图例，指令的IF阶段不再绘制

■ Scoreboard算法示例

□ Scoreboard数据结构

- 假设指令所需执行周期如下
  - ADD类指令：2周期
  - MULT类指令：10周期
  - DIVD类指令：40周期
- 为方便演示，操作数状态表增加一列剩余执行时间

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR

时钟周期
0

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1			

时钟周期
1

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F6		R2				YES
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
						Integer			

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2		
?			

时钟周期
2

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

思考：指令2能否发送?

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F6		R2				YES
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
						Integer			

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	
?			

时钟周期
3

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

思考：指令3能否发送?

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F6		R2				YES
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
						Integer			

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4

时钟周期
4

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5			

时钟周期
5

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F2		R3				YES
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
		Integer							

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6		
6			

时钟周期
6

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F2		R3				YES
	MUL1	NO	Mult	F0	F2	F4	Integer		NO	YES
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1		Integer							

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	
6			
7			

时钟周期
7

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F2		R3				YES
	MUL1	NO	Mult	F0	F2	F4	Integer		NO	YES
	MUL2									
	ADD	NO	SUB	F8	F6	F2		Integer	YES	NO
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1		Integer						ADD	

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	
6			
7			
8			

时钟周期
8a
前半周期

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer	YES	Load	F2		R3				YES
	MUL1	NO	Mult	F0	F2	F4	Integer		NO	YES
	MUL2									
	ADD	NO	SUB	F8	F6	F2		Integer	YES	NO
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1		Integer						ADD	DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6			
7			
8			

时钟周期
8b
后半周期

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	SUB	F8	F6	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1								ADD	DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9		
8			

时钟周期
9

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

思考：指令6能否发送？为什么？

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
10	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
2	ADD	YES	SUB	F8	F6	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1								ADD	DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9		
8			

时钟周期
10

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
9	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
1	ADD	YES	SUB	F8	F6	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1								ADD	DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	
8			

时钟周期
11

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
8	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
0	ADD	YES	SUB	F8	F6	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1								ADD	DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			

时钟周期
12

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
7	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD									
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1									DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13			

时钟周期
13

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
6	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14		

时钟周期
14

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
5	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
2	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14		

时钟周期
15

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
4	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
1	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14	16	

时钟周期
16

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

思考：指令6能否提交?

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
3	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
0	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14	16	

时钟周期
16

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

思考：指令6能否提交？
不能

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
3	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
0	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14	16	

时钟周期
17

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
2	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14	16	

时钟周期
18

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
1	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	
7	9	11	12
8			
13	14	16	

时钟周期
19

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
0	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8			
13	14	16	

时钟周期
20

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	YES	DIV	F10	F0	F6			YES	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
						ADD			DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8	21		
13	14	16	

时钟周期
21

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	YES	DIV	F10	F0	F6			YES	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
						ADD			DIVIDE

■ Scoreboard算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8	21		
13	14	16	22

时钟周期
22

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD									
40	DIVIDE	YES	DIV	F10	F0	F6			YES	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
									DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8	21	61	
13	14	16	22

时钟周期
61

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD									
0	DIVIDE	YES	DIV	F10	F0	F6			YES	YES

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
									DIVIDE

■ Scoreboard算法示例

		j	*k*
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8	21	61	62
13	14	16	22

时钟周期
62

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
	MUL1									
	MUL2									
	ADD									
	DIVIDE									

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10

■ Scoreboard算法示例

□ 注意到

- 指令执行完毕的先后顺序和发送的先后顺序不一致
- 原因是不同指令执行时间的不同

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9	19	20
7	9	11	12
8	21	61	62
13	14	16	22

■ 小结

□ 不足之处

- 指令调度窗口小，局限在基本块内
  - 只有5个FU，最多同时执行5条指令
- 功能部件较少，特别是用于Load的Integer部件
- 存在结构冒险就停止发射指令
  - 一直等待直到冒险消失
  - 效率比较低

■ Scoreboard算法——17周期WAR冒险处理

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	RO	EX	WR
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14	16	

时钟周期
17

ADD: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

剩余执行时间	FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
	Integer									
2	MUL1	YES	Mult	F0	F2	F4			YES	YES
	MUL2									
	ADD	YES	ADD	F6	F8	F2			YES	YES
	DIVIDE	NO	DIV	F10	F0	F6	MUL1		NO	YES

本可以更早提交，但被暂停执行，可能阻塞后续ADD类指令

寄存器依赖关系

F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
MUL1						ADD			DIVIDE

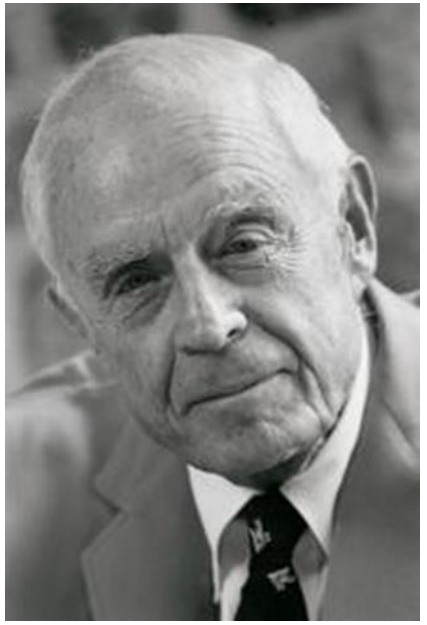
PART 03

Tomasulo算法

■ 背景

□ 背景

- CDC 6600取得巨大成功
- 时任IBM首席执行官Thomas Watson Jr.写给IBM员工备忘录
  - “上周，CDC发布了6600系统。而据我所知制造出CDC 6600的实验室算上门卫一共才34人”
  - “我无法理解为什么我们会失去行业龙头的地位，反而是其他公司提供世界上最先进的计算机”



- 类比一下：你导在一个领域志在必得，然后被一个名不见经传的小导师截胡。现在在组会上说了这么一通话
- 作为实验室的学生，你应该？

■ 背景

◘ IBM 360/91

- 在1964年宣布，作为CDC 6600的对标产品
  - 实际上Tomasulo算法在1967年正式发布
- 动态调度FU(利用Tomasulo算法)
  - 只有2个FU，1个加法器和1个乘法/除法器
- 采用流水线FU，而不是提供更多的FU
  - 加法器支持同时执行3条指令，乘法/除法器支持2条指令
  - 在本堂课中为简化，把多级流水线FU等效为存在多个FU

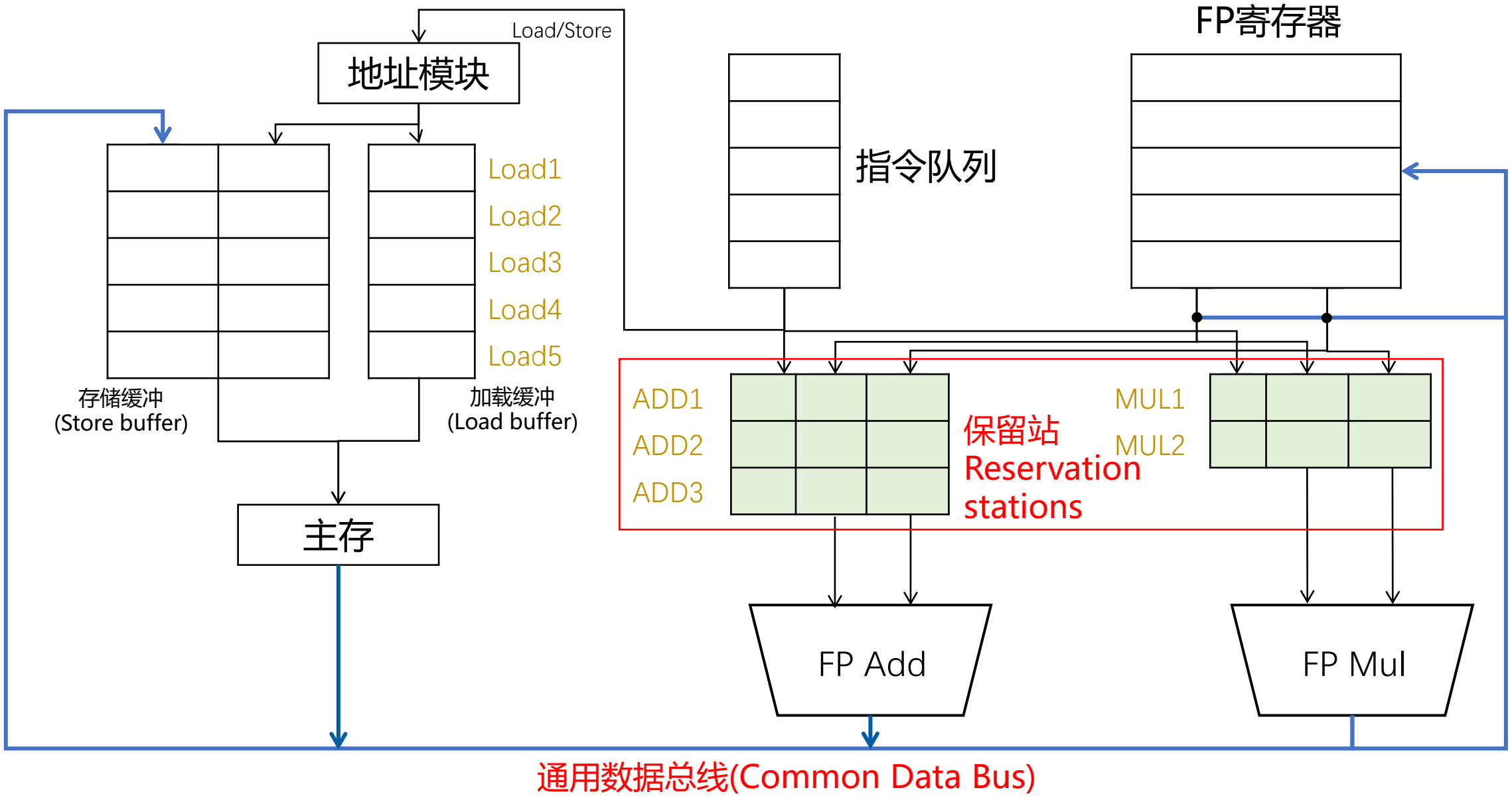
■ Tomasulo算法

□ 与Scoreboard算法对比

- 关键思路——换名
- 指令中的**寄存器**被**实际数值或者指向保留站的指针**代替；这一过程称为寄存器换名(register renaming)
  - 消除WAR、WAW风险
  - 因为保留站比实际寄存器数量多得多，所以可以优化编译器不能优化的一些指令序列

Tomasulo	scoreboard
(6 load, 3 store, 3+, 2×/÷)	(1 load/store, 1+, 2×,1÷)
窗口大小：≤ 14指令	窗口大小：≤ 5指令
WAW通过换名避免	WAW通过暂停指令发射避免
WAR通过换名避免	WAR通过暂停指令执行避免
通过保留站控制	由scoreboard集中控制

■ Tomasulo算法


Load/Store
地址模块
Load1
Load2
Load3
Load4
Load5
存储缓冲
(Store buffer)
加载缓冲
(Load buffer)
主存
指令队列
FP寄存器
ADD1
ADD2
ADD3
保留站
Reservation
stations
MUL1
MUL2
FP Add
FP Mul
通用数据总线(Common Data Bus)

■ Tomasulo算法

□ 保留站(Reservation stations)

- 保留站作用
  - ➢ 一旦操作数就绪，立即获取并缓存操作数
  - ➢ 当所有操作数就绪，立即通知相关FU执行指令
- Load/Store 缓冲
  - ➢ 功能类似于保留站
  - ➢ 包含一个tag字段，保证数据来源正确
  - ➢ 取代Scoreboard中的integer FU

■ Tomasulo算法

□ 寄存器换名(Register renaming)

- 作用：把指令中的寄存器，替换为tag或指向保留站的指针
- 从而缓解寄存器名字一致导致的结构冒险
  - 但没有完全消除结构冒险
  - 多个保留站可能会竞争同一个FU
- 为充分利用寄存器换名优化
  - 一般提供保留站的数量远远大于寄存器数量
  - 因而硬件能完成一些编译器不能完成的优化工作
  - 换名过程不一定是显式的，这在后面的样例中有所体现
- 由此启发后续CPU的设计
  - 例如Alpha 21264、HP8000、MIPS 10000、Pentium II、PowerPC 604……

■ Tomasulo算法

□ 寄存器状态表和操作数状态表相对Scoreboard变化

- 寄存器状态表基本没有变化，履行的职能基本不变
- 操作数状态表去掉就绪的标志位Rj、Rk，目标地址Fi
  - 操作数Vj和Vk存放实际操作数，而不是Scoreboad的寄存器指针
  - 基本职能不变，增加数据缓存能力

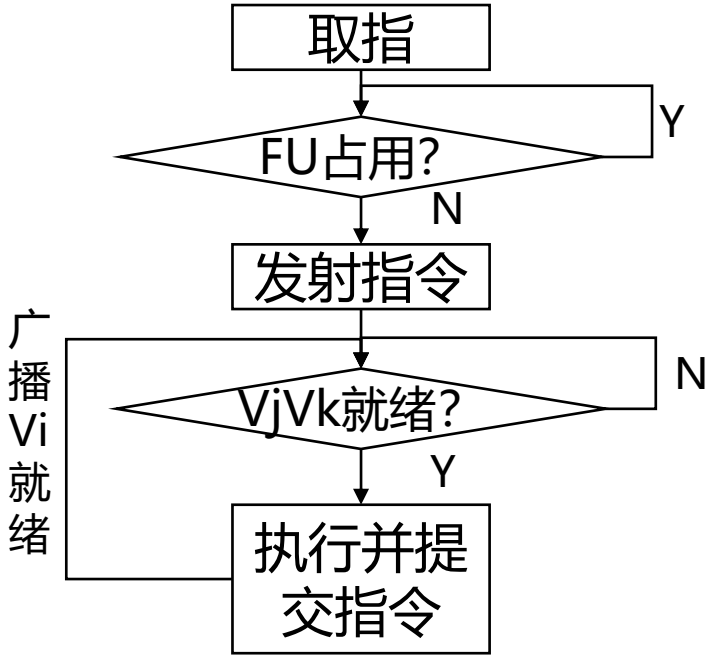
FU名字	Busy	Op	目标 Fi	操作数 Fj	操作数 Fk	来源 Qj	来源 Qk	就绪? Rj	就绪? Rk
integer									
ADD									

FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
ADD1						
MUL2						

■ Tomasulo算法

□ 算法概览

- 定义
  - Vi是指令执行的结果
  - Vj和Vk是操作数
- 相比Scoreboard算法流程简单不少
  - 寄存器换名解决WAW和WAR冒险



■ Tomasulo算法

□ 操作数状态表

- 发射前两条指令，第一条LD指令无依赖关系，因此可以直接执行
- 第二条ADD指令，检查Qj发现操作数F6需要从integer的FU中获取
  - Qj数据来源和Scoreboard算法一致，通过检查寄存器状态表得到
  - 数据未就绪，暂停ADD指令的执行
  - 只要Qj、Qk非空，就是数据未就绪，不可执行

```
1.LD   F6, 8(R1)
2.ADD  F4, F6, F2
3.LD   F6, 0(R1)
```

FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
integer	YES	LD	R1			
ADD	NO	ADD	F6	F2	integer	

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	LD

■ Tomasulo算法

□ 操作数状态表

- 假定第一条指令完成执行，广播F6数据就绪
  - ➢ 寄存器状态表更新，删去F6的依赖
- ADD项的Vj收到广播，保存F6数据，清除Qj，开始执行ADD指令
  - ➢ 从这里开始ADD项的F6不再指向寄存器，而是保留站
  - ➢ 保留站从刚才的广播中获得F6的最新数据

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
integer						
ADD	YES	ADD	F6	F2		

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	

■ Tomasulo算法

- 操作数状态表
  - 第三条指令LD可以发射，且没有数据依赖问题，可以直接执行
- 如果ADD指令因为F2未就绪无法执行，但F6是就绪的
  - 此时第三条LD指令仍然可以提交(Scoreboard必须阻塞LD的提交)
    - 因为ADD项的F6实际指向保留站，而LD的F6实际指向寄存器
    - 二者虽同名，但并不存在数据依赖关系
    - 这是隐式寄存器换名的一个体现

```
1.LD  F6, 8(R1)
2.ADD F4, F6, F2
3.LD  F6, 0(R1)
```

FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
integer	YES	LD	R1			
ADD	NO	ADD	F6	F2		xxx

寄存器依赖关系

F1	
F2	
F3	
F4	ADD
F5	
F6	LD

■ Tomasulo算法

□ Tomasulo算法数据结构

- 数据结构
  - *Op*：操作符
  - *Vj*、*Vk*：源操作数的值
  - *Qj*、*Qk*：源操作数来源
  - *Busy*：标记模块是否在忙
- Tomasulo的指令有三种状态，而Scoreboard有四种
  - 发送(Issue)：从指令缓冲中获得一条指令(一般是有序发送)
  - 执行(Execution)：执行操作(有可能是无序的)
  - 写结果(Write Result)：完成指令执行(有可能是无序的)
- 一旦操作数就绪立即执行指令
  - 通过数据总线唤醒后续指令

■ Tomasulo算法示例

时钟周期
0

				Issue	EX	WR
1. LD	F6	36	R2			
2. LD	F2	49	R3			
3. MUL.D	F0	F2	F4			
4. SUB.D	F8	F6	F2			
5. DIV. D	F10	F0	F6			
6. ADD.D	F6	F8	F2			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

Load:	2	cycles
Add:	2	cycles
Mult:	10	cycles
Divd:	40	cycles

寄存器结果

F0	
F1	
F2	
F3	
F4	
F5	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
1

Load:	2	cycles
Add:	2	cycles
Mult:	10	cycles
Divd:	40	cycles

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	EX	WR
1		

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

寄存器状态

F0	
F2	
F4	
F6	Load1
F8	
F10	

	Busy	Addr
Load1	Yes	36+R2
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
2

				Issue	EX	WR
		j	*k*			
1. LD	F6	36	R2	1	2	
2. LD	F2	49	R3	2		
3. MUL.D	F0	F2	F4			
4. SUB.D	F8	F6	F2			
5. DIV. D	F10	F0	F6			
6. ADD.D	F6	F8	F2			

Load:	2	cycles
Add:	2	cycles
Mult:	10	cycles
Divd:	40	cycles

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

寄存器状态

F0	
F2	Load2
F4	
F6	Load1
F8	
F10	

	Busy	Addr
Load1	Yes	36+R2
Load2	Yes	49+R3
Load3		

■ Tomasulo算法示例

时钟周期
3

				Issue	EX	WR
		j	k			
1. LD	F6	36	R2	1	2	
2. LD	F2	49	R3	2	3	
3. MUL.D	F0	F2	F4	3		
4. SUB.D	F8	F6	F2			
5. DIV. D	F10	F0	F6			
6. ADD.D	F6	F8	F2			

	Load:	2	cycles
	Add:	2	cycles
	Mult:	10	cycles
	Divd:	40	cycles

寄存器状态

F0	MUL1
F2	Load2
F4	
F6	Load1
F8	
F10	

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	NO	MUL		R[F4]	Load2	
	MUL2						

	Busy	Addr
Load1	Yes	36+R2
Load2	Yes	49+R3
Load3		

■ Tomasulo算法示例

时钟周期
4

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	EX	WR
1	2	4
2	3	
3		
4		

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	Load2
F4	
F6	M[A1]
F8	ADD1
F10	

M[A1]是虚拟寄存器

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1	NO	SUB	M[A1]			Load2
	ADD2						
	ADD3						
	MUL1	NO	MUL		R[F4]	Load2	
	MUL2						

	Busy	Addr
Load1		
Load2	Yes	49+R3
Load3		

■ Tomasulo算法示例

时钟周期
5

				Issue	EX	WR
		j	k			
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3		
4. SUB.D	F8	F6	F2	4		
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2			

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
2	ADD1	Yes	SUB	M[A1]	M[A2]		
	ADD2						
	ADD3						
10	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A1]
F8	ADD1
F10	MUL2

	Busy	Addr
Load1		
Load2		
Load3		

Tomasulo算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	EX	WR
1	2	4
2	3	5
3	6	
4	6	
5		
6		

时钟周期
6

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
1	ADD1	Yes	SUB	M[A1]	M[A2]		
	ADD2	NO	ADD		M[A2]	ADD1	
	ADD3						
9	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	ADD2
F8	ADD1
F10	MUL2

F6数据已经被存放，因此这里可以直接发射指令6

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
7

				Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6		

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
0	ADD1	Yes	SUB	M[A1]	M[A2]		
	ADD2	NO	ADD		M[A2]	ADD1	
	ADD3						
8	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	ADD2
F8	ADD1
F10	MUL2

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
8

				Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6		

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	ADD2
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
2	ADD2	Yes	ADD	M[A3]	M[A2]		
	ADD3						
7	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
9

				Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	ADD2
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
1	ADD2	Yes	ADD	M[A3]	M[A2]		
	ADD3						
6	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
10

				Issue	EX	WR
		j	k			
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	ADD2
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
0	ADD2	Yes	ADD	M[A3]	M[A2]		
	ADD3						
5	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

		j	k	Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	11

时钟周期
11

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
4	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
12

				Issue	EX	WR
		j	k			
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	11

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
3	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
13

		j	k	Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	11

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
2	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	EX	WR
1	2	4
2	3	5
3	6	
4	6	8
5		
6	9	11

时钟周期
14

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
1	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
15

		j	k	Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	11

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	MUL1
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
0	MUL1	Yes	MUL	M[A2]	R[F4]		
	MUL2	NO	DIV		M[A1]	MUL1	

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
16

				Issue	EX	WR
		j	k			
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	16
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5		
6. ADD.D	F6	F8	F2	6	9	11

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	M[A5]
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
40	MUL2	Yes	DIV	M[A5]	M[A1]		

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
17

		j	k
1. LD	F6	36	R2
2. LD	F2	49	R3
3. MUL.D	F0	F2	F4
4. SUB.D	F8	F6	F2
5. DIV. D	F10	F0	F6
6. ADD.D	F6	F8	F2

Issue	EX	WR
1	2	4
2	3	5
3	6	16
4	6	8
5	17	
6	9	11

Load:	2 cycles
Add:	2 cycles
Mult:	10 cycles
Divd:	40 cycles

寄存器状态

F0	M[A5]
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
39	MUL2	Yes	DIV	M[A5]	M[A1]		

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
56

				Issue	EX	WR
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	16
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5	17	
6. ADD.D	F6	F8	F2	6	9	11

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
0	MUL2	Yes	DIV	M[A5]	M[A1]		

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

寄存器状态

F0	M[A5]
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	MUL2

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法示例

时钟周期
57

				Issue	EX	WR
		j	*k*			
1. LD	F6	36	R2	1	2	4
2. LD	F2	49	R3	2	3	5
3. MUL.D	F0	F2	F4	3	6	16
4. SUB.D	F8	F6	F2	4	6	8
5. DIV. D	F10	F0	F6	5	17	57
6. ADD.D	F6	F8	F2	6	9	11

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divd: 40 cycles

寄存器状态

F0	M[A5]
F2	M[A2]
F4	
F6	M[A4]
F8	M[A3]
F10	M[A6]

	Busy	Addr
Load1		
Load2		
Load3		

■ Tomasulo算法

□ 总结

- 保留站：提前缓存操作数
  - 防止寄存器成为性能瓶颈，实现寄存器换名
  - 解决Scoreboard算法中WAR和WAW冒险
    - 例子中指令6和指令4就是WAR冒险，Scoreboard延后指令6的发送来解决冒险问题
  - 具有硬件动态完成循环展开的功能
- 缺点：
  - 复杂
  - 需要大量高速互联的寄存器堆
  - 公共数据总线将会成为新的性能瓶颈
    - 公共数据总线必须广播到多个保留站，意味着总线容量要大，写入操作多
    - 公共数据总线一次只能广播一条数据，限制多FU并发执行的可能

■ Tomasulo算法 V.S. Scoreboard算法

- Tomasulo关键思想
  - 利用保留站实现独立控制
  - 通用数据总线广播FU的计算结果
  - 使用tag区分指令中的同名寄存器
- 对比Scoreboard不同之处

Tomasulo	scoreboard
(6 load, 3 store, 3+, 2×/÷)	(1 load/store, 1+, 2×,1÷)
窗口大小：≤ 14指令	窗口大小：≤ 5指令
WAW通过换名避免	WAW通过暂停指令发射避免
WAR通过换名避免	WAR通过暂停指令执行避免
通过保留站控制	由scoreboard集中控制

■ Tomasulo算法 V.S. Scoreboard算法

□ 对比总时间

- Tomasulo算法大部分指令执行时间比Scoreboard算法快
- Scoreboard算法慢的主要原因是单一部件易被阻塞
  - ➢ 例如Scoreboard指令2因为缺少缓冲，必须等待指令1执行完毕让出计算部件才能发射
  - ➢ 指令3本可以在第3个周期发射，因为指令2的阻塞，被迫于第6周期发射

		j	k	tomasulo			scoreboard			
				Issue	EX	WR	Issue	RO	EX	WR
1. LD	F6	36	R2	1	2	4	1	2	3	4
2. LD	F2	49	R3	2	3	5	5	6	7	8
3. MUL.D	F0	F2	F4	3	6	16	6	9	19	20
4. SUB.D	F8	F6	F2	4	6	8	7	9	11	12
5. DIV. D	F10	F0	F6	5	17	57	8	21	61	62
6. ADD.D	F6	F8	F2	6	9	11	13	14	16	22

■ Tomasulo算法(番外)

□ 什么是循环展开

- 右图是求等差数列和的标准写法
- 编译成汇编代码如下所示
  - 可以发现真正执行有效操作的有add R0 R1和add R1 1这两条
  - 其余2条指令都为循环体服务
  - 整体指令利用率只有50%

```
int sum;
for (int i = 0; i < 100; i++) {
    sum += i;
}
```

```
loop:
    add R0 R1
    add R1 1
    cmp R1 100
    jle loop
```

■ Tomasulo算法(番外)

□ 什么是循环展开

- 如果循环体内部执行多次加法
- 编译得到汇编代码
  - 此时有效操作利用率高达80%
  - 为循环体服务的指令仍然只有2条
- 没有改变程序逻辑但显著提高指令利用率

```
int sum;
for (int i = 0; i < 100; i+=4) {
    sum += i;
    sum += i + 1;
    sum += i + 2;
    sum += i + 3;
}
```

```
loop:
    add R0 R1
    add R1 1
    add R0 R1
    add R1 1
    add R0 R1
    add R1 1
    add R0 R1
    add R1 1
    cmp R1 100
    jle loop
```

■ Tomasulo算法(番外)

- Tomasulo的结构非常适合循环展开
  - 假设给数组中每一项都乘以二
  - 编译得到的汇编代码如下所示
    - R1存放数组指针
  - 假设
    - 这里乘法只需要4周期
    - 第一次Load需要8周期(因为cache miss)
    - 第二次Load只需要1周期
    - Store统一为3周期
    - 为简化不展示Sub和Bnez的执行
    - EX C为指令执行完毕的时间

```
int num[100];
for (int i = 99; i > 0; i--) {
    num[i] = num[i] * 2
}
```

```
loop:
    Load  F0 0 R1
    MUL   F4 F0 F2
    Store F4 0 R1
    Sub   R1 R1 4
    Bnez  R1 loop
```

■ Tomasulo算法(番外)

时钟周期

0

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR

	Busy	Addr
Load1		
Load2		
Load3		
Store1		
Store2		
Store3		

	剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
ADD1								
ADD2								
ADD3								
MUL1								
MUL2								

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

1

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		

	Busy	Addr	Qi
Load1	YES	100	
Load2			
Load3			
Store1			
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	Load1
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

2

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		

	Busy	Addr	Qi
Load1	YES	100	
Load2			
Load3			
Store1			
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2						

F0	Load1
F2	
F4	MUL1
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

3

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		

	Busy	Addr	Qi
Load1	YES	100	
Load2			
Load3			
Store1	YES	100	MUL1
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2						

F0	Load1
F2	
F4	MUL1
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

4

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		

	Busy	Addr	Qi
Load1	YES	100	
Load2			
Load3			
Store1	YES	100	MUL1
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2						

注意这个周期在处理Sub指令

F0	Load1
F2	
F4	MUL1
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

5

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		

	Busy	Addr	Qi
Load1	YES	100	
Load2			
Load3			
Store1	YES	100	MUL1
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2						

注意这个周期在处理Bnez指令

F0	Load1
F2	
F4	MUL1
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

6

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		
6		

	Busy	Addr	Qi
Load1	YES	100	
Load2	YES	96	
Load3			
Store1	YES	100	MUL1
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2						

F0	Load2
F2	
F4	MUL1
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

7

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		
6		
7		

	Busy	Addr	Qi
Load1	YES	100	
Load2	YES	96	
Load3			
Store1	YES	100	MUL1
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2	YES	MUL		R[F2]	Load2	

F0	Load2
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

8

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1		
2		
3		
6		
7		
8		

	Busy	Addr	Qi
Load1	YES	100	
Load2	YES	96	
Load3			
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2	YES	MUL		R[F2]	Load2	

F0	Load2
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

9

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	
2. MUL	F4	F0	F2	2		
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6		
7. MUL	F4	F0	F2	7		
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1	YES	100	
Load2	YES	96	
Load3			
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1	YES	MUL		R[F2]	Load1	
	MUL2	YES	MUL		R[F2]	Load2	

F0	Load2
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

10

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2		
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	
7. MUL	F4	F0	F2	7		
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2	YES	96	
Load3			
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
4	MUL1	YES	MUL	M[100]	R[F2]		
	MUL2	YES	MUL		R[F2]	Load2	

F0	Load2
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

11

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1	9	10
2		
3		
6	10	11
7		
8		

	Busy	Addr	Qi
Load1			
Load2			
Load3	YES	92	
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
3	MUL1	YES	MUL	M[100]	R[F2]		
4	MUL2	YES	MUL	M[96]	R[F2]		

此时Load3已经发射，因为只演示两个周期的原因，后续不再讨论

	Load3
F0	
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

12

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1	9	10
2		
3		
6	10	11
7		
8		

	Busy	Addr	Qi
Load1			
Load2			
Load3	YES	92	
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
2	MUL1	YES	MUL	M[100]	R[F2]		
3	MUL2	YES	MUL	M[96]	R[F2]		

为什么不能发送第三条MUL?

F0	Load3
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

13

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2		
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7		
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
1	MUL1	YES	MUL	M[100]	R[F2]		
2	MUL2	YES	MUL	M[96]	R[F2]		

F0	
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

14

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7		
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	MUL1
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
0	MUL1	YES	MUL	M[100]	R[F2]		
1	MUL2	YES	MUL	M[96]	R[F2]		

F0	
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

15

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	M[100]
Store2	YES	96	MUL2
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
0	MUL2	YES	MUL	M[96]	R[F2]		

F0	
F2	
F4	MUL2
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

16

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	16
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	M[100]
Store2	YES	96	M[96]
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

17

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3		
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	16
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	M[100]
Store2	YES	96	M[96]
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

18

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3	18	
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	16
8. SD	F4	0	R1	8		

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1	YES	100	M[100]
Store2	YES	96	M[96]
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

19

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3	18	19
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	16
8. SD	F4	0	R1	8	19	

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1			
Store2	YES	96	M[96]
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

时钟周期

20

		j	k	Issue	EX C	WR
1. LD	F0	0	R1	1	9	10
2. MUL	F4	F0	F2	2	14	15
3. SD	F4	0	R1	3	18	19
6. LD	F0	0	R1	6	10	11
7. MUL	F4	F0	F2	7	15	16
8. SD	F4	0	R1	8	19	20

	Busy	Addr	Qi
Load1			
Load2			
Load3			
Store1			
Store2			
Store3			

剩余执行时间	FU名字	Busy	Op	操作数 Vj	操作数 Vk	来源 Qj	来源 Qk
	ADD1						
	ADD2						
	ADD3						
	MUL1						
	MUL2						

F0	
F2	
F4	
F6	
F8	
F10	

■ Tomasulo算法(番外)

□ 观察到

- 第一次MUL和第二次MUL执行结束时间恰好只相差一个周期
- 实际上两次MUL指令发送相差五个时钟周期
  - ➢ 一方面cache miss推迟第一次MUL指令真正执行时间，另一方面保留站消除WAR冒险使得第二次MUL指令不必等待第一次Store指令执行完毕，提前第二次MUL真正执行时间
  - ➢ 是否可以配置足够多的保留站让计算指令连续的执行?

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1	9	10
2	14	15
3	18	19
6	10	11
7	15	16
8	19	20

■ 小结

□ 观察到

- 同一个循环体，两次MUL执行写入的寄存器名相同
  - ➢ 这是编译器优化不到的地方，必须在硬件层面加以区分
- Tomasulo算法的寄存器换名策略
  - ➢ 发现两次执行的MUL指令之间没有数据依赖，完全可以重定向到另一个寄存器避免虚假的数据依赖
  - ➢ 更加适合循环展开优化

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1	9	10
2	14	15
3	18	19
6	10	11
7	15	16
8	19	20

PART 04 重排序缓冲

■ 重排序缓冲

□ 乱序执行(Out of Order processing)

– 乱序执行的处理器早在1960年代就商用，但在1990年代销声匿迹。乱序执行存在两个关键问题无法解决

➢ 精确断点(precise traps)：断在第i条和i+1条指令之间

- 第i条及其之前的指令造成的数据改变需要完全可见
- 第i+1条及其之后的指令造成的数据改变不可见

➢ 乱序执行的特点，增加程序Debug的难度

- 比如在15周期停止，想看第一次MUL的结果，但此时F0已经是第二次Load结果，二者不匹配

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

Issue	EX C	WR
1	9	10
2	14	15
3	18	19
6	10	11
7	15	16
8	19	20

■ 重排序缓冲

□ 乱序执行

– 第二个关键问题

- ➢ 分支预测(branch prediction)：比较结果未计算完毕，只能猜测是否跳转
- ➢ 前面提到的例子一直以正确跳转为前提。如果指令5没有按预期跳转到指令1，那么后续发射的指令678都是错误指令，执行没有意义
  - 假设bne需要4个周期，一次错误的分支预测导致后面4条指令没有意义

```
loop:
1    Load   F0 0 R1
2    MUL    F4 F0 F2
3    Store  F4 0 R1
4    Sub    R1 R1 4
5    Bnez   R1 loop
```

		j	k
1. LD	F0	0	R1
2. MUL	F4	F0	F2
3. SD	F4	0	R1
6. LD	F0	0	R1
7. MUL	F4	F0	F2
8. SD	F4	0	R1

■ 重排序缓冲

□ 重排序缓冲(Reorder Buffer, ROB)

- 按照指令的发送顺序存放指令的执行结果
  - 每个ROB表项包括tag、PC、目标寄存器、结果、意外状态等等
- 指令完成，不立即提交，而是放入ROB中缓存
  - 当其他指令需要ROB缓存中的操作数，像保留站一样提供
- 真正的指令提交
  - 从ROB顶部取出结果，保存为新的计算机状态
  - 比如写入内存、写入寄存器等
- 达成按序发射、乱序执行、按序提交的目的

■ 重排序缓冲

□ 结构设计

- 整个结构类似环形缓冲区，新指令存入Free指针位置，从Oldest指针处提交
- 调度时，就绪的操作数从寄存器堆读取，同时设置p标志位。未就绪的操作数，清除p标志位，复制对应的Tag
- v标志位表明当前指令可用，i标志位表示指令在发射
- 指令完成执行，找到ROB中依赖当前指令的所有Tag，设置p标志位。同时计算结果写入Result，如果有Except发生也记录

Oldest →

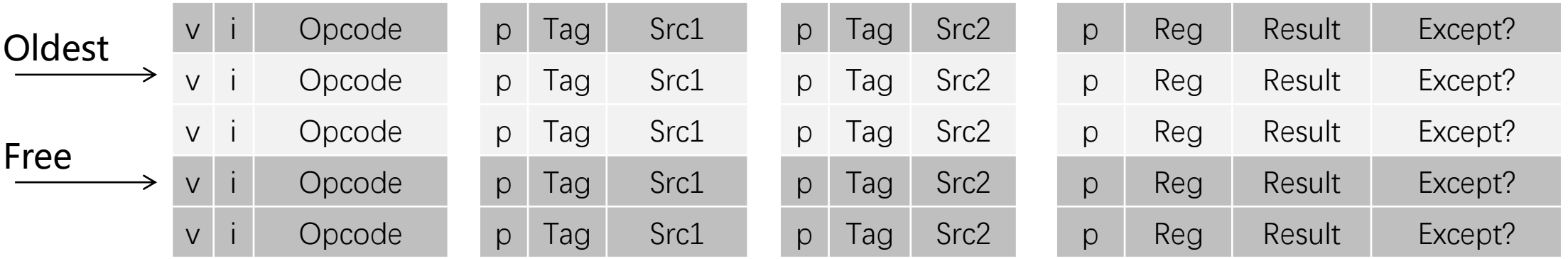
Free →

v	i	Opcode	p	Tag	Src1	p	Tag	Src2	p	Reg	Result	Except?
v	i	Opcode	p	Tag	Src1	p	Tag	Src2	p	Reg	Result	Except?
v	i	Opcode	p	Tag	Src1	p	Tag	Src2	p	Reg	Result	Except?
v	i	Opcode	p	Tag	Src1	p	Tag	Src2	p	Reg	Result	Except?
v	i	Opcode	p	Tag	Src1	p	Tag	Src2	p	Reg	Result	Except?

■ 重排序缓冲

□ 结构设计

- 提交指令结果时，从Oldest开始依次提交，把结果写回寄存器堆
- 如果发生中断，提交指令到Oldest=Free，然后执行中断处理
  - 实际上是提交到发生中断的指令处
  - 这样中断前的指令完全可见



■ 统一物理寄存器堆

□ 统一物理寄存器堆架构(Unified Physical Register File)

- 有ROB后，可以为先前不支持寄存器换名的算法提供换名的支持
  - ➢ 最简单的：直接利用ROB提供换名功能。ROB中的Tag和Src恰好可以构成一个重命名寄存器
    - Tag提供标记确保正确寻址，Src存放数据提供寄存器功能
    - 存在弊端：若一条指令没有目的寄存器，ROB中Result字段、Reg字段就会被浪费
  - ➢ 扩展逻辑寄存器堆：把Result字段和Reg字段剥离，单独提供额外一组寄存器堆，利用重命名映射表记录ROB表项和额外寄存器堆之间关系
    - 和纯ROB的重命名没有区别，只是存放结果的字段从ROB中剥离
    - 这组额外的寄存器堆在ISA层面不可见，称为逻辑寄存器堆(Architecture register file)；相对应有物理寄存器堆(Physical register file)，是CPU执行时真正使用的寄存器
    - 存在弊端：额外增加一组寄存器堆，芯片功耗和面积难以降低。需要额外寄存器堆和普通寄存器堆之间数据转移，增加延迟

■ 统一物理寄存器堆

□ 统一物理寄存器堆架构

- 第二次改进，合并逻辑寄存器堆和物理寄存器堆
  - 扩展逻辑寄存器堆的方法引入额外的一组寄存器堆，导致芯片功耗和面积增加
  - 那就合并，用一组寄存器堆同时实现逻辑寄存器堆和物理寄存器堆功能
  - 分为三个组件
    - 重命名映射表，记录逻辑寄存器和物理寄存器之间对应关系
    - 物理寄存器堆：提供寄存器的数据保存服务
    - 空闲列表：记录物理寄存器堆中空闲寄存器序号
  - 所有ISA中使用的逻辑寄存器，通过访问统一物理寄存器堆架构来访问寄存器
  - 指令提交时修改映射表，从而避免数据在寄存器间转移

■ 统一物理寄存器堆

□ 统一物理寄存器堆

- 寄存器堆负责具体数据的保存
- 寄存器功能从ROB中剥离，ROB不再保存具体的数据

- 思考：何时能够重新利用P3寄存器？假设指令1中的x3当前映射到P3
  - 即何时可以释放x3→P3寄存器**映射关系**?
  - 提示，下边所示的指令序列在当前架构的ROB中执行会产生什么问题

```
1. addi x3, x1, #4
2. sub x6, x3, x9
3. add x3, x7, x6
```

■ 统一物理寄存器堆

- 何时能够重新利用P3寄存器？假设指令1中的x3当前映射到P3
  - 答：下一条写入x3的指令完成提交，即指令3完成提交后
  - 当前指令序列存在的问题
    - 指令3会覆盖指令1关于逻辑寄存器x3→P3的映射关系，导致指令2读取x3时出错
    - 所以需要额外的映射表记录x3之前的映射关系，**在ROB中增加一个字段实现**
  - 为什么是指令3提交之后?
    - ROB按序提交的特性保证指令3提交后，保证指令2也已经执行完，不可能有指令需要访问x3→P3映射条目

```
1. addi x3, x1, #4
2. sub x6, x3, x9
3. add x3, x7, x6
```

■ 统一物理寄存器堆

□ 新设计的ROB结构

- 结合前面提到的改进，删去寄存器功能，增加上一映射关系字段
- 稍后通过一个例子学习新ROB结构的工作原理

v	i	Opcode
v	i	Opcode
v	i	Opcode
v	i	Opcode
v	i	Opcode

p	Tag	Src1
p	Tag	Src1
p	Tag	Src1
p	Tag	Src1
p	Tag	Src1

p	Tag	Src2
p	Tag	Src2
p	Tag	Src2
p	Tag	Src2
p	Tag	Src2

p	Reg	Result	Except?
p	Reg	Result	Except?
p	Reg	Result	Except?
p	Reg	Result	Except?
p	Reg	Result	Except?

v	i	Opcode
v	i	Opcode
v	i	Opcode
v	i	Opcode
v	i	Opcode

p	Tag
p	Tag
p	Tag
p	Tag
p	Tag

p	Tag
p	Tag
p	Tag
p	Tag
p	Tag

p	Reg	LPReg	PReg	Except?
p	Reg	LPReg	PReg	Except?
p	Reg	LPReg	PReg	Except?
p	Reg	LPReg	PReg	Except?
p	Reg	LPReg	PReg	Except?

■ 统一物理寄存器堆

□ 假设表中已经存在部分数据

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

重名名映射表

x0	
x1	P8
x2	
x3	P7
x4	
x5	
x6	P5
x7	P6

物理寄存器

P0	
P1	
P2	
P3	
P4	
P5	<x6>
P6	<x7>
P7	<x3>
P8	<x1>

空闲寄存器队列

P0
P1
P3
P2
P4

v	i	Opcode

p	Tag

p	Tag

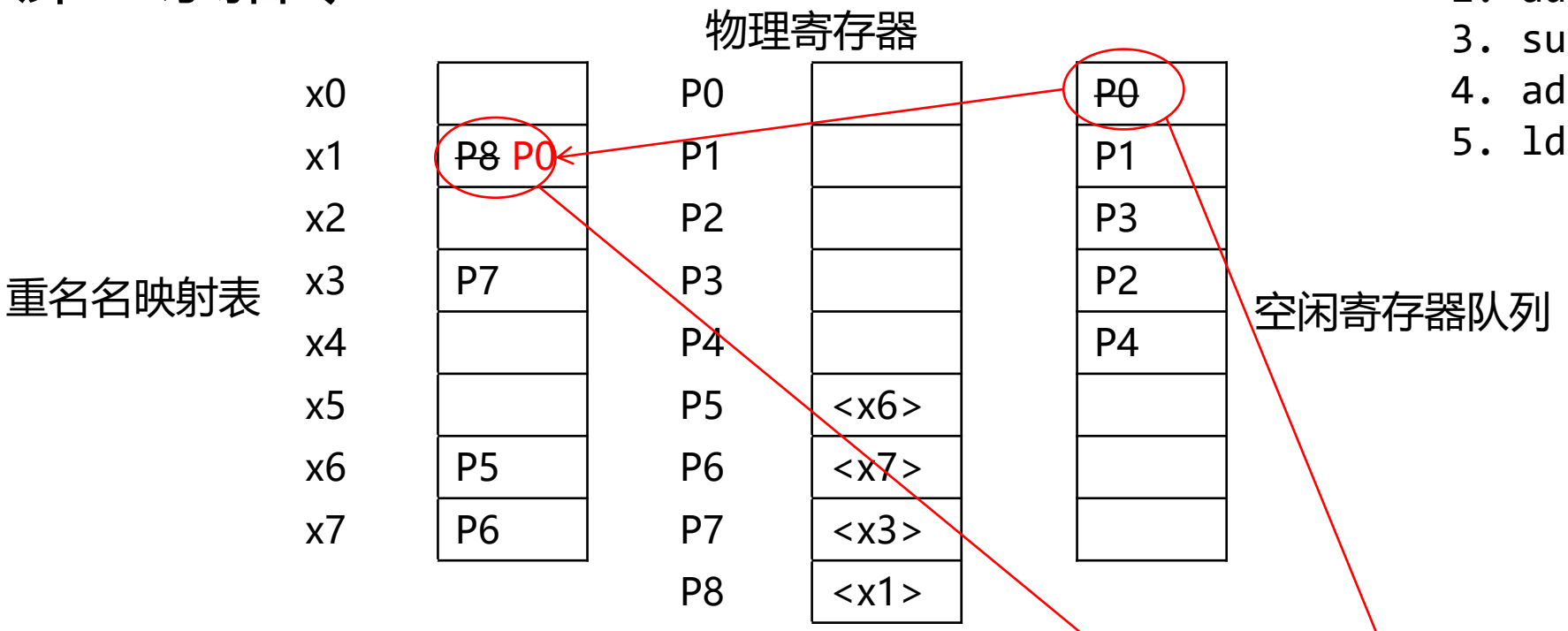
p	vReg	LPReg	PReg

vReg表示指令中寄存器名
LPReg表示上一次映射关系
PReg表示当前寄存器映射关系
Except不考虑

■ 统一物理寄存器堆

□ 读取第一条指令

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```



v	i	Opcode
	Y	ld

p	Tag
Y	P7

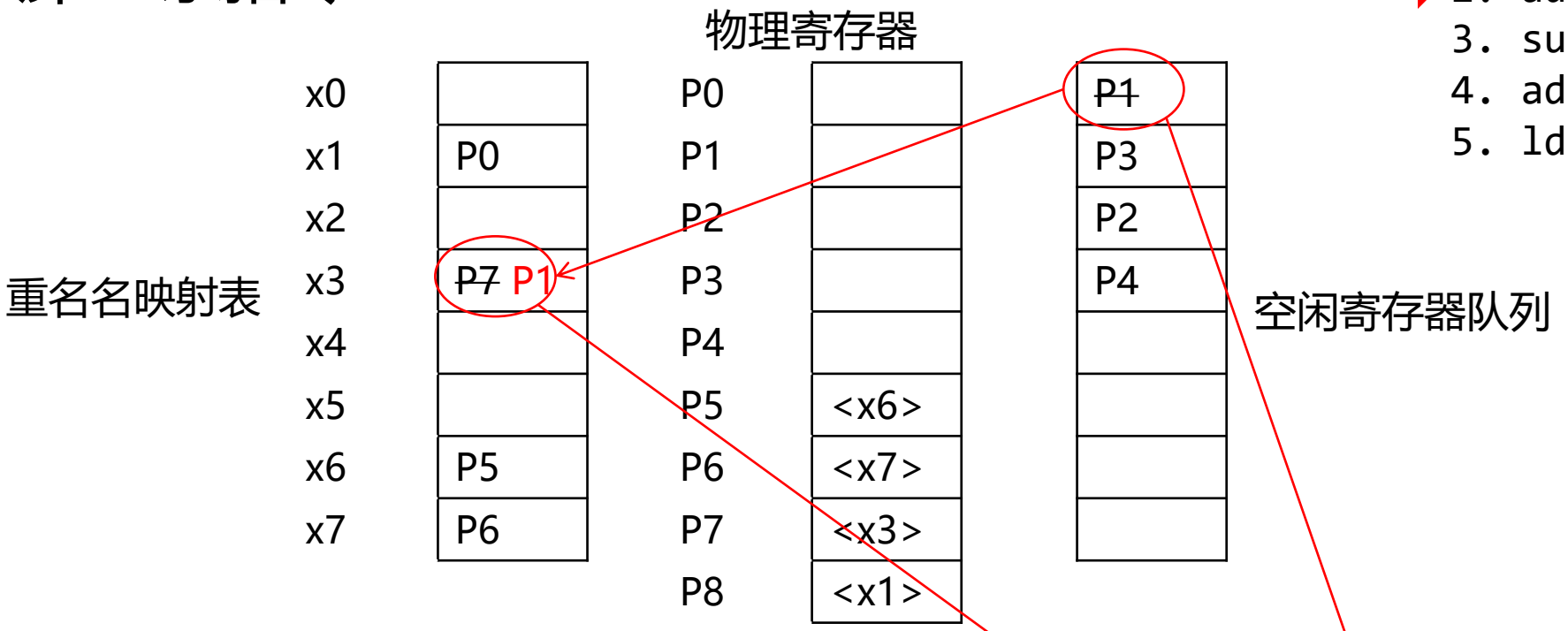
p	Tag

p	vReg	LPReg	PReg
Y	x1	P8	P0

■ 统一物理寄存器堆

□ 读取第二条指令

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```



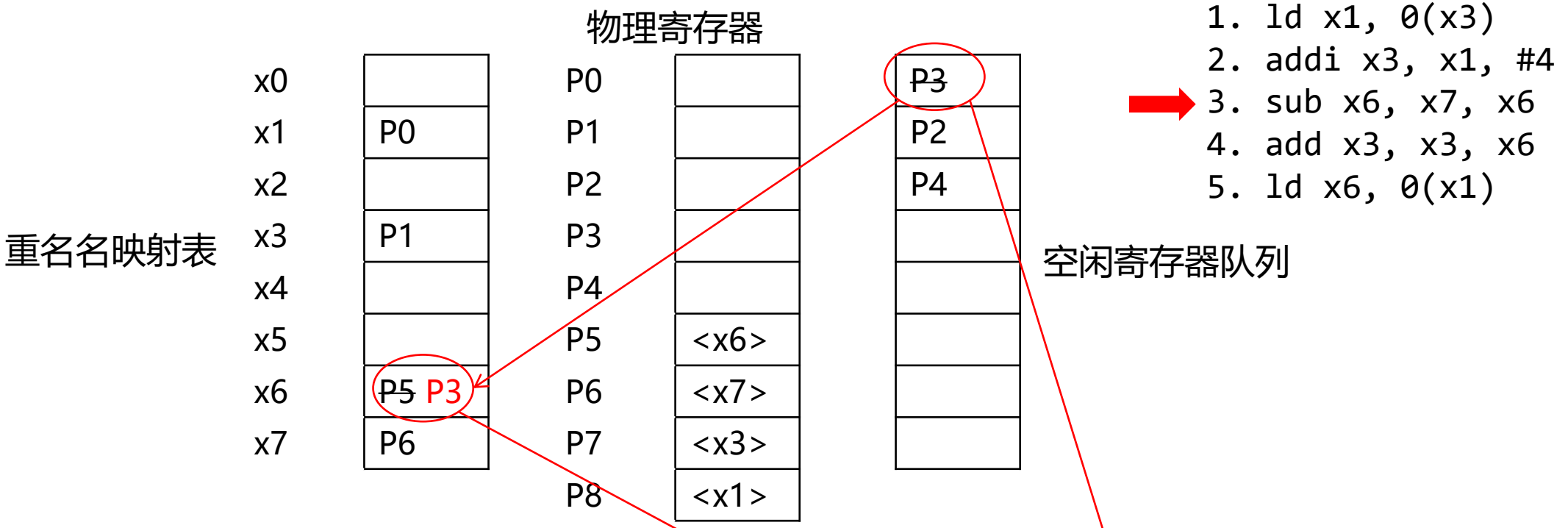
v	i	Opcode
	Y	ld
	Y	addi

p	Tag
Y	P7
N	P0

p	Tag

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1

■ 统一物理寄存器堆



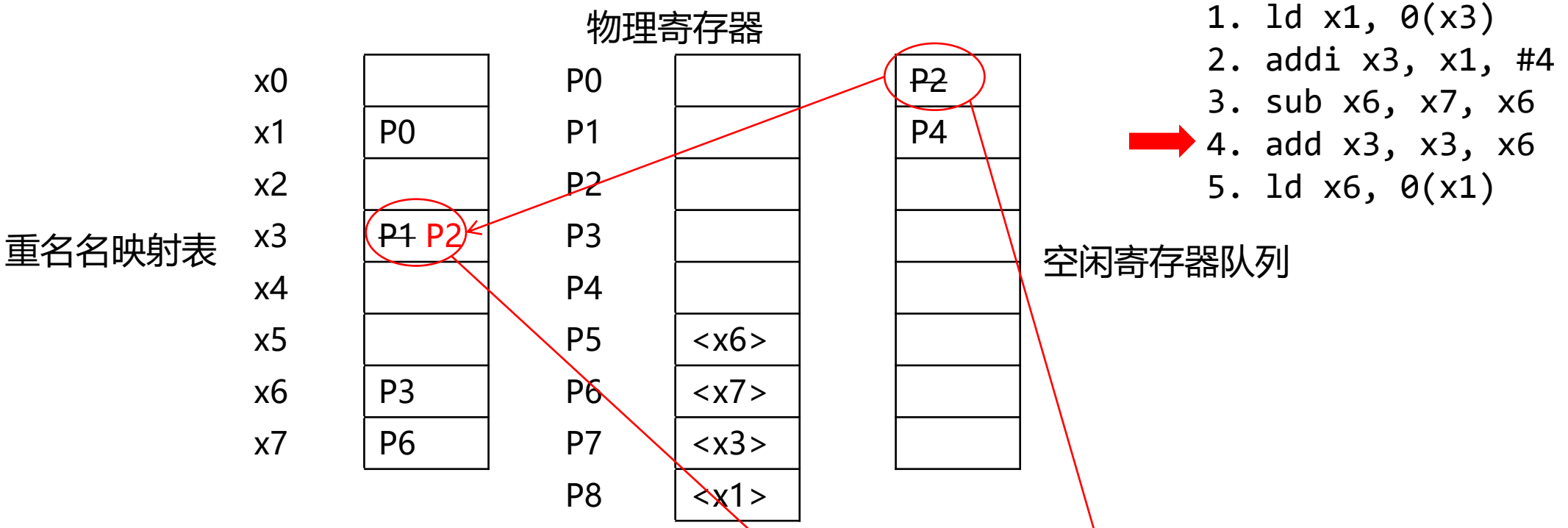
v	i	Opcode
	Y	ld
	Y	addi
	Y	sub

p	Tag
Y	P7
N	P0
Y	P6

p	Tag
Y	P5

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3

■ 统一物理寄存器堆



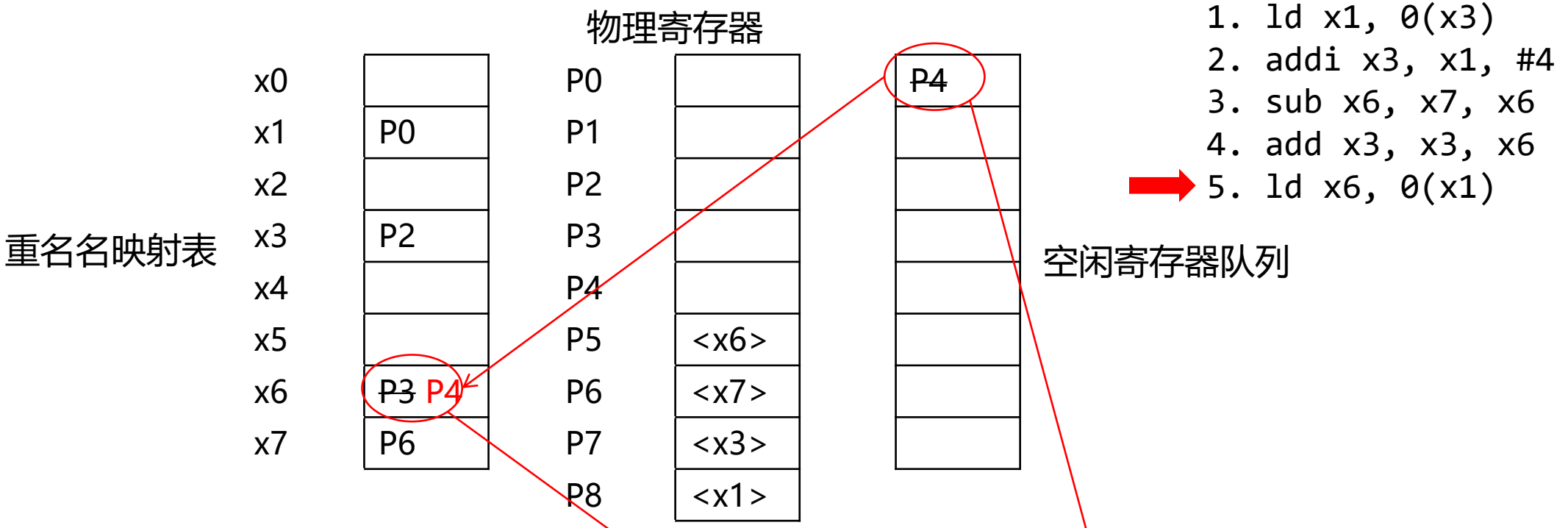
v	i	Opcode
	Y	ld
	Y	addi
	Y	sub
	Y	add

p	Tag
Y	P7
N	P0
Y	P6
N	P1

p	Tag
Y	P5
N	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2

■ 统一物理寄存器堆



v	i	Opcode
	Y	ld
	Y	addi
	Y	sub
	Y	add
	Y	ld

p	Tag
Y	P7
N	P0
Y	P6
N	P1
N	P0

p	Tag
Y	P5
N	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

重名名映射表

x0	
x1	P0
x2	
x3	P2
x4	
x5	
x6	P4
x7	P6

物理寄存器

P0	<x1>
P1	
P2	
P3	
P4	
P5	<x6>
P6	<x7>
P7	<x3>
P8	~~<x1>~~

空闲寄存器队列

P8

v	i	Opcode
Y	Y	ld
	Y	addi
	Y	sub
	Y	add
	Y	ld

p	Tag
Y	P7
Y	P0
Y	P6
N	P1
Y	P0

p	Tag
Y	P5
N	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

物理寄存器

重名名映射表

x0	
x1	P0
x2	
x3	P2
x4	
x5	
x6	P4
x7	P6

P0	<x1>
P1	<x3>
P2	
P3	
P4	
P5	<x6>
P6	<x7>
P7	~~<x3>~~
P8	

空闲寄存器队列

P8
P7

v	i	Opcode
Y	Y	ld
Y	Y	addi
	Y	sub
	Y	add
	Y	ld

p	Tag
Y	P7
Y	P0
Y	P6
Y	P1
Y	P0

p	Tag
Y	P5
N	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

重名名映射表

x0	
x1	P0
x2	
x3	P2
x4	
x5	
x6	P4
x7	P6

物理寄存器

P0	<x1>
P1	<x3>
P2	
P3	<x6>
P4	
P5	~~<x6>~~
P6	<x7>
P7	
P8	

空闲寄存器队列

P8
P7
P5

v	i	Opcode
Y	Y	ld
Y	Y	addi
Y	Y	sub
	Y	add
	Y	ld

p	Tag
Y	P7
Y	P0
Y	P6
Y	P1
Y	P0

p	Tag
Y	P5
Y	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

重名名映射表

x0	
x1	P0
x2	
x3	P2
x4	
x5	
x6	P4
x7	P6

物理寄存器

P0	<x1>
P1	~~<x3>~~
P2	<x3>
P3	<x6>
P4	
P5	
P6	<x7>
P7	
P8	

空闲寄存器队列

P8
P7
P5
P1

v	i	Opcode
Y	Y	ld
Y	Y	addi
Y	Y	sub
Y	Y	add
	Y	ld

p	Tag
Y	P7
Y	P0
Y	P6
Y	P1
Y	P0

p	Tag
Y	P5
Y	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

```
1. ld x1, 0(x3)
2. addi x3, x1, #4
3. sub x6, x7, x6
4. add x3, x3, x6
5. ld x6, 0(x1)
```

重名名映射表

x0	
x1	P0
x2	
x3	P2
x4	
x5	
x6	P4
x7	P6

物理寄存器

P0	<x1>
P1	
P2	<x3>
P3	~~<x6>~~
P4	<x6>
P5	
P6	<x7>
P7	
P8	

空闲寄存器队列

P8
P7
P5
P1
P3

v	i	Opcode
Y	Y	ld
Y	Y	addi
Y	Y	sub
Y	Y	add
Y	Y	ld

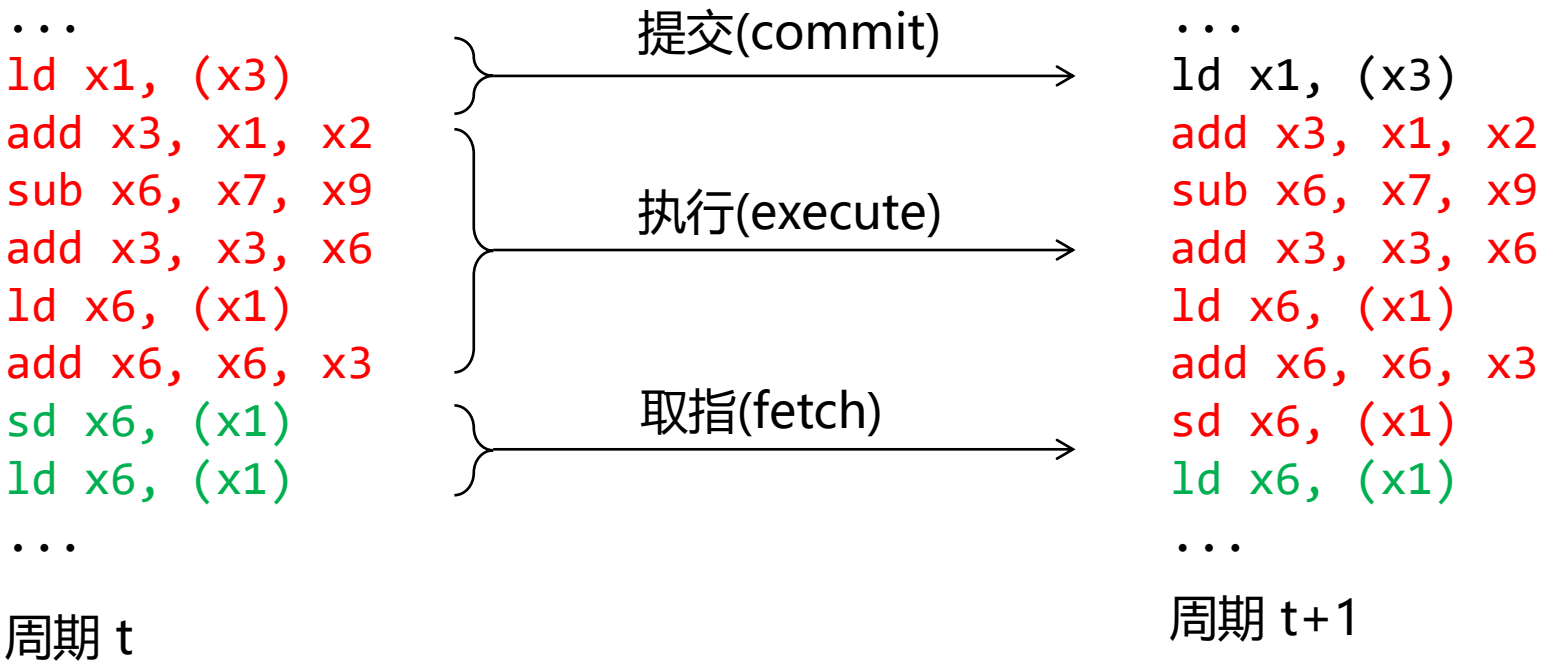
p	Tag
Y	P7
Y	P0
Y	P6
Y	P1
Y	P0

p	Tag
Y	P5
Y	P3

p	vReg	LPReg	PReg
	x1	P8	P0
	x3	P7	P1
	x6	P5	P3
	x3	P1	P2
	x6	P3	P4

■ 统一物理寄存器堆

- 从指令的角度来看，ROB更像是一个滑动窗口



■ 显式寄存器换名

- 基于ROB提供的是隐式寄存器换名
  - 对于程序员，CPU内部的换名表不可见
- 显式寄存器换名
  - 提供的物理寄存器数量大于ISA定义的寄存器数量
  - 关键想法：要求每条指令写入的寄存器都是全新分配的新寄存器
    - 每条需要写入寄存器的指令
      - 从空闲寄存器列表中取出一个寄存器分配给当前指令写入
  - 相对易于实现
    - 可结合普通流水线、Scoreboard或类Tomasulo的实现
    - 不需要复杂的旁路数据通路，比如ROB中数据的转移

■ 小结

□ 重排序缓冲

- 实现指令的按序发射、乱序执行、按序提交
- 支持精确中断/预测执行
  - 预测错误时将分支指令之后所有错误发射的指令丢弃即可
  - 精确中断只需要提交所有在ROB中的，在中断前的指令

感谢！