

Computer Architecture

Course code: 0521292B

08. Exploiting ILP

Jianhua Li

College of Computer and Information
Hefei University of Technology

slides are adapted from CA course of wisc, princeton, mit, berkeley, etc.

The uses of the slides of this course are for educational purposes only and should be used only in conjunction with the textbook. Derivatives of the slides must acknowledge the copyright notices of this and the originals.



内容概要

- 指令级并行的概念
- 指令的动态调度
- 动态分支预测技术
- 多指令流出技术
- 循环展开和指令调度

指令级并行的概念

- 处理机利用流水线将具有潜在并行性的指令重叠并行执行，以提升性能的技术称作指令级并行性(**Instruction-Level Parallelism**)。
- 本节讨论：如何通过各种可能的技术，获得更多的指令级并行性。
 - 硬件+软件：必须要硬件技术和软件技术互相配合，才能够最大限度地挖掘出程序中存在的指令级并行。

指令级并行的概念

- 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

- IPC: Instructions per Cycle

- 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。
- 据统计：程序平均每5~7条指令就会有一个分支。

问题：这意味着什么？

指令级并行的概念

- 循环级并行：使一个循环中的不同循环体并行执行。

- 开发循环体中存在的并行性
 - 最常见、最基本的ILP技术
- 是指令级并行研究的重点之一
- 例如，考虑下述语句：

```
for (i=1; i<=500; i++)
```

```
    a[i]=a[i]+s;
```

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

指令级并行的概念

- 最基本的开发循环级并行的技术
 - 循环展开 (loop unrolling)
 - 采用SIMD方式来处理
- 相关与流水线冲突
 - 流水线冲突是指对于具体的流水线来说，相关导致指令流中的下一条指令不能在指定的时钟周期执行。
 - 流水线冲突类型：结构冲突、数据冲突、控制冲突
 - 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。

指令级并行的概念

- 可以从两个方面来处理相关问题：
 - 保持相关，但避免发生冲突；
 - 通过代码变换，消除相关；
- 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。
 - 由于相关的存在可能影响程序结果，所以必须保持程序顺序。

指令级并行的概念

- 控制相关并不是一个必须严格保持的关键属性。
- 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：数据流和异常行为。
 - 保持异常行为：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。
 - 为了提升性能可以弱化为：改变指令的执行顺序不能导致程序中发生新的异常。
 - 如果我们能做到保持程序的数据相关和控制相关，就能保持程序的异常行为。

指令级并行的概念

- 示例：

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

必须保持对R2的数据相关，否则会影响结果！

L1 :

- 数据流：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输入可以有多个来源。
 - 仅仅保持数据相关性是不够的，只有再加上保持控制相关，才能够保持程序顺序。（与保持异常行为的要求一样）

指令级并行的概念

- 示例：

DADDU R1, R2, R3

BEQZ R4, L1

DSUBU R1, R5, R6

L1 : ...

OR R7, R1, R8

R1的值依赖于数据流，
数据流（之前多条指令），
具体由分支指令决定！

指令级并行的概念

- 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。
- 示例：

	DADDU	R1, R2, R3
	BEQZ	R12, Skipnext
	DSUBU	R4, R5, R6
	DADDU	R5, R4, R9
skip:	OR	R7, R8, R9

- 假设 skip后R4不再使用，且DSUBU不产生异常，则可以将DSUBU调度到分支之前。

内容概要

- 指令级并行的概念
- 指令的动态调度
- 动态分支预测技术
- 多指令流出技术
- 循环展开和指令调度

基本方法

- 原理：基于编译器充分开发指令之间存在的并行性，找出**不相关的指令序列**，让它们在流水线上重叠并行执行。
- 增加指令间并行性最简单和最常用的方法
 - 开发循环级并行性——循环的不同迭代之间存在的并行性。
 - 在把循环展开后，通过寄存器重命名和指令调度来开发更多的并行性。

循环展开和指令调度

- 编译器完成这种指令调度的能力受限于两个特性：
 - 程序固有的指令级并行性
 - 流水线功能部件的执行延迟
- 假设：我们使用具有如下延迟的浮点流水线：

产生结果的指令	使用结果的指令	延迟（时钟周期数）
浮点计算	另一个浮点计算	3
浮点计算	浮点store（S.D）	2
浮点load（L.D）	浮点计算	1
浮点load（L.D）	浮点store（S.D）	0

循环展开和指令调度

- 假设：采用前面介绍的MIPS 五段流水线：
 - 分支的延迟：1个时钟周期
 - 整数load指令的延迟：1个时钟周期
 - 整数运算部件是全流水或者重复设置了足够的份数（无结构冲突）
- 基于上面的二个假设，看如下问题：

循环展开和指令调度

- 例1. 对于下面的代码，转换成MIPS汇编语言，在不进行指令调度和进行指令调度两种情况下，分析其代码一次循环所需的时间。

```
for (i=1; i<=1000; i++)
```

```
    x[i] = x[i] + s;
```

解： 把该程序翻译成MIPS汇编语言代码：

假设基于如下的寄存器分配：

- R1的初值是指向第一个元素
- 8 (R2) 指向最后一个元素。
- 浮点寄存器F2：用于保存常数s。

循环展开和指令调度

```
Loop:  L.D      F0, 0(R1)
        ADD.D   F4, F0, F2
        S.D     F4, 0(R1)
        DADDIU  R1, R1, #-8
        BNE     R1, R2, Loop
```

其中：

整数寄存器R1：指向向量中的当前元素。（初值为向量中最后一个元素的地址）

循环展开和指令调度

- 不进行指令调度的情况下，程序的实际执行情况：

		指令流出时钟
Loop: L.D	F0, 0(R1)	1
(空转)		2
ADD.D	F4, F0, F2	3
(空转)		4
(空转)		5
S.D	F4, 0(R1)	6
DADDIU	R1, R1, # -8	7
(空转)		8
BNE	R1, R2, Loop	9
(空转)		10


- 每个元素的操作需要10个时钟周期，其中5个是空转周期。

循环展开和指令调度

- 如果进行如下的指令调度：
 - 把DADDIU指令调度到L.D指令和ADD.D指令之间的“空转”拍；
 - 把S.D指令放到了分支指令的延迟槽中；
 - 对存储器地址偏移量进行调整；

循环展开和指令调度

Loop: L. D F0, 0 (R1)
 (空转)
 ADD. D F4, F0, F2
 (空转)
 (空转)
 S. D F4, 0 (R1)
 DADDIU R1, R1, #-8
 (空转)
 BNE R1, R2, Loop
 (空转)



Loop: L. D F0, 0(R1)
 DADDIU R1, R1, #-8
 ADD. D F4, F0, F2
 (空转)
 BNE R1, R2, Loop
 S. D F4, 8(R1)



循环展开和指令调度

指令流出时钟

Loop: L.D	F0, 0(R1)	1
DADDIU	R1, R1, #-8	2
ADD.D	F4, F0, F2	3
(空转)		4
BNE	R1, Loop	5
S.D	F4, 8(R1)	6

- 一轮循环的操作时间从10个时钟周期减少到6个, 其中5个周期是有指令执行的, 1个为空转周期。

循环展开和指令调度

- 上述调度的问题及解决方案

- 问题：只有L. D、ADD. D和S. D这3条指令是有效操作。
 - 占用3个时钟周期。
 - 而DADDIU、空转和BEN这3个时钟周期都是附加的循环控制开销
- 解决方案：循环展开技术
 - 把循环体的代码复制多次并按顺序排列，然后相应调整循环的结束条件。
 - 这给编译器进行指令调度带来了更大的空间

循环展开和指令调度

- **例2.** 将上述例子中的循环展开4次得到4个循环体，然后对展开后的指令序列在不调度和调度两种情况下，分析代码的性能。
 - 假定R1的初值为32的倍数，即循环次数为4的倍数。
 - 消除冗余的指令，并且不要重复使用寄存器。

解：无需在循环体后面增加补偿代码，分配寄存器（不重复使用寄存器）如下：

- F0、F4：用于展开后的第1个循环体
- F2：保存常数s
- F6、F8：第2个循环体
- F10、F12：第3个循环体
- F14、F16：第4个循环体

- 展开后没有调度的代码如下：

指令流出时钟				指令流出时钟			
Loop:	L. D	F0, 0 (R1)	1	ADD. D	F12, F10, F2	15	
	(空转)		2	(空转)		16	
	ADD. D	F4, F0, F2	3	(空转)		17	
	(空转)		4	S. D	F12, -16 (R1)	18	
	(空转)		5	L. D	F14, -24 (R1)	19	
	S. D	F4, 0 (R1)	6	(空转)		20	
	L. D	F6, -8 (R1)	7	ADD. D	F16, F14, F2	21	
	(空转)		8	(空转)		22	
	ADD. D	F8, F6, F2	9	(空转)		23	
	(空转)		10	S. D	F16, -24 (R1)	24	
	(空转)		11	DADDIU	R1, R1, #-32	25	
	S. D	F8, -8 (R1)	12	(空转)		26	
	L. D	F10, -16 (R1)	13	BNE	R1, R2, Loop	27	
	(空转)		14	(空转)		28	

循环展开和指令调度

- 分析:

- 这个循环每遍共使用了28个时钟周期。
- 有4个循环体，完成4个元素的操作。平均每个元素

In conclusion, it's not efficient.

间：从减少循环控制的开销中获得。

- 在整个展开后的循环中，实际用于计算的指令周期只有14个，其他14个周期都是空转。

- 对指令序列进行优化调度，以减少空转周期：

			指令流出时钟
Loop:	L. D	F0, 0(R1)	1
	L. D	F6, -8(R1)	2
	L. D	F10, -16(R1)	3
	L. D	F14, -24(R1)	4
	ADD. D	F4, F0, F2	5
	ADD. D	F8, F6, F2	6
	ADD. D	F12, F10, F2	7
	ADD. D	F16, F14, F2	8
	S. D	F4, 0(R1)	9
	S. D	F8, -8(R1)	10
	DADDIU	R1, R1, #-32	12
	S. D	F12, 16(R1)	11
	BNE	R1, R2, Loop	13
	S. D	F16, 8(R1)	14

循环展开和指令调度

- 分析:

- 没有数据相关引起的空转等待。
 - 整个循环仅仅使用了14个时钟周期。
 - 平均每个元素的操作使用 $14/4=3.5$ 个时钟周期。

- 结论:

- 通过循环展开、寄存器重命名和指令调度，可以有效地开发出指令级并行。

循环展开和指令调度

- 循环展开和指令调度时要注意以下几个方面：

- 保证正确性。

- 在循环展开和调度过程中尤其要注意两个地方的正确性：
循环控制，操作数偏移量的修改。

- 注意有效性。

- 只有能够找到不同循环体之间的无关性，才能有效地使用循环展开。

- 使用不同的寄存器, 否则可能导致新的冲突。

- 注意对存储器数据的相关性分析

- 注意新的相关性

- 由于原循环不同次的迭代在展开后都到了同一次循环体中，因此可能带来新的相关性。

内容概要

- 指令级并行的概念
- 指令的动态调度
- 动态分支预测技术
- 多指令流出技术
- 循环展开和指令调度

指令的动态调度

- 静态调度

- 依靠编译器对代码进行调度，以减少相关和冲突。
- 它不是在程序执行的过程中，而是在编译期间进行代码调度和优化。
- 通过把相关的指令拉开距离来减少可能产生的停顿。

- 动态调度

- 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。

指令的动态调度

- **动态调度的优点：**

- 能够处理一些在编译时情况不明的相关（比如涉及到存储器访问的相关），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。

- **动态调度的缺点：**

- 动态调度所需要的硬件比较复杂，开销较大。

指令的动态调度

- 到目前为止我们所使用流水线的最大的局限性

- 指令必须按序流出和执行

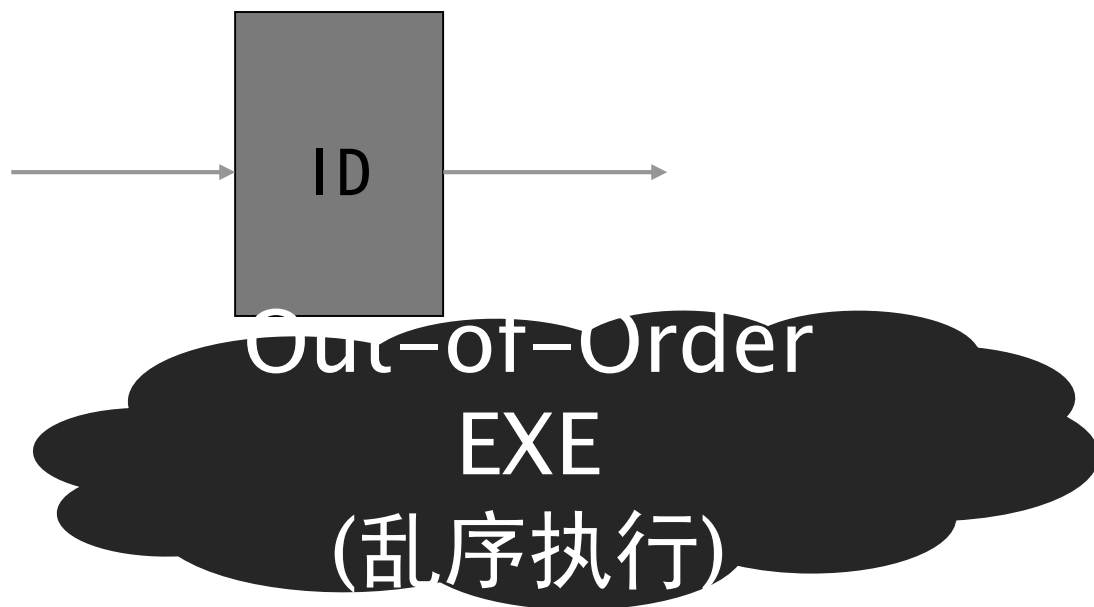
- 考虑下面一段代码：

```
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指令与流水线中的指令都没有关系，但也因此受阻；

指令的动态调度

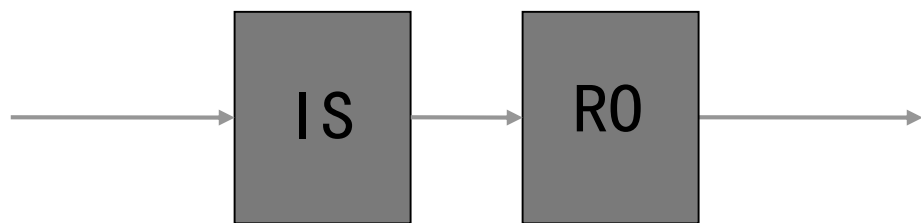
- 在前面的基本流水线中：



- 一旦一条指令受阻，其后的指令都将停顿，这种处理策略会限制流水线的性能。

指令的动态调度

- 为了乱序执行，需要将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，检查是否存在结构冲突。
 - 读操作数 (Read Operands, RO)
 - 等待数据冲突消失，然后读操作数。



检测结构冲突 检测数据冲突

指令的动态调度

- 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。

例如，考虑下面的代码

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

- Tomasulo算法可以通过使用寄存器重命名来消除上面的相关，从而提升指令执行效率。
 - 后面会介绍该算法

指令的动态调度

- I/O device request
- Invoking an operating system service from a user program
- Tracing instruction execution
- Breakpoint (programmer-requested interrupt)
- Integer arithmetic overflow
- FP arithmetic anomaly
- Page fault (not in main memory)
- Misaligned memory accesses (if alignment is required)
- Memory protection violation
- Using an undefined or unimplemented instruction
- Hardware malfunctions
- Power failure

指令的动态调度

- 动态调度让指令乱序完成，会大大增加异常处理的难度。
- 动态调度需要保持正确的异常行为：
 - 对于一条会产生异常的指令来说，只有当处理机确切地知道该指令将被执行后，才允许它产生异常。
 - 即使保持了正确的异常行为，动态调度处理机仍可能发生不精确异常。

指令的动态调度

- 不精确异常：
 - 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指令*i*发生异常的现场不同。
 - 当指令*i*发生异常时：
 - 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。
 - 不精确异常使得在异常处理后难以接着继续执行程序。
- 精确异常：
 - 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。
 - 如何实现精确异常？请参考Smith, et al. ISCA 85

记分板调度算法

- 例：数据先读后写（WAR）相关引起的阻塞代码序列：

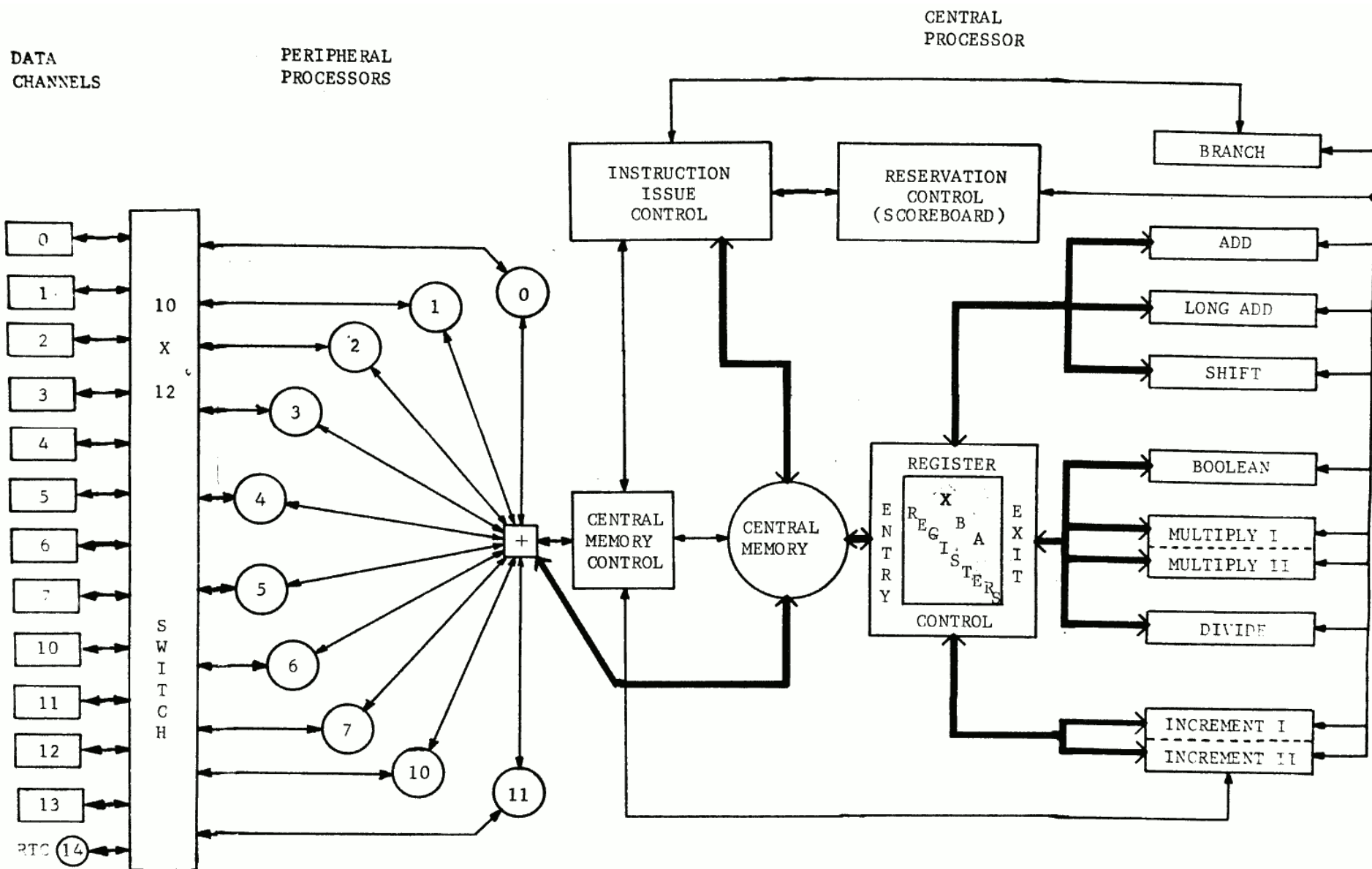
```
DIVD    F0 , F2 , F4  
ADDD    F10 , F0 , F8  
SUBD    F8 , F8 , F14
```

- 指令乱序执行时就会出现先读后写冒险。
- 动态调度之记分板调度算法
 - 目标：在资源充足时，尽可能早地执行没有数据阻塞的指令，达到每个时钟周期执行一条指令。

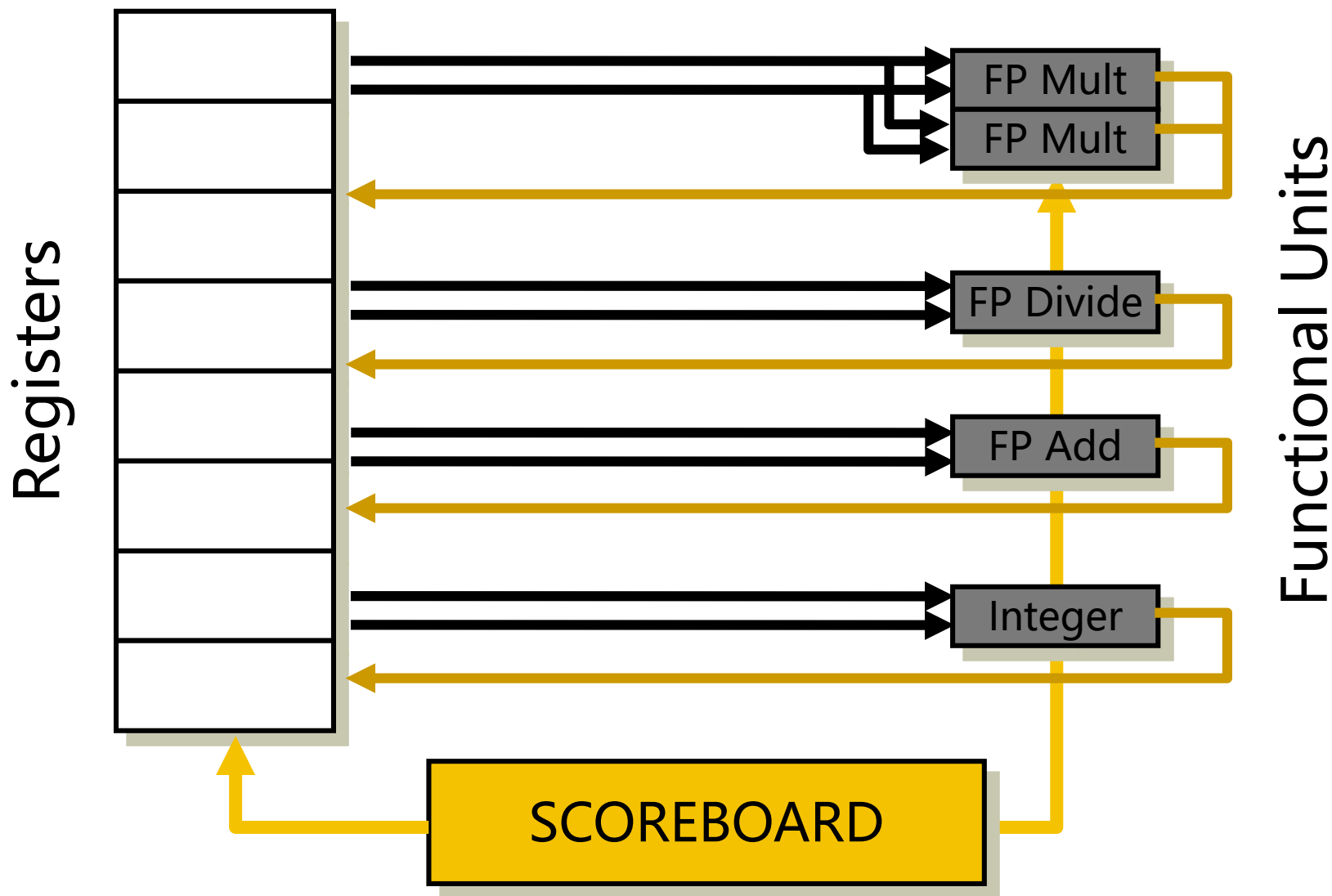
CDC 6600



CDC 6600 结构简图



记分板体系结构



记分板的含义

- 乱序完成 \Rightarrow WAR, WAW冒险?
- 对WAR的解决方案:
 - 排队等待操作以及它们操作数的拷贝
 - 只在读操作段才读取寄存器
- 对WAW的解决方案, 必须检测冒险: 暂停等待到其他指令完成;
- 在执行阶段可能有多个指令 \Rightarrow 设置多个执行部件或者流水化执行部件;
- 记分板跟踪相关、状态或操作;
- 记分板用四个流水段代替ID、EX、WB三段;

记分板控制的四级

1. Issue—decode instructions & check for structural hazards.

If ❶ a functional unit for the instruction is **free** and ❷ no other active instruction has the same destination register (**WAW**), the scoreboard issues the instruction to the functional unit and updates its internal data structure. If a structural or WAW hazard exists, then the instruction issue **stalls**, and no further instructions will issue until these hazards are cleared (**in-order issue**).

2. Read operands—wait until no data hazards, then read operands.

❸ A source operand is available if **no earlier issued** active instruction is going to write it, or if the register containing the operand is being written by a **currently active** functional unit. When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves **RAW** hazards dynamically in this step, and **instructions may be sent into execution out of order**.

记分板控制的四级

3. Execution—operate on operands (EX)

The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

4. Write result—finish execution (WB)

Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. ④ If **WAR**, then it stalls the instruction.

Example:

```
DIVD  F0,F2,F4
ADDD  F10,F0,F8
SUBD  F8,F8,F14
```

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

记分板需要记录的信息

1. Instruction status

- which of 4 steps the instruction is in

2. Functional unit status—Indicates the state of the functional unit (FU).

Busy — Indicates whether the unit is busy or not

Op — Operation to perform in the unit (e.g., + or -)

Fi — Destination register

Fj, Fk — Source-register numbers

Qj, Qk— Functional units producing source registers Fj, Fk

Rj, Rk— Flags indicating when Fj, Fk are ready

3. Register result status

- Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register.

记分板示例 第0周期

<u>Instruction status</u>				<i>Read</i>	<i>Exec</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>		<i>Issue</i>	<i>opera</i>	<i>compl</i>	<i>Result</i>
LD	F6	34+	R2				
LD	F2	45+	R3				
MUL	F0	F2	F4				
SUB	F8	F6	F2				
DIV	F10	F0	F6				
ADD	F6	F8	F2				

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>		
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	$F0$	$F2$	$F4$	$F6$	$F8$	$F10$	$F12$...	$F30$
FU									

记分板示例 第1周期

Instruction status: Read Execu Write
Instruction j k Issue operat compl Result

LD	F6	34+	R2	1
LD	F2	45+	R3	
MUL	F0	F2	F4	
SUB	F8	F6	F2	
DIV	F10	F0	F6	
ADD	F6	F8	F2	

Functional unit status dest S1 S2 FU for FU for Fj? Fk?

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
1				Integer					

记分板示例 第2个周期

Instruction status Read Execu Write
 Instruction *j* *k* Issue operat compl Result

LD F6 34+ R2

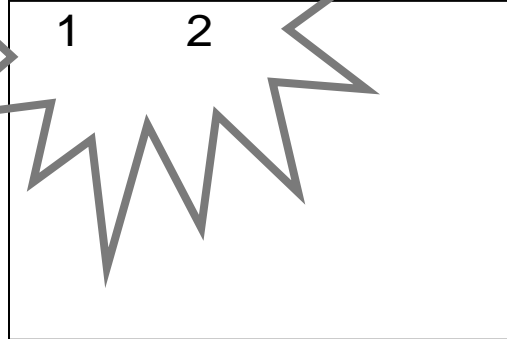
LD F2 45+ R3

MUL F0 F2 F4

SUB F8 F6 F2

DIV F10 F0 F6

ADD F6 F8 F2



• Issue 2nd LD?

Functional unit status *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*

Time *Name* *Busy* *Op* *Fi* *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer Yes Load F6 R2 Yes

Mult1 No

Mult2 No

Add No

Divide No

Register result status

Clock *F0* *F2* *F4* *F6* *F8* *F10* *F12* ... *F30*

2

FU

Integer

记分板示例 第3周期

Instruction status				Read Execution Write			
Instruction	<i>j</i>		<i>k</i>	Issue	operands	complete	Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

</

• Issue MULT?

记分板示例 第4周期

Instruction status


Instruction	j	k	Read	Execu	Write
			Issue	operat	compl Result
LD F6 34+ R2			1	2	3 4
LD F2 45+ R3					
MUL F0 F2 F4					
SUB F8 F6 F2					
DIV F10 F0 F6					
ADD F6 F8 F2					

Functional unit status

	dest	S1	S2	FI for FI for Fj?	Fk?
Time Name	Busy Op	Fi	Fj	Fk	Qj Qk Rj Rk
Integer	No				
Mult1	No				
Mult2	No				
Add	No				
Divide	No				

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
4 FU									



记分板示例 第5周期

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Read</i>	<i>Execu</i>	<i>Write</i>
			<i>Issue</i>	<i>operat</i>	<i>compl</i>
			<i>Result</i>		

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MUL	F0	F2	F4				
SUB	F8	F6	F2				
DIV	F10	F0	F6				
ADD	F6	F8	F2				

Functional unit status

	<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>
	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>

Time Name

Busy Op

Fi

Fj

Fk

Qj

Qk

Rj

Rk

Integer

Yes

Load

F2

R3

Yes

Mult1

No

Mult2

No

Add

No

Divide

No

Register result status

Clock

F0

F2

F4

F6

F8

F10

F12

...

F30

5

FU

Integer

记分板示例 第6周期

<u>Instruction status</u>				<i>Read</i>	<i>Execu</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>		<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MUL	F0	F2	F4	6			
SUB	F8	F6	F2				
DIV	F10	F0	F6				
ADD	F6	F8	F2				

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes Load	F2		R3				Yes
	Mult1	Yes Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No							
	Add	No							
	Divide	No							

Register result status		$F0$	$F2$	$F4$	$F6$	$F8$	$F10$	$F12$...	$F30$
Clock										
6	FU	Mult	Integer							

记分板示例 第7周期

Instruction status				Read	Execu	Write				
Instruction	<i>j</i>	<i>k</i>		Issue	operat	compl	Result			
LD	F6	34+	R2	1	2	3	4			
LD	F2	45+	R3	5	6	7				
MUL	F0	F2	F4	6						
SUB	F8	F6	F2	7						
DIV	F10	F0	F6							
ADD	F6	F8	F2							
Functional unit status				dest	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>
Time	Name	Busy	Op	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								
Register result status										
Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
7	<i>FU</i>	Mult1	Integer			Add				

- Read multiply operands?

记分板示例 第8a周期(前半周期)

Instruction status

Instruction *j* *k* *Read* *Execu* *Write*
Issue *operat* *compl* *Result*

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MUL	F0	F2	F4	6			
SUB	F8	F6	F2	7			
DIV	F10	F0	F6	8			
ADD	F6	F8	F2				

Functional unit status

Time *Name* *Busy* *Op* *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*
Fi *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer	Yes	Load	F2		R3				No
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	Integer			Add	Divide			

记分板示例 第8b周期(后半周期)

Instruction status Read Execu Write
Instruction j k Issue opera compl Result

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6			
SUB	F8	F6	F2	7			
DIV	F10	F0	F6	8			
ADD	F6	F8	F2				

Functional unit status dest S1 S2 FU for FU for Fj? Fk?
Time Name Busy Op Fi Fj Fk Qj Qk Rj Rk

Integer	No									
Mult1	Yes	Mult	F0	F2	F4				Yes	Yes
Mult2	No									
Add	Yes	Sub	F8	F6	F2				Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1			No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1				Add	Divide			

记分板示例 第9周期

Instruction status				Read	Execu	Write
Instruction	j	k	Issue	operat	compl	Result
LD F6 34+ R2			1	2	3	4
LD F2 45+ R3			5	6	7	8
MUL F0 F2 F4			6	9		
SUB F8 F6 F2			7	9		
DIV F10 F0 F6			8			
ADD F6 F8 F2						

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

• Read operands for MULT & SUBD? Issue ADDD?

Integer	No								
10 Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
Mult2	No								
2 Add	Yes	Sub	F8	F6	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1				Add	Divide			

记分板示例 第10周期

Instruction status

Instruction *j* *k* *Issue* *operat* *compl* *Result*

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9		
SUB	F8	F6	F2	7	9		
DIV	F10	F0	F6	8			
ADD	F6	F8	F2				

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

Functional unit status

Time *Name*

Busy *Op* *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*
Fi *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer	No								
9 Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
1 Add	Yes	Sub	F8	F6	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1				Add	Divide			

记分板示例 第11周期

Instruction status				Read	Execu	Write
Instruction	<i>j</i>	<i>k</i>		Issue	operat	compl Result
LD F6	34+	R2		1	2	3 4
LD F2	45+	R3		5	6	7 8
MUL F0	F2	F4		6	9	
SUB F8	F6	F2		7	9	11
DIV F10	F0	F6		8		
ADD F6	F8	F2				

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

<u>Functional unit status</u>			<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
11	<i>FU</i>	Mult1				Add	Divide			

记分板示例 第12周期

Instruction status

Instruction *j* *k* *Issue* *operat* *compl* *Result*

LD F6 34+ R2

LD F2 45+ R3

MUL F0 F2 F4

SUB F8 F6 F2

DIV F10 F0 F6

ADD F6 F8 F2

Read *Execu* *Write*

1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			

- Read operands for DIVD?

Functional unit status

Time *Name*

Busy *Op* *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*
Fi *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer

7 Mult1

Mult2

Add

Divide

No								
Yes	Mult	F0	F2	F4			No	No
No								
No								
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock

F0 F2 F4 F6 F8 F10 F12 ... F30

12

FU

Mult1

Divide

记分板示例 第13周期

Instruction status				Read	Execu	Write
Instruction	j	k		Issue	operat	compl Result
LD F6 34+ R2				1	2	3 4
LD F2 45+ R3				5	6	7 8
MUL F0 F2 F4				6	9	
SUB F8 F6 F2				7	9	11 12
DIV F10 F0 F6				8		
ADD F6 F8 F2				13		

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No							
6	Mult1	Yes Mult	F0	F2	F4			No	No
	Mult2	No							
	Add	Yes Add	F6	F8	F2			Yes	Yes
	Divide	Yes Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1			Add		Divide			

记分板示例 第14周期

Instruction status				Read	Execu	Write				
Instruction	j	k		Issue	operat	compl	Result			
LD	F6	34+	R2	1	2	3	4			
LD	F2	45+	R3	5	6	7	8			
MUL	F0	F2	F4	6	9					
SUB	F8	F6	F2	7	9	11	12			
DIV	F10	F0	F6	8						
ADD	F6	F8	F2	13	14					
Functional unit status				dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes
Register result status										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1			Add		Divide			

记分板示例 第15周期

Instruction status				Read Execu Write			
Instruction	j	k		Issue	operat	compl	Result
LD F6 34+ R2				1	2	3	4
LD F2 45+ R3				5	6	7	8
MUL F0 F2 F4				6	9		
SUB F8 F6 F2				7	9	11	12
DIV F10 F0 F6				8			
ADD F6 F8 F2				13	14		

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No							
4	Mult1	Yes Mult	F0	F2	F4			No	No
	Mult2	No							
1	Add	Yes Add	F6	F8	F2			No	No
	Divide	Yes Div	F10	F0	F6	Mult1		No	Yes

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
15	FU	Mult1			Add		Divide			

记分板示例 第16周期

Instruction status				Read Execu Write			
Instruction	<i>j</i>	<i>k</i>		Issue	operat	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9		
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8			
ADD	F6	F8	F2	13	14	16	

Functional unit status			<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for Fj?</i>	<i>FU for Fk?</i>
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>
	Integer	No					
3	Mult1	Yes Mult	F0	F2	F4		No No
	Mult2	No					
0	Add	Yes Add	F6	F8	F2		No No
	Divide	Yes Div	F10	F0	F6	Mult1	No Yes

Register result status		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock										
16	FU	Mult1			Add		Divide			

记分板示例 第17周期

Instruction status Read Execu Write
Instruction *j* *k* Issue operat compl Result

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9		
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8			
ADD	F6	F8	F2	13	14	16	

• Write result of ADDD?

Functional unit status *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*
Time *Name* *Busy* *Op* *Fi* *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer	No									
2 Mult1	Yes	Mult	F0	F2	F4				No	No
Mult2	No									
Add	Yes	Add	F6	F8	F2				No	No
Divide	Yes	Div	F10	F0	F6	Mult1			No	Yes

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
17	<i>FU</i>	Mult1			Add		Divide			

记分板示例 第18周期

Instruction status Read Execu Write
Instruction *j* *k* Issue operat compl Result

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9		
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8			
ADD	F6	F8	F2	13	14	16	

Functional unit status *dest* *S1* *S2* *FU for* *FU for* *Fj?* *Fk?*
Time *Name* *Busy* *Op* *Fi* *Fj* *Fk* *Qj* *Qk* *Rj* *Rk*

Integer	No									
1 Mult1	Yes	Mult	F0	F2	F4				No	No
Mult2	No									
Add	Yes	Add	F6	F8	F2				No	No
Divide	Yes	Div	F10	F0	F6	Mult1			No	Yes

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
18	<i>FU</i>	Mult1			Add		Divide			

记分板示例 第19周期

Instruction status				Read Execu Write			
Instruction	<i>j</i>	<i>k</i>		Issue	operat	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9	19	
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8			
ADD	F6	F8	F2	13	14	16	

Functional unit status			<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock										
19	FU	Mult1			Add		Divide			

记分板示例 第20周期

Instruction status				Read Execu Write			
Instruction	<i>j</i>	<i>k</i>		Issue	operat	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9	19	20
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8			
ADD	F6	F8	F2	13	14	16	

Functional unit status			<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock										
20	FU				Add		Divide			

记分板示例 第21周期

Instruction status				Read Execu Write			
Instruction	j	k		Issue	operat	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9	19	20
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8	21		
ADD	F6	F8	F2	13	14	16	

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
21	FU				Add		Divide			

记分板示例 第22周期

<u>Instruction status</u>				<i>Read</i>	<i>Execu</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>		<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL	F0	F2	F4	6	9	19	20
SUB	F8	F6	F2	7	9	11	12
DIV	F10	F0	F6	8	21		
ADD	F6	F8	F2	13	14	16	22

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No							
	Mult1	No							
	Mult2	No							
	Add	No							
40	Divide	Yes Div	F10	F0	F6			No	No

Register result status

Clock	$F0$	$F2$	$F4$	$F6$	$F8$	$F10$	$F12$...	$F30$
22	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">FU</div> <div style="border: 1px solid black; flex-grow: 1; position: relative;"> <div style="position: absolute; top: -10px; left: 50%; transform: translateX(-50%);">Divide</div> </div> </div>								

记分板示例 第61周期

Instruction status				Read Execu Write						
Instruction	j	k		Issue	operat	compl	Result			
LD F6	34+	R2		1	2	3	4			
LD F2	45+	R3		5	6	7	8			
MUL F0	F2	F4		6	9	19	20			
SUB F8	F6	F2		7	9	11	12			
DIV F10	F0	F6		8	21	61				
ADD F6	F8	F2		13	14	16	22			
Functional unit status				dest	S1	S2	FU for	FU for	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			No	No
Register result status										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
61	FU						Divide			

记分板示例 第62周期

<u>Instruction status</u>				<i>Read</i>		<i>Execu</i>		<i>Write</i>			
Instruction	<i>j</i>	<i>k</i>		<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>				
LD	F6	34+	R2	1	2	3	4				
LD	F2	45+	R3	5	6	7	8				
MUL	F0	F2	F4	6	9	19	20				
SUB	F8	F6	F2	7	9	11	12				
DIV	F10	F0	F6	8	21	61	62				
ADD	F6	F8	F2	13	14	16	22				
<u>Functional unit status</u>				<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>	
	<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
		Integer	No								
		Mult1	No								
		Mult2	No								
		Add	No								
		0 Divide	No								
<u>Register result status</u>											
Clock			<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	<i>...</i>	<i>F30</i>
62		<i>FU</i>									

CDC 6600 的记分板

- 6600记分板的局限性：
 - 没有前递/旁路(forwarding)硬件；
 - 指令调度局限于基本块内(指令窗口小)；
 - 功能部件少（结构冒险），特别是integer/load store部件；
 - 存在结构冒险，就暂停发射指令；
 - 等待到WAR冒险解决；
 - 防止WAW冒险；

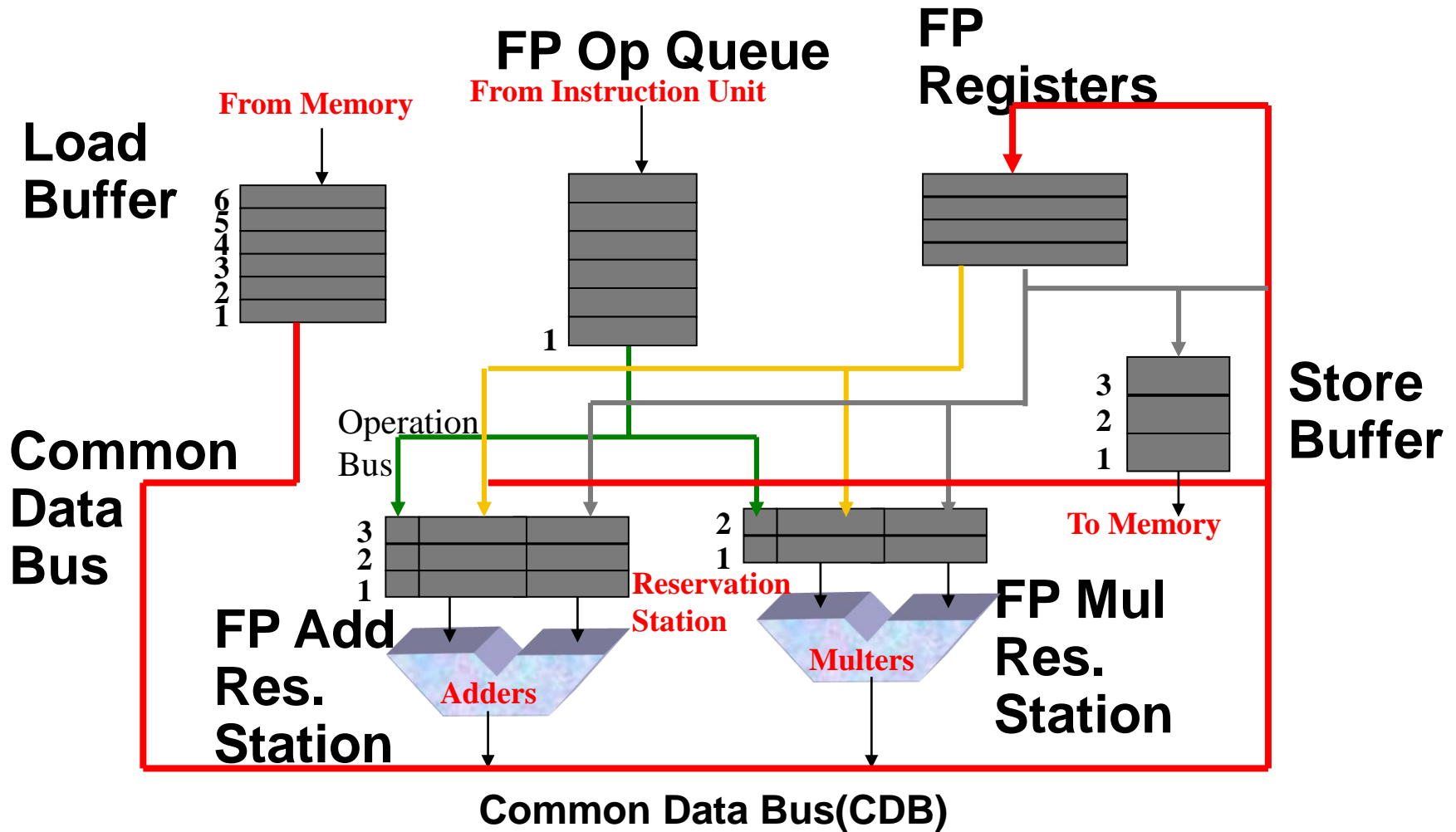
Tomasulo算法

- 为IBM 360/91设计的、在CDC 6600三年之后（1966）
- 目标：即使在没有特殊编译支持的情况下，也能取得高性能；
- IBM 360 和 CDC 6600指令系统体系结构之间的差异：
 - IBM360的每条指令有两个寄存器描述符(register specifiers)，而CDC 6600有三个；
 - IBM360只有四个浮点寄存器，而CDC 6600有八个。
- 为什么要学习Tomasulo算法？
 - 因为后续的Alpha 21264、HP 8000、MIPS 10000、Pentium II、PowerPC 604，…都有用到。

Tomasulo算法

- 控制&缓冲器**分布于功能部件** 与 **集中于记分板**；
 - 功能部件缓冲器称为 “保留站(reservation stations)”，用来存放未决的操作数；
- 指令中的寄存器被**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)**；
 - 消除了WAR、WAW冒险；
 - 保留站比实际寄存器多，因而可以完成编译器所不能完成的一些优化工作；
- 计算结果从RS直通FU，无需通过寄存器，而是通过公共数据总线（Common Data Bus）把结果广播到所有的FU；
- 装入（Load） 和 存储（Stores）也象其他功能部件一样具有保留站（专门的缓冲器）；

Tomasulo的架构图



Tomasulo算法的三个阶段

1. Issue—从FP Op Queue中取出指令

- 如果保留站空闲（**无结构冒险**），控制机制发射指令&发送操作数（**对寄存器进行换名，消除名相关**）。

2. Execution—对操作数执行操作(EX)

- 如果两个操作数都已就绪，就执行；（**RAW的处理**）
- 如果没有就绪，就观测公共数据总线等待所需结果；

3. Write result—完成执行(WB)

- 通过公共数据总线将结果写入到所有等待的部件；
- 标记保留站可用（Not busy）；

公共数据总线：数据 + 源（“来源”总线）

- 64位数据 + 4位功能部件源地址；
- 如果与期望的功能部件匹配，就“写”（产生结果）；
- 进行广播（broadcast）；

保留站的组成

- **Op**—该部件将完成的具体操作(例如, + or -)
- **Vj, Vk**—源操作数的实际数值
 - 存储缓冲器(Store buffers)设有V域, 存放将存储的结果;
- **Qj, Qk**—将产生源寄存器值(将写的值)的保留站编号
- **Busy**—指明保留站 或 FU 处于忙状态

注意:

- 没有记分板中的就绪(READY)标志;
 - 在tomasulo算法中 $Qj, Qk=0$ 就表示操作数处于ready状态。
-
- **Register result status**—指明哪个功能部件将写到哪个寄存器(Qi)。如果没有将写入寄存器的未决指令, 该域为空 ;

Tomasulo示例 第0周期

Instruction status				Execution Write							
Instruction	j	k	Issue complete	Result	Busy	Address					
LD F6	34+	R2			Load1	No					
LD F2	45+	R3			Load2	No					
MUL F0	F2	F4			Load3	No					
SUB F8	F6	F2									
DIV F10	F0	F6									
ADD F6	F8	F2									
Reservation Stations				S1	S2	RS for j	RS for k				
Time	Name	Busy	Op	Vj	Vk	Qj	Qk				
0	Add1	No									
0	Add2	No									
0	Add3	No									
0	Mult1	No									
0	Mult2	No									
Register result status											
Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU										

LD: 2 cycles

ADD: 2 cycles

Mult: 10 cycles

Divd: 40 cycles

Tomasulo示例 第1周期

Instruction status Execution Write

Instruction *j* *k* Issue complete Result

LD F6 34+ R2

1

LD F2 45+ R3

MUL F0 F2 F4

SUB F8 F6 F2

DIV F10 F0 F6

ADD F6 F8 F2

Busy Address

2 Load1 Yes 34+R2

0 Load2 No

0 Load3 No

Reservation Stations

S1

S2

RS for *j*

RS for *k*

Time Name Bus Op

V_{*j*}

V_{*k*}

Q_{*j*}

Q_{*k*}

0 Add1 No

0 Add2 No

Add3 No

0 Mult1 No

0 Mult2 No

Register result status

Clock

F0 F2

F4

F6

F8

F10 F12 ...

F30

1

FU

Load1

Tomasulo示例 第2周期

Instruction status Execution Write

Instruction *j* *k* Issue complete Result

LD	F6	34+	R2	1	
LD	F2	45+	R3	2	
MUL	F0	F2	F4		
SUB	F8	F6	F2		
DIV	F10	F0	F6		
ADD	F6	F8	F2		

	Busy	Address
1 Load1	Yes	34+R2
2 Load2	Yes	45+R3
0 Load3	No	

Reservation Stations

Time Name Bus Op S1 S2 RS for *j* RS for *k*
 V_j V_k Q_j Q_k

0 Add1 No
 0 Add2 No
 Add3 No
 0 Mult1 No
 0 Mult2 No

注意: 与CDC6600不同, 可以有多个 loads 被发射!

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
2	FU		Load2		Load1				

Tomasulo示例 第3周期

Instruction status				Execution Write			
Instruction	j	k		Issue	complete	Result	
LD F6	34+	R2		1	3		0 Load1
LD F2	45+	R3		2			1 Load2
MUL F0	F2	F4		3			0 Load3
SUB F8	F6	F2					
DIV F10	F0	F6					
ADD F6	F8	F2					

Busy Address

Yes	34+R2
Yes	45+R3
No	

<u>Reservation Stations</u>				<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>
<i>Time</i>	<i>Name</i>	<i>Bus</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD		R(F4)	Load2	
0	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1				

- 保留站中寄存器名被“换名”；MULT 可发射（与记分板比较）
- Load1完成，哪些指令在等待Load1的结果？

Tomasulo示例 第4周期

Instruction status				Execution Write				
Instruction	j	k		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	0 Load1	No
LD F2	45+	R3		2	4		0 Load2	Yes 45+R3
MUL F0	F2	F4		3			0 Load3	No
SUB F8	F6	F2		4				
DIV F10	F0	F6						
ADD F6	F8	F2						

Reservation Stations			S1	S2	RS for j	RS for k
Time	Name	Bus Op	Vj	Vk	Qj	Qk
0	Add1	Yes	SUBC	M(34+R2)		Load2
0	Add2	No				
	Add3	No				
0	Mult1	Yes	MULTD	R(F4)	Load2	
0	Mult2	No				

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
4	FU	Mult1	Load2		M(34+R2)	Add1			

- Load2将完成，哪些指令在等待Load2?

Tomasulo示例 第5周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3			Load3	No
SUB F8	F6	F2		4				
DIV F10	F0	F6		5				
ADD F6	F8	F2						

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus. Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
2	Add1	Yes	SUBC	M(34+R2)	M(45+R3)	
0	Add2	No				
	Add3	No				
10	Mult1	Yes	MULT	M(45+R3)	R(F4)	
0	Mult2	Yes	DIVD	M(34+R2)	Mult1	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
5	FU	Mult1	M(45+R3)		M(34+R2)	Add1	Mult2		

Tomasulo示例 第6周期

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Execution complete	Write Result
LD F6 34+ R2			1	3	4
LD F2 45+ R3			2	4	5
MUL F0 F2 F4			3		
SUB F8 F6 F2			4		
DIV F10 F0 F6			5		
ADD F6 F8 F2			6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

		<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>
<i>Time</i>	<i>Name</i>	<i>Bus</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>
				<i>Qj</i>	<i>Qk</i>

1	Add1	Yes	SUBD	M(34+R2)	M(45+R3)
0	Add2	Yes	ADDD		M(45+R3)
	Add3	No			
9	Mult1	Yes	MULT	M(45+R3)	R(F4)
0	Mult2	Yes	DIVD		M(34+R2)

• 发射ADDD

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
6	FU	Mult1	M(45+R3)		Add2	Add1	Mult2			

Tomasulo示例 第7周期

Instruction status				Execution Write				
Instruction	j	k		Issue	complete	Result	Busy	Address
LD F6 34+ R2				1	3	4	Load1	No
LD F2 45+ R3				2	4	5	Load2	No
MUL F0 F2 F4				3			Load3	No
SUB F8 F6 F2				4	7			
DIV F10 F0 F6				5				
ADD F6 F8 F2				6				

Reservation Stations			S1	S2	RS for j	RS for k
Time	Name	Bus Op	Vj	Vk	Qj	Qk
0	Add1	Yes	SUBD M(34+R2)	M(45+R3)		
0	Add2	Yes	ADDD	M(45+R3)	Add1	
	Add3	No				
8	Mult1	Yes	MULT M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD	M(34+R2)	Mult1	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(45+R3)		Add2	Add1	Mult2			

- Add1完成，哪些指令在等待Add1？

Tomasulo示例 第8周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3			Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5				
ADD F6	F8	F2		6				

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus	Op	V _j	V _k	Q _j Q _k
0	Add1	No				
2	Add2	Yes	ADD	M()-M()	M(45+R3)	
	Add3	No				
7	Mult1	Yes	MULT	M(45+R3)	R(F4)	
0	Mult2	Yes	DIV		M(34+R2)	Mult1

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12...	F30
8	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

Tomasulo示例 第9周期

Instruction	j	k	Issue complete	Result	Busy	Address
LD F6	34+	R2	1	3	4	Load1 No
LD F2	45+	R3	2	4	5	Load2 No
MULTD F2	F4		3			Load3 No
SUBB8 F6	F2		4	7	8	
DIVD F10	F0	F6	5			
ADDB6 F8	F2		6			

Reservation Stations	S1	S2	RS for j	RS for k
TimeNameBusyOp Vj Vk Qj Qk				
0 Add1	No			
1 Add2	Yes ADD	M()-M()	M(45+R3)	
Add3	No			
6 Mult1	Yes MULTD	M(45+R3)R(F4)		
0 Mult2	Yes DIVD	M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12...	F30
9	FU	Mult1 M(45+R3)		Add2	M()-M()	Mult2		

Tomasulo示例 第10周期

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result
LD F6	34+	R2	1	3	4
LD F2	45+	R3	2	4	5
MUL F0	F2	F4	3		
SUB F8	F6	F2	4	7	8
DIV F10	F0	F6	5		
ADD F6	F8	F2	6	10	

Busy Address

Load1	No
Load2	No
Load3	No

Reservation Stations

Time	Name	Bus	Op	<i>S1</i> <i>Vj</i>	<i>S2</i> <i>Vk</i>	<i>RS for j</i> <i>Qj</i>	<i>RS for k</i> <i>Qk</i>
0	Add1	No					
0	Add2	Yes	ADDC	M()-M()	M(45+R3)		
	Add3	No					
5	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
10	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

- Add2完成，哪些指令在等待Add2?

Tomasulo示例 第11周期

Instruction status

Instruction			<i>j</i>	<i>k</i>	<i>Execution Write</i>			Busy Address	
					<i>Issue</i>	<i>complete</i>	<i>Result</i>		
LD	F6	34+	R2		1	3	4	Load1	No
LD	F2	45+	R3		2	4	5	Load2	No
MUL	F0	F2	F4		3			Load3	No
SUB	F8	F6	F2		4	7	8		
DIV	F10	F0	F6		5				
ADD	F6	F8	F2		6	10	11		

Reservation Stations

			<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>
<i>Time</i>	<i>Name</i>	<i>Bus. Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
0	Add1	No				
0	Add2	No				
	Add3	No				
4	Mult1	Yes	MULT	M(45+R3)	R(F4)	
0	Mult2	Yes	DIVD	M(34+R2)	Mult1	

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12 ...</i>	<i>F30</i>
11	FU	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2		

- ADDD在该周期写结果

Tomasulo示例 第12周期

Instruction status				Execution Write				
Instruction	j	k		Issue	complete	Result	Busy	Address
LD F6 34+ R2				1	3	4	Load1	No
LD F2 45+ R3				2	4	5	Load2	No
MUL F0 F2 F4				3			Load3	No
SUB F8 F6 F2				4	6	7		
DIV F10 F0 F6				5				
ADD F6 F8 F2				6	10	11		

Reservation Stations				S1	S2	RS for j	RS for k
Time	Name	Bus	Op	Vj	Vk	Qj	Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
3	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status									
Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
12	FU	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2		

- 注意：所有短周期指令都已经完成

Tomasulo示例 第13周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3			Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5				
ADD F6	F8	F2		6	10	11		

Reservation Stations				S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
0	Add1	No					
0	Add2	No					
	Add3	No					
2	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12 ...	F30
13	FU	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2		

Tomasulo示例 第14周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3			Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5				
ADD F6	F8	F2		6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus	Op	V _j	V _k	Q _j Q _k
0	Add1	No				
0	Add2	No				
	Add3	No				
1	Mult1	Yes	MULT	M(45+R3)	R(F4)	
0	Mult2	Yes	DIVD		M(34+R2)	Mult1

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12...	F30
14	FU	Mult1 M(45+R3)			(M-M)+M() M()-M(Mult2				

Tomasulo示例 第15周期

Instruction status				Execution Write				
Instruction	j	k		Issue	complete	Result	Busy	Address
LD F6 34+ R2				1	3	4	Load1	No
LD F2 45+ R3				2	4	5	Load2	No
MUL F0 F2 F4				3	15		Load3	No
SUB F8 F6 F2				4	7	8		
DIV F10 F0 F6				5				
ADD F6 F8 F2				6	10	11		

Reservation Stations				S1	S2	RS for j	RS for k
Time	Name	Bus	Op	Vj	Vk	Qj	Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

- Mult1 completing; what is waiting for it?

Tomasulo示例 第16周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3	15	16	Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5				
ADD F6	F8	F2		6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
40	Mult2	Yes	DIVD	M*F4		M(34+R2)

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	M*F4	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

- 注意: 只有除法指令没有完成

Tomasulo示例 第55周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3	15	16	Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5				
ADD F6	F8	F2		6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Bus Op	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
1	Mult2	Yes	DIVD	M*F4		M(34+R2)

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU	M*F4	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

Tomasulo示例 第56周期

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3	15	16	Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5	56			
ADD F6	F8	F2		6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Busy	Op	V _j	V _k	Q _j Q _k
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
0	Mult2	Yes	DIVD	M*F4	M(34+R2)	

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

- Mult2 完成

Tomasulo示例 第57周期

Instruction status				Execution Write				
Instruction	j	k		Issue	complete	Result	Busy	Address
LD F6	34+	R2		1	3	4	Load1	No
LD F2	45+	R3		2	4	5	Load2	No
MUL F0	F2	F4		3	15	16	Load3	No
SUB F8	F6	F2		4	7	8		
DIV F10	F0	F6		5	56	57		
ADD F6	F8	F2		6	10	11		

Reservation Stations			S1	S2	RS for j	RS for k
Time	Name	Bus Op	Vj	Vk	Qj	Qk
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
0	Mult2	No				

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU	M*F4	M(45+R3)		(M-M)+M()	M()-M()	M*F4/M			

- 在基于tomasulo算法的动态调度中, 指令是: 按序发送、乱序执行、乱序完成的。

与记分板的第62周期相比

Instruction status				Read Execu Write			
Instruction	j	k		Issue	operat	compl	Result
LD F6 34+ R2				1	2	3	4
LD F2 45+ R3				5	6	7	8
MUL F0 F2 F4				6	9	19	20
SUB F8 F6 F2				7	9	11	12
DIV F10 F0 F6				8	21	61	62
ADD F6 F8 F2				13	14	16	22

Functional unit status			dest	S1	S2	FU for	FU for	Fj?	Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	0 Divide	No								

Register result status			F0	F2	F4	F6	F8	F10	F12	...	F30
Clock											
62	FU										

- CDC6600的记分板为什么需要更长的时间？

Tomasulo 与 记分板

流水化的功能部件

(6 load, 3 store, 3 +, 2 x/÷)

窗口大小: ≤ 14 指令

结构冒险暂停发射

WAR: 通过换名避免

WAW: 通过换名避免

从FU广播结果

控制: 分布式的保留站

多功能部件

(1 load/store, 1 +, 2 x, 1 ÷)

≤ 5 指令

相同

暂停完成

暂停发射

写/读 寄存器

集中控制的记分板

Tomasulo算法的缺点

- 功能实现复杂
 - 需要复杂的硬件实现相应的功能
- 需要大量高速的相联存储 (associative buffer)
- 公共数据总线是制约性能增长的瓶颈
 - 每个CDB必须广播到多个功能部件单元 \Rightarrow 大容量、写操作密集；
 - 每个周期可以同时完成的功能部件数量可能由于单总线而受限（最坏情况为“1”）！
 - 多个CDB \Rightarrow 为完成并行相联存储，功能部件FU需要更复杂的逻辑控制；

内容概要

- 指令级并行的概念
- 指令的动态调度
- 动态分支预测技术
 - ① BHT/BPB
 - ② BTB
 - ③ 前瞻执行
- 多指令流出技术
- 循环展开和指令调度

分支预测技术的重要性

- 所开发的ILP越多或流水越深，控制相关的制约就越大，分支预测就要有更高的准确度。
- 本节中介绍的方法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。
 - 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；
 - Amdahl定律告诉我们，机器的CPI越小，控制停顿的相对影响就更大。

动态分支预测技术

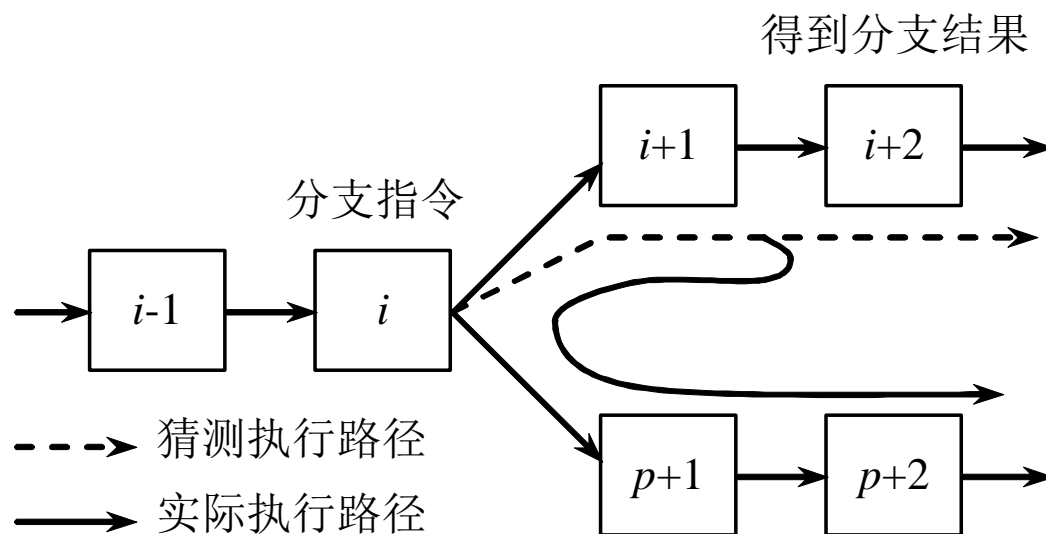
- 动态分支预测：在程序运行时，根据分支指令过去的表现来预测其将来的行为。
 - 如果分支行为发生了变化，预测结果也跟着改变；
 - 有更好的预测准确度和适应性；
- 分支预测的有效性取决于：
 - 预测算法的准确率；
 - 预测正确和不正确两种情况下的分支开销；
- 决定分支开销的因素：
 - 流水线的结构；
 - 预测的方法；
 - 预测错误时的恢复策略，等；

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有两个：
 - 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？

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。

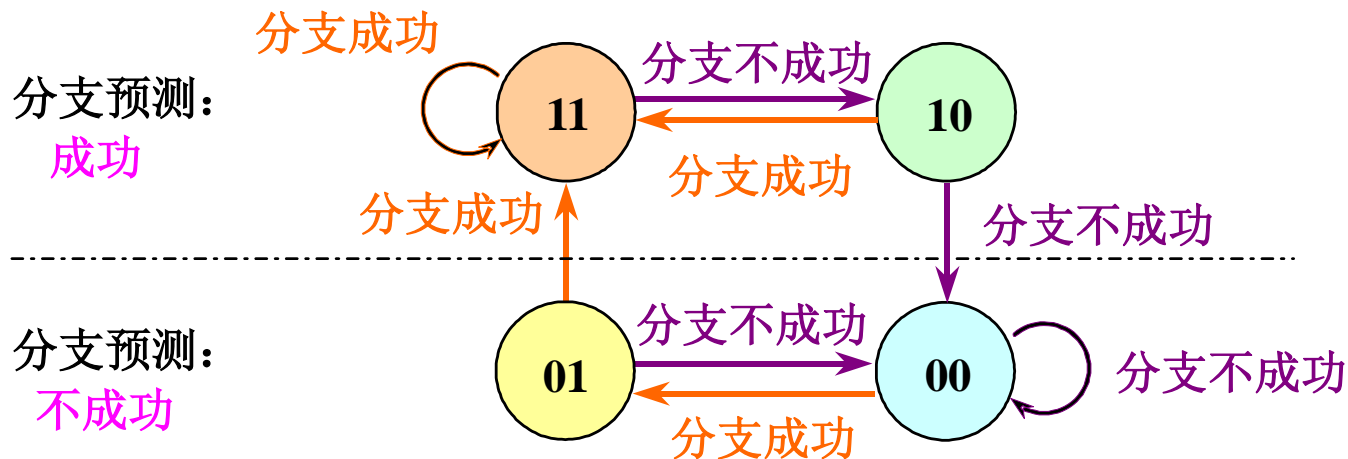


分支历史表

- 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;
- 只有1个预测位的分支预测缓冲
 - 记录分支指令最近一次的历史, BHT中只需要1位二进制位。(最简单)
 - 缺点: 预测的精度不高

分支历史表

- 采用两位二进制位来记录历史
 - 提高预测的准确度；
 - 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；
- 两位分支预测的状态转换如下所示：



分支历史表

- 两位分支预测有两个步骤：
 - Step 1: 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。
 - Step 2: 状态更新，修改BHT的状态。
- BHT方法只在以下情况下才有用：
 - 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。
 - 因为BHT对分支目标地址没提供支持，或者说在BHT下依然要计算分支目标。

分支历史表

- 课后思考：BHT方法对MIPS 5段流水线是否有好处？
- 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%。
 - 一般来说，采用4K的BHT就可以了。
- BHT可以跟分支指令一起存放在指令Cache中，也可以用专门的硬件来实现。

高级的分支预测技术

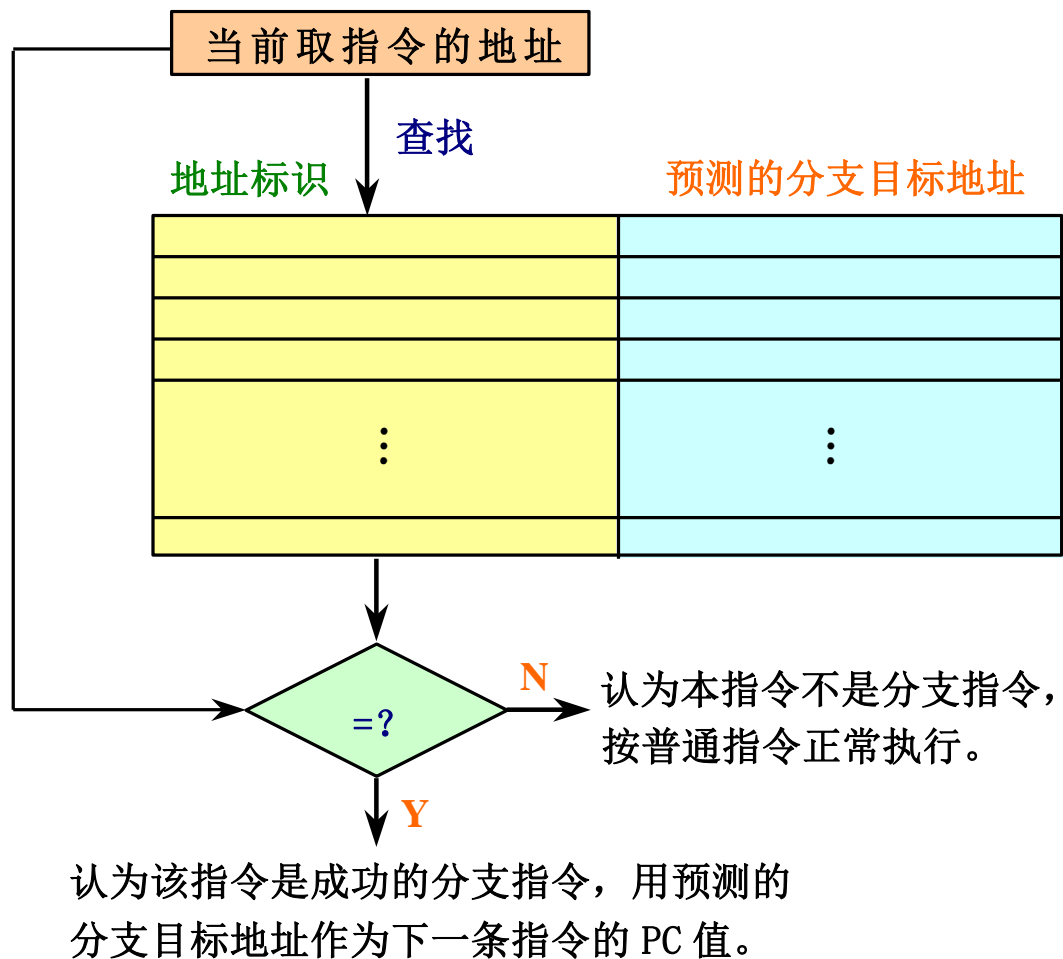
- Correlation Branch Prediction
- Tournament Branch Prediction
- Bi-mode Branch Prediction
- Neuron based Branch Prediction
- Bias-Free Branch Prediction
-

Check them in [ACM Digital Library](#) or [IEEE xlore](#)

分支目标缓冲器

- 目标：将分支的开销降为 0
- 方法：利用分支目标缓冲器
 - 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）。

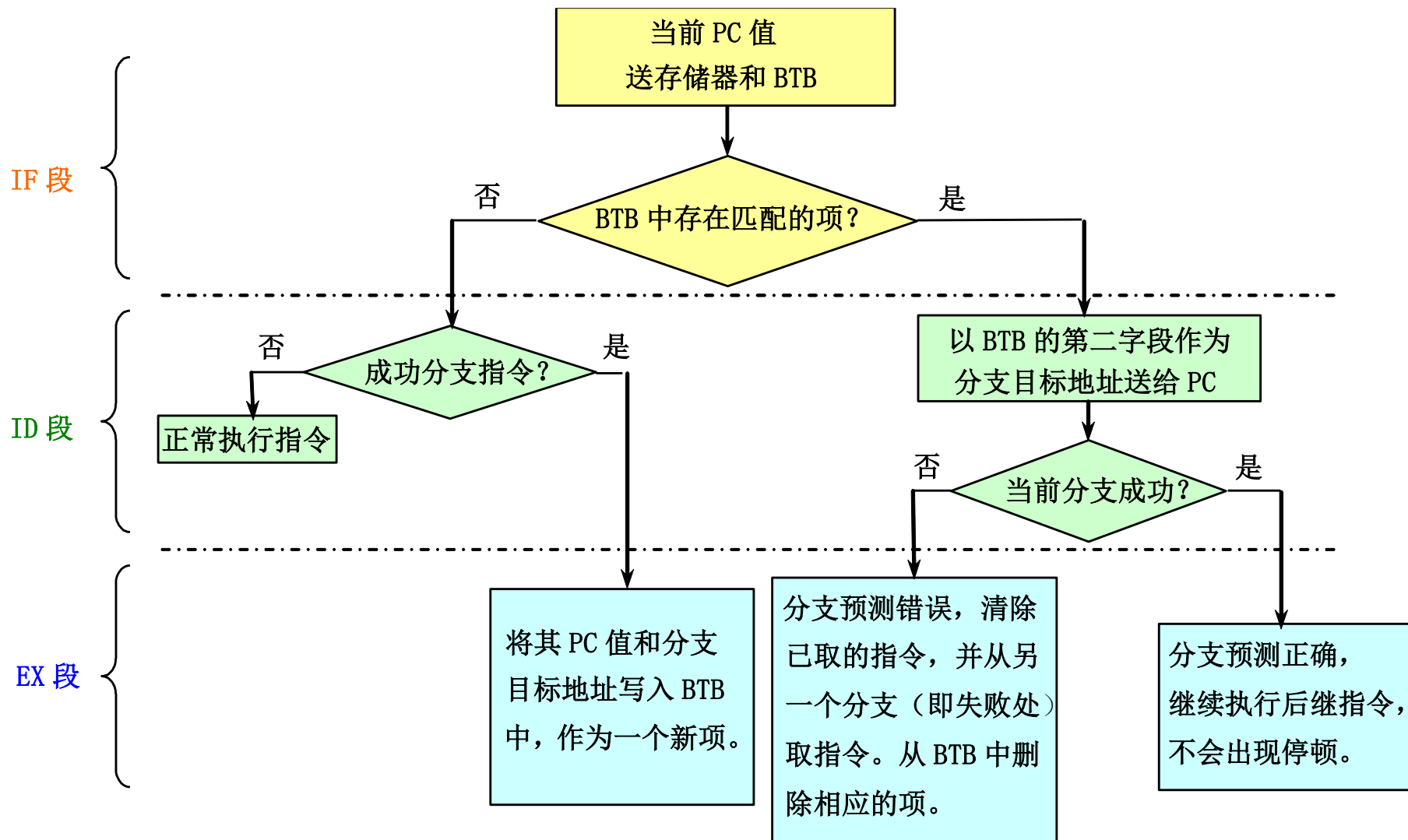
分支目标缓冲器的结构图



分支目标缓冲器

- BTB可以用专门的硬件实现的一张表格；
- 表格中的每一项通常有两个字段：
 - 执行过的成功分支指令的地址；
 - 作为该表的匹配标识 (key)
 - 预测的分支目标地址；

支持BTB的流水线操作



前瞻(推测)执行

- 前瞻执行 (speculative execution) 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。
 - 指令执行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。

为什么写到ROB之中？

前瞻(推测)执行

- 基于硬件的前瞻执行结合了三种技术：
 - ① 动态分支预测，用来选择后续执行的指令；
 - ② 在控制相关的结果尚未出来之前，前瞻地执行后续指令；
 - ③ 用动态调度对基本块的各种组合进行跨块调度；
- 对Tomasulo算法加以扩充，就可以支持前瞻执行：
 - 把Tomasulo算法的最后一个阶段中写结果和指令完成加以区分，分成两个不同的段：
 - ① 写结果；
 - ② 指令确认；

前瞻(推测)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。

实现前瞻的关键思想：

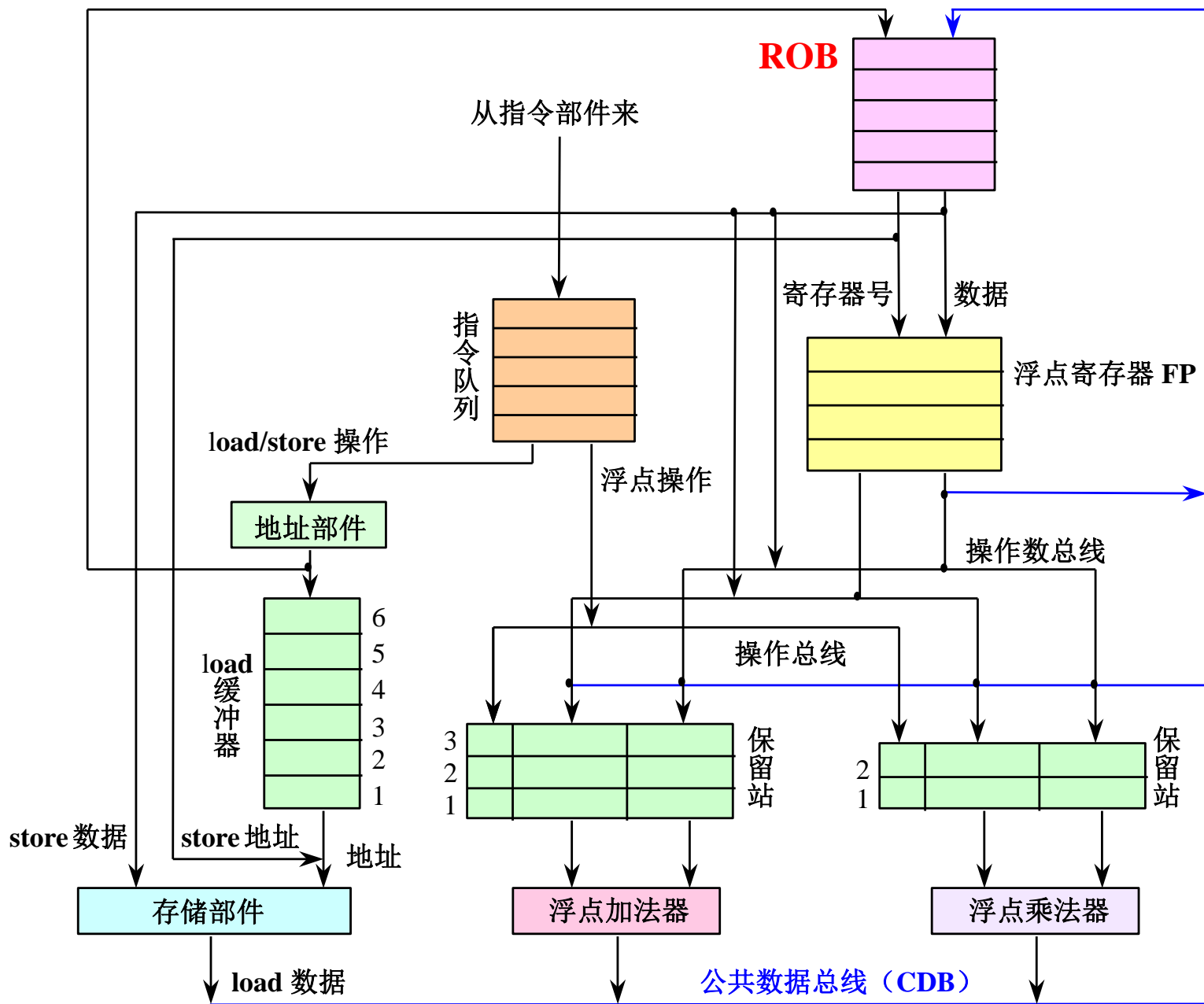
允许指令乱序执行，但必须
顺序确认。

写

以确认，

并... 顺序执行；

支持前瞻执行的浮点结构



前瞻(推测)执行

- ROB中的每一项由以下4个字段组成：
 - ① **指令类型**：指出该指令是分支指令、store指令或寄存器操作指令；
 - ② **目标地址**：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）；
 - ③ **数据值字段**：用来保存指令前瞻执行的结果，直到指令得到确认；
 - ④ **就绪字段**：指出指令是否已经完成执行并且数据已就绪；

前瞻(推测)执行

- 在前瞻执行下，Tomasulo算法中保留站(RS)的换名功能是由ROB来完成的。
- 采用前瞻执行机制后，指令的执行步骤：

① 流出

- 从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。（**结构冲突**）

前瞻(推测)执行

② 执行

- 当两个操作数都就绪后，就可以执行该指令的操作；
- 如果有操作数尚未就绪，就等待，并不断地监测CDB。
(RAW冲突)

③ 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。
- 此时，释放产生该结果的保留站；
- store指令在写结果阶段完成，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；

前瞻(推测)执行

④ 确认：不同指令的处理不同：

- 其他指令（除分支指令和store指令之外）：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；

前瞻(推测)执行

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即将确认时的各个状态表内容。

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

前瞻(推测)执行

- 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	

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 F5 (R3)	确认	F2	Mem[45+Regs[R3]]
3	yes	MUL.D F0, F2, F4	写回	F0	F2×Regs[F4]
4	yes	SUB			
5	yes				
6	yes				


请同学们自己对照
Tomasulo调度算法和推
测执行的过程，自己推
导执行过程。

字段	写回状态							
	F0	F2	F4	F6	F8	F10	...	F30
ROB项编号	3			6	4	5		
Busy	yes	no	no	yes	yes	yes	...	no

前瞻(推测)执行

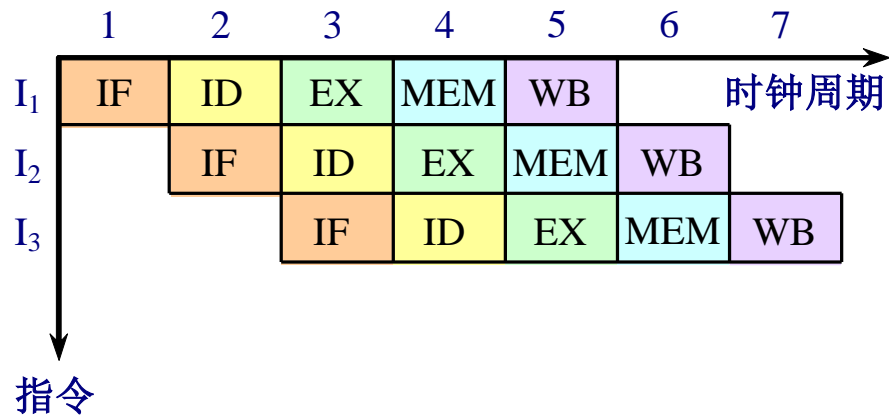
- 前瞻执行的优点
 - 通过ROB实现了指令的顺序完成;
 - 能够实现精确异常;
 - 很容易地推广到整数寄存器和整数功能单元上;
- 前瞻执行的缺点:
 - 复杂的控制导致所需的硬件太复杂;

本章内容概要

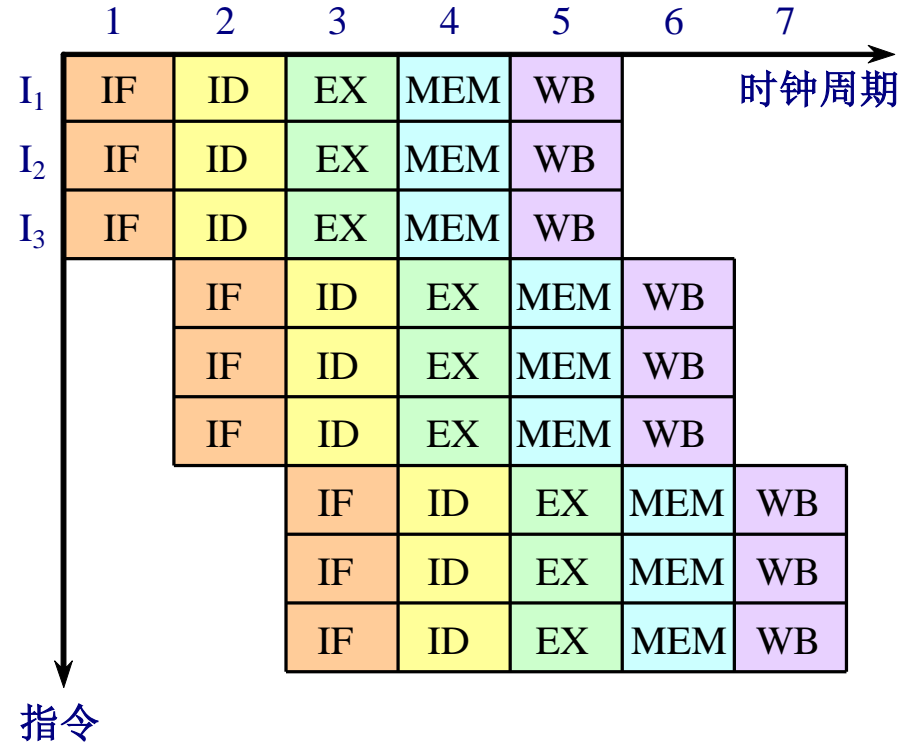
- 指令级并行的概念
 - 指令的动态调度
 - 动态分支预测技术
 - **多指令流出技术**
 - 循环展开和指令调度
- 
- ① 静态调度超标量
 - ② 动态调度超标量
 - ③ VLIW & EPIC

多指令流出技术

单流出时空图



多流出时空图



单流出和多流出处理器执行指令的时空图

多指令流出技术

- 多流出处理机有两种基本风格：
 - 超标量（Superscalar）〈可以再分为二类〉
 - 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 n ，就称该处理机为 n 流出；
 - 可以基于编译器的静态调度，也可以基于Tomasulo算法进行动态调度；
 - VLIW/EPIC
 - 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

- 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，当然运行的效果通常不会很好。

使用多指令流出技术的处理器示例

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 UltraSPARC II / III
超标量 (动态)	动态	硬件	动态	部分乱序执行	IBM Power2
超标量 (猜测)	动态	硬件	带有猜 测的动 态执行	带有猜测的 乱序执行	Pentium III/4, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64 III
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 (IA-64)

静态多指令流出技术

- 基于静态调度的多流出技术
 - 在典型的超标量处理器中，每个时钟周期可流出1到8条指令。
 - 指令按序流出，在流出时进行冲突检测。
 - 保证在当前流出的指令序列中不存在数据冲突。

静态多指令流出技术

- 例：一个4流出的静态调度超标量处理机
 - 在取指令阶段，流水线将从取指令部件收到1~4条指令（称为流出包）。
 - 在一个时钟周期内，这些指令有可能是全部都能流出，也可能是只有一部分能流出。
 - 流出部件检测结构冲突或者数据冲突。一般分两阶段实现：
 - ① 进行流出包内的冲突检测，选出初步判定可以流出的指令。
 - ② 步骤①所选出的指令与正在执行的指令的冲突检测。

静态多指令流出技术

- MIPS处理机是怎样实现超标量的呢？
 - 假设：每个时钟周期流出两条指令：1条整型指令+1条浮点指令。其中，把load指令、store指令、分支指令归类为整数型指令。
 - 要求：同时取两条指令（64位），译码两条指令（64位）；

静态多指令流出技术

- 解答：
 - 对指令的处理步骤如下：
 - 从指令缓存中取两条指令；
 - 确定几条指令可以流出（0~2条指令）；
 - 把它们发送到相应的功能部件；
 - 理想情况下，指令的执行过程：
 - 假设：所有的浮点指令都是加法指令，其执行时间为两个时钟周期；
 - 为简单，下图总把整数指令放在浮点指令的前面；

静态多指令流出技术

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

静态多指令流出技术

- 采用“1条整数型指令+1条浮点指令”并行流出的方式，需要增加的硬件很少。
 - 需要增加冲突检测的硬件机制；
 - 浮点load或浮点store指令需要使用整数部件，还会增加对浮点寄存器的访问冲突。可以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。

静态多指令流出技术

- 限制超标量流水线的性能发挥的障碍。
 - load指令
 - 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；
 - 如果分支指令是流出包中的第二条指令，其延迟就是2条指令；

动态多指令流出技术

- 基于动态调度的多流出技术
 - 扩展Tomasulo算法：支持两路超标量
 - 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。
- 一种比较简单的实现方法：
 - 指令按顺序流向保留站。
 - 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。

动态多指令流出技术

- 有两种不同的方法可以实现多流出。关键在于：
对保留站的分配和对流水线控制表格的修改；
 - ① 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；
 - ② 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。

动态多指令流出技术

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，表示尚未结束，转移到Loop继续

动态多指令流出技术

- 现做以下假设：
 - 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1个时钟周期。
 - 具有动态分支预测部件和1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。
 - 产生结果的延迟为：整数运算1个周期，load指令2个周期，浮点加法运算3个周期。
- 请列出该程序前面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。

解：执行时，该循环将动态展开后的执行过程如下：

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

动态多指令流出技术

- 从图中可以看出：
 - 程序基本可以达到3拍流出5条指令
 - $IPC = 5/3 = 1.67$ 条/拍
 - 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。
 - 可增加一个加法器，把ALU功能和地址运算功能分开。

动态多指令流出技术

- 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2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。

超长指令字技术VLIW/EPIC

- 把能并行执行的多条指令组装成一条很长的指令；
 - 100多位到几百位
- 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；
- 在VLIW处理机中，所有的处理和指令安排都是由编译器完成的；

超长指令字技术VLIW/EPIC

例： 假设VLIW处理机每个时钟周期可同时流出5条指令：两条访存指令、两条浮点操作指令和一条整数指令或分支指令。对于例前面例子中的循环展开后的代码，给出它在该VLIW中的代码序列。不考虑分支指令的延迟槽。

- **解答：** 代码序列如下图所示。
 - 运行时间为8个时钟周期。
 - 每遍循环平均1.6个时钟周期。
 - 8个时钟周期内流出了17条指令，每个时钟周期2.1条。
 - 8个时钟周期共有操作槽 $8 \times 5 = 40$ 个，有效槽的比例为42.5%。

超长指令字技术VLIW/EPIC

访存指令1	访存指令2	浮点指令1	浮点指令2	整数/转移指令
L. D F0, 0(R1)	L. D F6, -8(R1)			
L. D F10, -16(R1)	L. D F14, -24(R1)			
L. D F18, -32(R1)		ADD. D F4, F0, F2	ADD. DF8, F6, F2	
		ADD. DF12, F10, F2	ADD. DF16, F14, F2	
		ADD. DF20, F18, F2		
S. D F4, 0(R1)	S. D F8, -8(R1)			
S. D F12, -16(R1)	S. D F16, - 24(R1)			DADDIUR1, R1, #-40
S. D F20, 8(R1)				BNE R1, R2, Loop

超长指令字技术VLIW/EPIC

- 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；
 - 解决：采用指令共享立即数字段的方法，或者采用指令压缩存储、调入Cache或译码时展开的方法。
 - 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

Next Topic

Memory Hierarchy