计算机系统结构
Computer Architecture

Ch.04 – Instruction Level Parallelism
指令级并行

Dr. Borui Li
李博睿 博士



2026年秋季·东南大学

Overview



- **本章回答的问题**
  - Q1 (Why): 为什么要学习指令级并行?
  - Q2 (What): 什么是指令级并行?
  - Q3 (How): 如何开发指令级并行?

- **本章总体要求本次课主要内容**
  - **掌握**动态执行技术，**理解**指令级并行技术原理



Part I. 为什么要学习指令级并行?

再从CPU性能公式讲起



- **CPU性能公式**

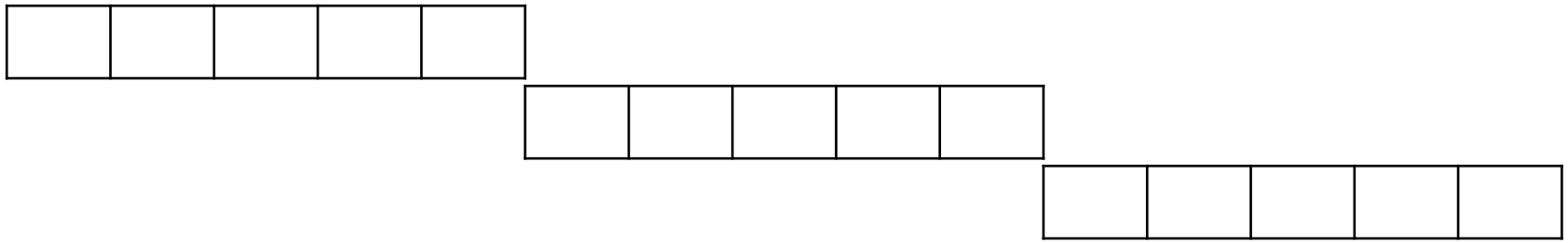
平均指令周期

$$\text{程序CPU时间} = \text{IC} \times \text{CPI} \times \text{时钟周期时间}$$

指令条数

- **使用流水线技术优化CPI**
  - $\text{CPI}_{\text{opt}} \rightarrow 1$
  - 但是CPI在流水线中无法完全到1

- **是否能够进一步优化CPI?**





Part I. 为什么要学习指令级并行?

Part II. 什么是指令级并行?

指令级并行与线程级并行



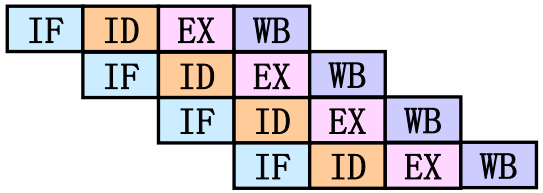
- **指令级并行（Instruction Level Parallelism，ILP）**
  - 当指令之间不存在相关时，它们在流水线中可以重叠起来并行执行
  - 充分利用指令间存在的并行性，以提高计算机性能

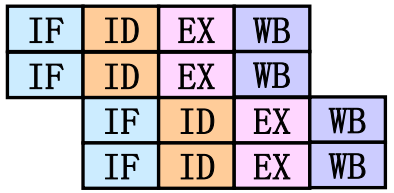
- **线程级并行（Thread Level Parallelism，TLP）**
  - 线程是CPU调度的基本单位
  - 线程切换只需保存和设置少量寄存器的内容，最快可以每周期都切换
  - 在ILP开发的基础上进一步将TLP与ILP同时开发?

指令级并行相关概念

- **并行度表示**
  - CPI (Cycle Per Instructions)→IPC (Instructions Per Cycle)，有CPI×IPC = 1
- **开发途径**
  - 时间重叠(指令流水线)、资源重复(指令同时执行)

    └→IPC = 1、$CPI_{理想} = 1$　　└→IPC > 1、$CPI_{理想} < 1$





- **性能：** $CPI_{实际} = CPI_{理想} + 停顿_{结构/数据/控制冒险}$
- **影响因素：** 数据相关(正相关/反相关/名相关)、控制相关

本章研究的技术



- **性能：** $\mathrm{CPI}_{\text{实际}} = \mathrm{CPI}_{\text{理想}} + \text{停顿}_{\text{结构/数据/控制冒险}}$

并行度	技术	提高性能的方面
指令级并行	指令动态调度 （记分牌和Tomasulo算法）	各种数据相关停顿
	动态分支预测	控制相关停顿
	前瞻执行 （Speculation）	所有数据/控制相关停顿
	多指令流出 （超标量和超长指令字）	提高理想CPI
线程级并行	多线程技术	所有数据/控制相关停顿



Part I. 为什么要学习指令级并行？

Part II. 什么是指令级并行？

Part III. 如何开发指令级并行？—数据相关动态调度

基本思想→记分牌算法（1964）→Tomasulo算法（1967）



Part I. 为什么要学习指令级并行？

Part II. 什么是指令级并行？

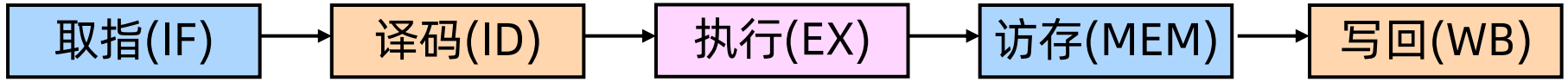
Part III. 如何开发指令级并行？—数据相关动态调度

基本思想→记分牌算法（1964）→Tomasulo算法（1967）

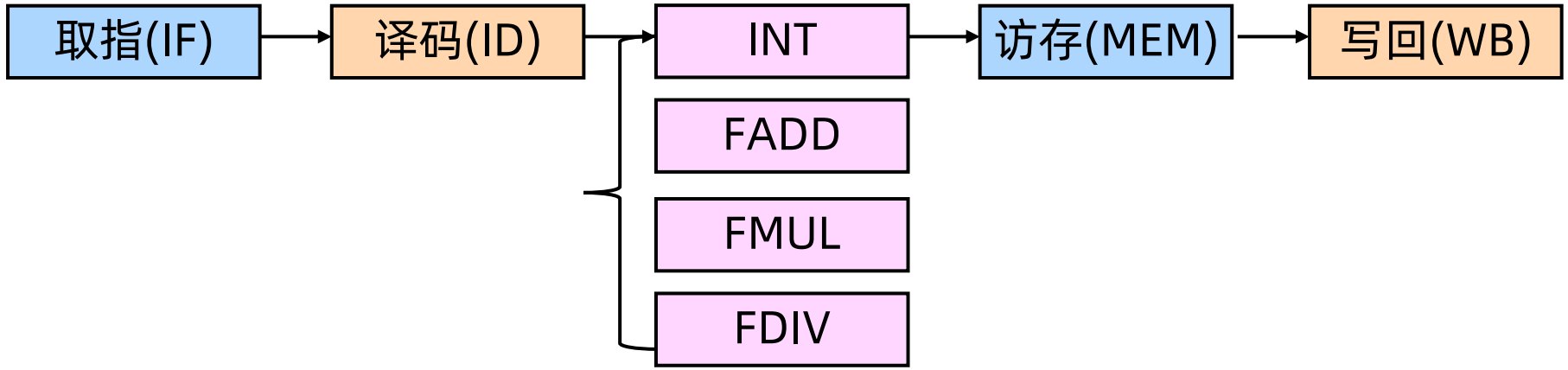
理想五级流水线→实际硬件设计

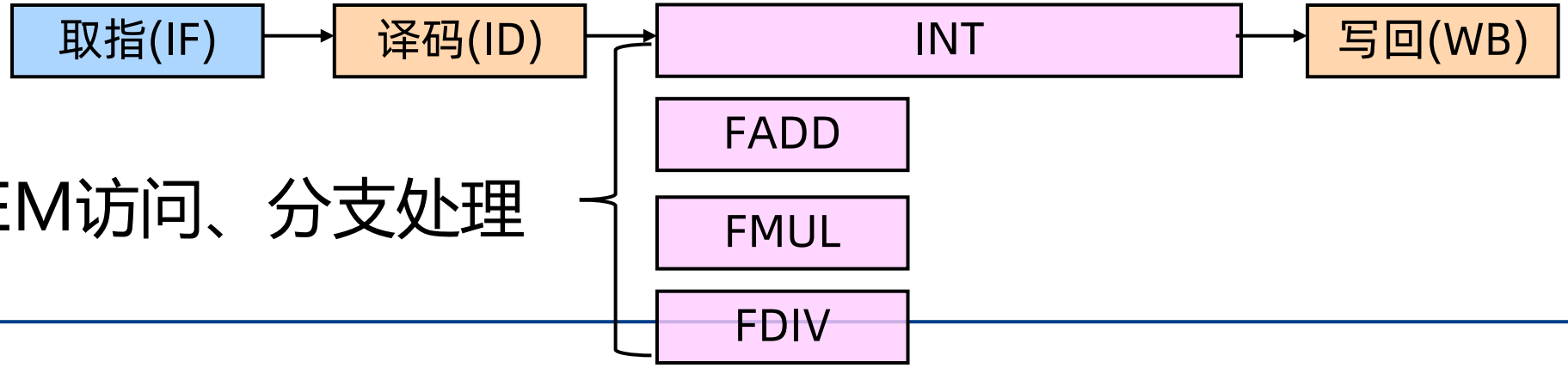


- **理想五级流水线：IF-ID-EX-MEM-WB**



- **执行器常有多个**
  - 各自周期不同

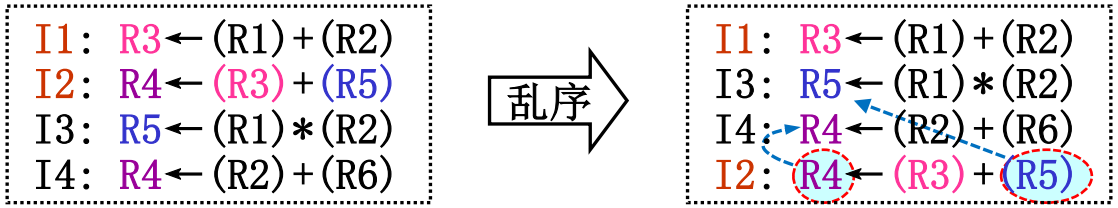




- **用整数部件实现**
  - 整数操作、MEM访问、分支处理

动态调度解决数据相关的基本思想

RAW(写后读)，新增WAR(读后写)、WAW(写后写)

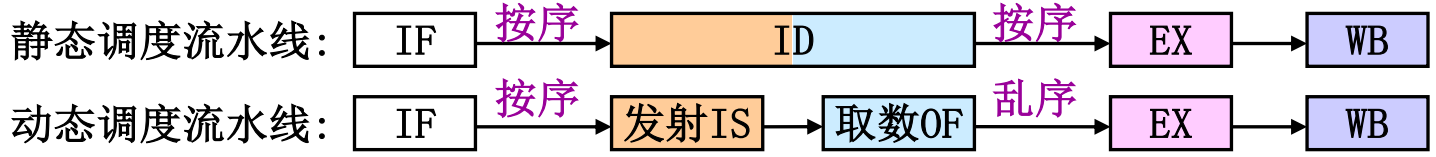


- **基本思想**：执行时进行指令重排序
  - ①ID段分为发射段(Issue,IS)、读操作数段(Operand Fetch,OF)

在发射段处理结构冒险，在读操作数段处理数据冒险

  - ②操作数就绪的指令优先进入执行段　←停顿最小化

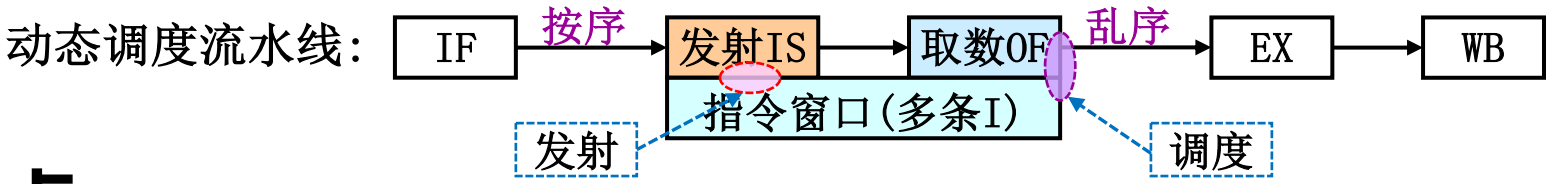
教材翻译为流出



- **可能的数据冒险：RAW、WAR、WAW**

动态调度解决数据相关的基本思想(Cont'd)

- **流水线结构**
  - ①设置指令窗口，IS段按序保存指令 ←提供乱序选择平台
  - ②OF段可乱序流出指令(操作数就绪时) ←消除/减少RAW停顿
  - ③IS&OF段避免WAR/WAW冒险 ←处理乱序遗留问题



- **流水线特点**
  - ①允许多条指令处于EX段(均含多个部件) ←静态调度为1条
    - **└→可提高性能←┴←指令乱序进入**
  - ②会发生不精确异常(个数不改变) ←静态调度亦为不精确异常
    - **└←不同于指令序列(EX段乱序所致)，需处理**
- **常见调度算法：记分牌算法 (早期) 、 Tomasulo算法 (主流)**



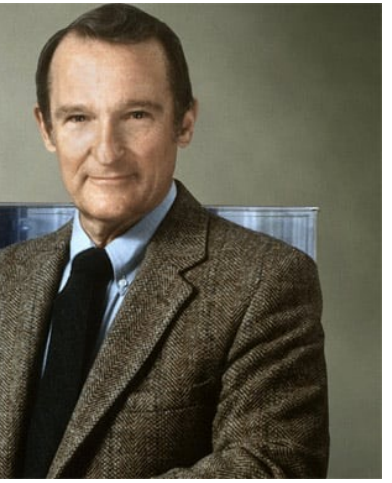
Part I. 为什么要学习指令级并行？

Part II. 什么是指令级并行？

Part III. 如何开发指令级并行？—数据相关动态调度

基本思想→**记分牌算法（1964）**→Tomasulo算法（1967）

记分牌(Score Board)算法



- **记分牌1964年被Seymour Cray用于CDC 660**
  - "Thomas Edison" for supercomputers
- **记分牌允许指令乱序执行，前提是**
  - 充足的资源(计算部件EX)，无数据相关
  - 记分牌动态解决了RAW相关
  - 指令可以乱序执行
- **基本原理**
  - 每条指令均经过记分牌，记录各指令间数据相关的信息
  - 如果记分牌判断出一条指令不能立即执行，就放入指令窗口
    - 后续检测硬件的变化从而决定何时能够执行

https://www.theregister.com/2015/10/02/seymour_cray_90_anniversary/?page=3

CDC 6600

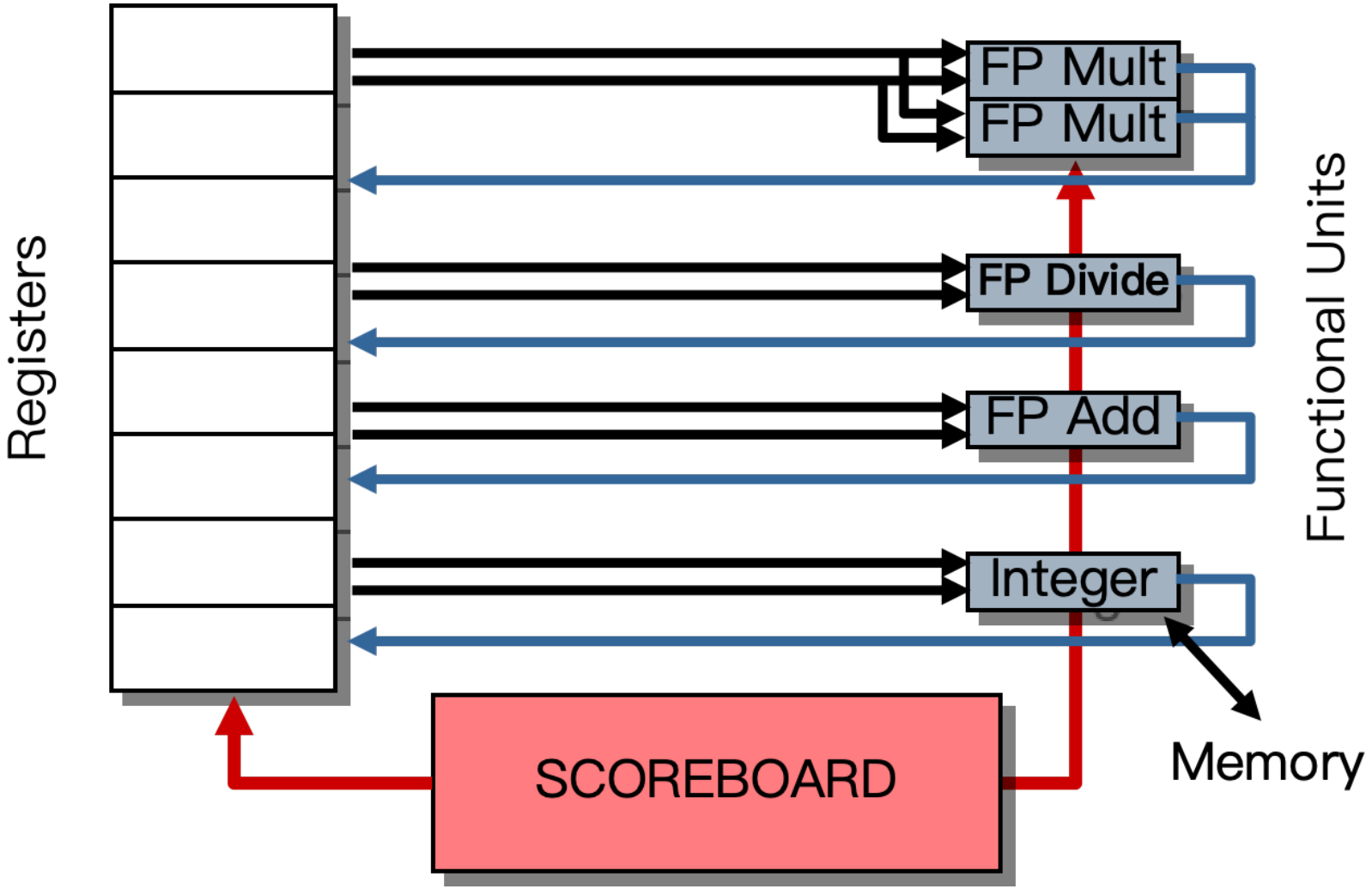
- 150台CDC6600在欧美顶级实验室效力于高能核物理研究。
- 整个60年代，CDC（控制数据公司）几乎独霸巨型计算机的市场







记分牌结构


Registers
FP Mult
FP Mult
FP Divide
FP Add
Integer
Functional Units
SCOREBOARD
Memory

记分牌执行过程



- **功能部件状态表**
- **用9个域来记录每个功能部件的状态**
  - Busy：指示功能部件是否工作
  - Op：功能部件当前执行的操作
  - F_i：目的寄存器编号
  - F_j，F_k：源寄存器编号
  - Q_j，Q_k：向R_j，R_k中写结果的功能部件
  - R_j，R_k：标示F_j，F_k是否就绪，是否已经被使用

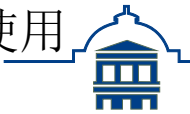
记分牌算法案例



- **指令序列**
  1. LD F6,34(R2)
  2. LD F2,45(R3)
  3. MULTD F0,F2,F4
  4. SUBD F8,F6,F2
  5. DIVD F10,F0,F6
  6. ADDD F6,F8,F2
- **浮点部件执行时间延迟**
  - 加（减）法2拍
  - 乘法 10 拍
  - 除法 40 拍
- **RAW相关**
  - I3/I4-I2
  - I5-I3
  - I6-I4
- **WAR相关**
  - I6-I5
- **WAW相关**
  - I6-I1

记分牌实例

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

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

Register result status:

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

Cycle 1

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock 1

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
FU				Integer					

流出LD，写Integer部件记分牌，写寄存器状态表

Cycle 2

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用

Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

第2条LD指令是否能够流出？

- 不能,结构冒险,INT被占用

第3条MULTD指令呢？

- 不能,需要顺序流出

Functional unit status:

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

LD完成OF，拿到了R2中操作数

Cycle 3

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用

Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

为什么要在这里修改Rk?

- 尽快释放寄存器，防止WAR阻塞

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

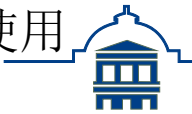
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*	Integer								

LD地址计算完成（1拍），释放R2（已经进入计算）

Cycle 4

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

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

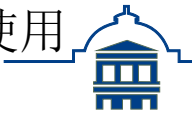
Register result status:

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

LD前半拍从内存取数，后半拍结果写回寄存器，释放整数部件

Cycle 5

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

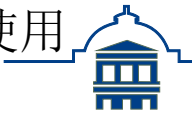
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*		Integer							

流出第二条LD，R3数据有效（Rk=yes）

Cycle 6

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

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

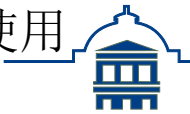
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	*FU*	Multi1	Integer							

LD完成R3读取；第三条MULTD指令流出，占用Mult1部件，但是源OPD F2未就绪

Cycle 7

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

是否可以读出MULTD操作数？

- 不能,要等F2写回

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Multd	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	Multi1	Integer			Add				

LD进入地址计算，Rk置为空闲，并完成计算；SUBD指令流出，占用Add部件，但是源OPD F2未就绪

Cycle 8 (前半)



Instruction status:

Instruction		j	k	Issue	Oper Fetch	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

是否读MULTD和SUB的操作数？

- 不能,要等F2写回

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Multd	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

Register result status:

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

DIVD指令流出，占用Divide部件，但是源OPD F0未就绪，是乘法器Mult1的结果；LD前半拍去内存读数

Cycle 8 (后半)



Instruction status:

Instruction		j	k	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	Yes	Multd	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

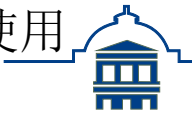
Register result status:

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

LD后半拍拿到数，放入F2；原本依赖F2的两条指令数据均就绪，可以启动执行

Cycle 9

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

是否流出ADDD?

- 不能,部件被SUBD占用

Functional unit status:

记录剩余时间→

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
10	Mult1	Yes	Multd	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Subd	F8	F6	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

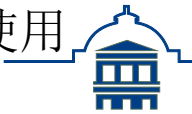
Register result status:

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

更新计数器，两条长指令开始进行计算

Cycle 10

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
9	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Subd	F8	F6	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

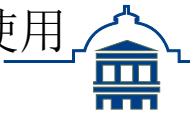
Register result status:

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

MULTD与SUBD执行1拍

Cycle 11

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
8	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Subd	F8	F6	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

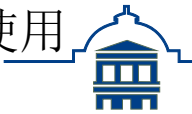
Register result status:

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

MULTD与SUBD执行1拍，SUBD的执行本周期就结束

Cycle 12

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
7	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

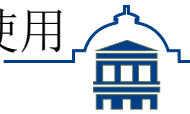
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	*FU*	Mult1					Divide			

SUBD写结果到F8，指令结束，清空功能部件状态行

Cycle 13

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
6	Mult1	Yes	Multd	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Addd	F6	F8	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

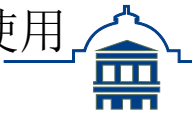
Register result status:

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

Add空闲，流出ADDD指令

Cycle 14

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
5	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Addd	F6	F8	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

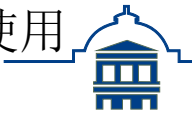
Register result status:

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

MULTD与ADDD执行1拍

Cycle 15

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
4	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

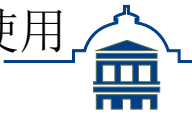
Register result status:

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

MULTD与ADDD执行1拍

Cycle 16

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
3	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

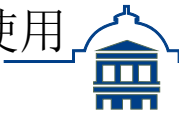
Register result status:

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

MULTD与ADDD执行1拍，ADDD执行完毕

Cycle 17

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

为什么不写ADDD的结果？

- DIVD的Rk是Yes，说明数据还没用
- F6存在WAR冒险！

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
2	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

Register result status:

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

ADDD执行完毕，扫描所有源寄存器阵列，准备写回F6

Cycle 18

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用

Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
1	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

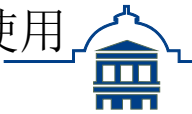
Register result status:

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

MULTD继续执行，其他两条指令阻塞

Cycle 19

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
0	Mult1	Yes	Multd	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

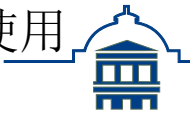
Register result status:

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

MULTD本周期执行完毕

Cycle 20

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Addd	F6	F8	F2			No	No
	Divide	Yes	Divd	F10	F0	F6			Yes	Yes

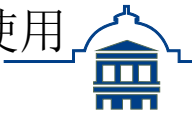
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
20	*FU*				Add		Divide			

MULTD写结果到F0；同时DIVD的阻塞消除，下拍开始取数

Cycle 21

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

WAR冒险已经消除

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Addd	F6	F8	F2			No	No
40	Divide	Yes	Divd	F10	F0	F6			Yes	Yes

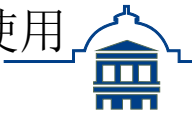
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
21	*FU*				Add		Divide			

DIVD的阻塞消除，拿到F0的内容，下拍开始执行

Cycle 22

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
39	Divide	Yes	Divd	F10	F0	F6			No	No

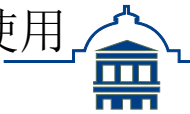
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
22	*FU*						Divide			

DIVD开始运行，执行第一拍就把F6用掉并释放；ADDD指令得以完成WB，释放Add组件

Cycle→61

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Divd	F10	F0	F6			No	No

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
61	*FU*						Divide			

DIVD执行段结束

Cycle 62

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

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

DIVD将结果写回F10，整段代码执行结束

记分牌最终状态



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

- 指令流出是完全顺序的
- 读操作数和执行是乱序的
- 写结果顺序是乱序的

Functional unit status:

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

Register result status:

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

记分牌最终状态

- 指令流出是完全顺序的
- 读操作数和执行是乱序的
- 写结果顺序是乱序的



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Qj	Rj	状态
≠0	0	OPD未就绪
0	1	OPD未取用
0	0	OPD已取用
≠0	1	不存在！

Functional unit status:

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

Register result status:

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

记分牌动态调度

- **基本思想：RAW采用乱序执行法，WAR及WAW采用阻塞法**
- **结构组织**
  - 流水线为IF+IS+OF+EX+WB，OF段从RF读OPD
  - 指令窗口及调度机构为集中式(记分牌)
- **指令发射**
  - IS段将指令信息写入指令窗口(3张状态表)
- **指令调度**
  - 在适当段检测，基于部件处理RAW，WAR/WAW阻塞

指令状态表(1行/指令)

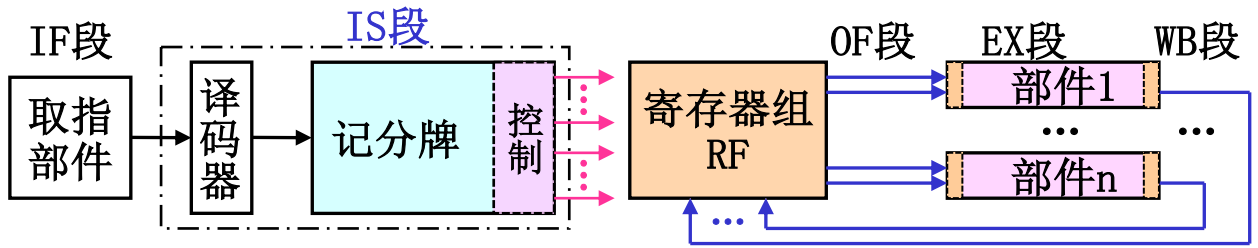
IS	OF	EX	WB

功能部件FU状态表（1行/部件）

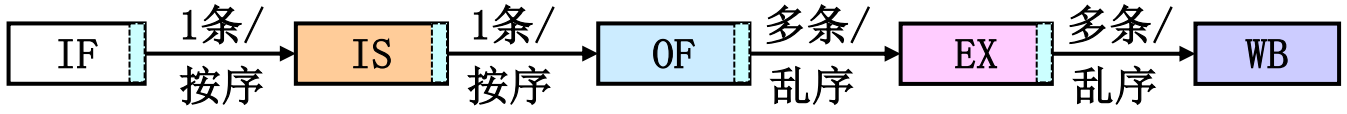
名称	Bs	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk

RF状态表(1行/REG)

Qi	Vi



指令流水组织



指令	指令状态 IS	OF	EX	WB
F1=M[R4]	√	√		
F0=F1+F2	√			
F1=F3+F3				

名称	功能部件状态（1行/部件） Bs	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	1	LD	F1	R4	/	0	/	1	/
Add1	1	ADD	F0	F1	F2	Int	0	0	1

编号	寄存器状态 Qi	Vi
F0	Add1	2
F1	Int	4

IF段— 无被阻塞指令时，取当前指令(写入指令寄存器)

IS段— 当前指令无结构&WAW冒险时，发射该指令(写入状态表)

对应部件空闲→┘ └←输出相关源指令已完成WB (WAW阻塞法)

OF段— 各功能部件行在其源OPD就绪时，从RF中读出数据

└←1条指令/行 └←正相关源指令已完成WB (RAW阻塞法)

EX段— 各部件进行相应操作 ←无进入条件(结果是/否有效)

WB段— 各功能部件行无WAR冒险时，将结果写入RF中

└←反相关源指令已完成OF (WAR阻塞法)



示例一若IF/IS/OF/WB段均时延为1Tc，EX段中Int、Add、Mult时延分别为3Tc、2Tc、10Tc，分析下列指令的流水过程

t2：LD指令进入IS(无结构&WAW冒险)　　←段时延1 T_C指进入到完成只需1拍

指令	IS	OF	EX	WB
F1=M[R4]	√			
F3=F1-F2				
F4=F1*F3				
F1=F2+F5				

名称	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	1	LD	F1	R4	/	0	/	1	/
Add1	0								
Add2	0								
Mult	0								

	Qi	值
F1	Int	
F2	0	4
F3	0	
F4	0	

OPD就绪&未取走时=1

t3：LD指令进入OF(无RAW[Rj=1时])，SUB指令进入IS

	IS	OF	EX	WB
F1=M[R4]	√	√		
F3=F1-F2	√			
F4=F1*F3				
F1=F2+F5				

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	1	LD	F1	R4	/	0	/	0	/
Add1	1	SUB	F3	F1	F2	Int	0	0	1
Add2	0								
Mult	0								

	Qi	值
F1	Int	
F2	0	4
F3	Add1	
F4	0	

OPD取走时Rj←0

	1	2	3
F1=M[F4]	IF	IS	OF
F3=F1-F2		IF	IS
F4=F1*F3			IF
F1=F2+F5			

思考：OF段的结果放在哪？

对应部件的OF/EX段间寄存器中

为什么在OF段就修改Rj？

在OF段或在EX段第一拍修改Rj取决于OF/EX的段间寄存器是如何设置的。如果是每个FU各有一个段间寄存器，那么是在OF段

t4：LD指令进入EX(OF已完成)，SUB阻塞在IS段(有RAW)，MUL指令进入IS

	IS	OF	EX	WB
F1=M[F4]	√	√	√	
F3=F1-F2	√			
F4=F1*F3	√			
F1=F2+F5				

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	1	LD	F1	R4	/	0	/	0	/
Add1	1	SUB	F3	F1	F2	Int	0	0	1
Add2	0								
Mult	1	MUL	F4	F1	F3	Int	Add1	0	0

	Qi	值
F1	Int	
F2	0	4
F3	Add1	
F4	Mult	

√表示处于该段(进入~离开)

t5-t6：LD完成EX，SUB、MUL阻塞在IS(有RAW)，ADD指令阻塞在IF(有WAW)

t7：LD进入并完成WB(写结果/释放部件/处理RAW冒险[修改SUB及MUL状态])，
SUB阻塞在IS(拍结束时改状态[下拍生效])，MUL、ADD的阻塞同上

	IS	OF	EX	WB
F1=M[F4]	√	√	√	√
F3=F1-F2	√			
F4=F1*F3	√			
F1=F2+F5				

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	0	LD	F1	R4	/	0	/	0	/
Add1	1	SUB	F3	F1	F2	0	0	1	1
Add2	0								
Mult	1	MUL	F4	F1	F3	0	Add1	1	0

	Qi	值
F1	0	9
F2	0	4
F3	Add1	
F4	Mult	

	1	2	3	4	5	6	7	8	9	10	11	12	13~22	23
F1=M[F4]	IF	IS	OF	EX	EX	EX	WB							
F3=F1-F2		IF	IS	z	z	z	z	OF	EX	EX	WB			
F4=F1*F3			IF	IS	z	z	z	z	z	z	z	OF	EX	WB
F1=F2+F5				IF	z	z	z	IS	OF	EX	EX	z	WB	

t8：SUB进入OF(RAW已消除)，MUL被阻塞(有RAW)，ADD进入IS(WAW已消除)

	IS	OF	EX	WB
F1=M[F4]	√	√	√	√
F3=F1-F2	√	√		
F4=F1*F3	√			
F1=F2+F5	√			

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	0	LD	F1	R4	/	0	/	0	/
Add1	1	SUB	F3	F1	F2	0	0	0	0
Add2	1	ADD	F1	F2	F5	0	0	1	1
Mult	1	MUL	F4	F1	F3	0	Add1	1	0

	Qi	值
F1	Add2	9
F2	0	4
F3	Add1	
F4	Mult	

t9-t11：SUB完成EX、WB，MUL被阻塞(t11改状态/下拍生效)，ADD完成OF、EX

	IS	OF	EX	WB
F1=M[F4]	√	√	√	√
F3=F1-F2	√	√	√	√
F4=F1*F3	√			
F1=F2+F5	√	√	√	

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	0	LD	F1	R4	/	0	/	0	/
Add1	0	SUB	F3	F1	F2	0	0	0	0
Add2	1	ADD	F1	F2	F5	0	0	0	0
Mult	1	MUL	F4	F1	F3	0	0	1	1

	Qi	值
F1	Add2	9
F2	0	4
F3	0	5
F4	Mult	

t12：MUL进入OF，ADD的WB被阻塞(有WAR[等待MUL完成OF])

	IS	OF	EX	WB
F1=M[F4]	√	√	√	√
F3=F1-F2	√	√	√	√
F4=F1*F3	√	√		
F1=F2+F5	√	√	√	

	Bs	Oper	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	0	LD	F1	R4	/	0	/	0	/
Add1	0	SUB	F3	F1	F2	0	0	0	0
Add2	1	ADD	F1	F2	F5	0	0	0	0
Mult	1	MUL	F4	F1	F3	0	0	0	0

	Qi	值
F1	Add2	9
F2	0	4
F3	0	5
F4	Mult	

t13：MUL开始EX，ADD完成WB(WAR已消除)

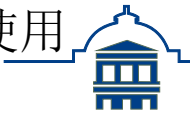
t14-t23：MUL完成EX、WB

记分牌的问题

1．解决冒险时不支持转发法，均采用阻塞法

Cycle 7

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

是否可以读出MULTD操作数？

- 不能,要等F2写回

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Multd	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	Multi1	Integer			Add				

LD进入地址计算，Rk置为空闲，并完成计算；SUBD指令流出，占用Add部件，但是源OPD F2未就绪

Cycle 8a

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

是否读MULTD和SUB的操作数？

- 不能,要等F2写回

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Multd	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

Register result status:

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

DIVD指令流出，占用Divide部件，但是源OPD F0未就绪，是乘法器Mult1的结果；LD前半拍去内存读数

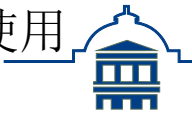
记分牌的问题

1. 解决冒险时不支持转发法，均采用阻塞法

2. 每个部件仅有一个指令缓冲位置，会阻塞后续指令

Cycle 9

Fi：目的寄存器编号　　*Qj*，*Qk*：向*Rj*，*Rk*中写结果的功能部件
Fj，*Fk*：源寄存器编号　　*Rj*，*Rk*：标示*Fj*，*Fk*是否就绪，是否已经被使用



Instruction status:

Instruction		*j*	*k*	*Issue*	*Oper Fetch*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

是否流出ADDD?

- 不能,部件被SUBD占用

Functional unit status:

记录剩余时间→

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
10	Mult1	Yes	Multd	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Subd	F8	F6	F2			Yes	Yes
	Divide	Yes	Divd	F10	F0	F6	Mult1		No	Yes

Register result status:

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

更新计数器，两条长指令开始进行计算



Part I. 为什么要学习指令级并行？

Part II. 什么是指令级并行？

Part III. 如何开发指令级并行？—数据相关动态调度

基本思想→记分牌算法（1964）→**Tomasulo算法（1967）**

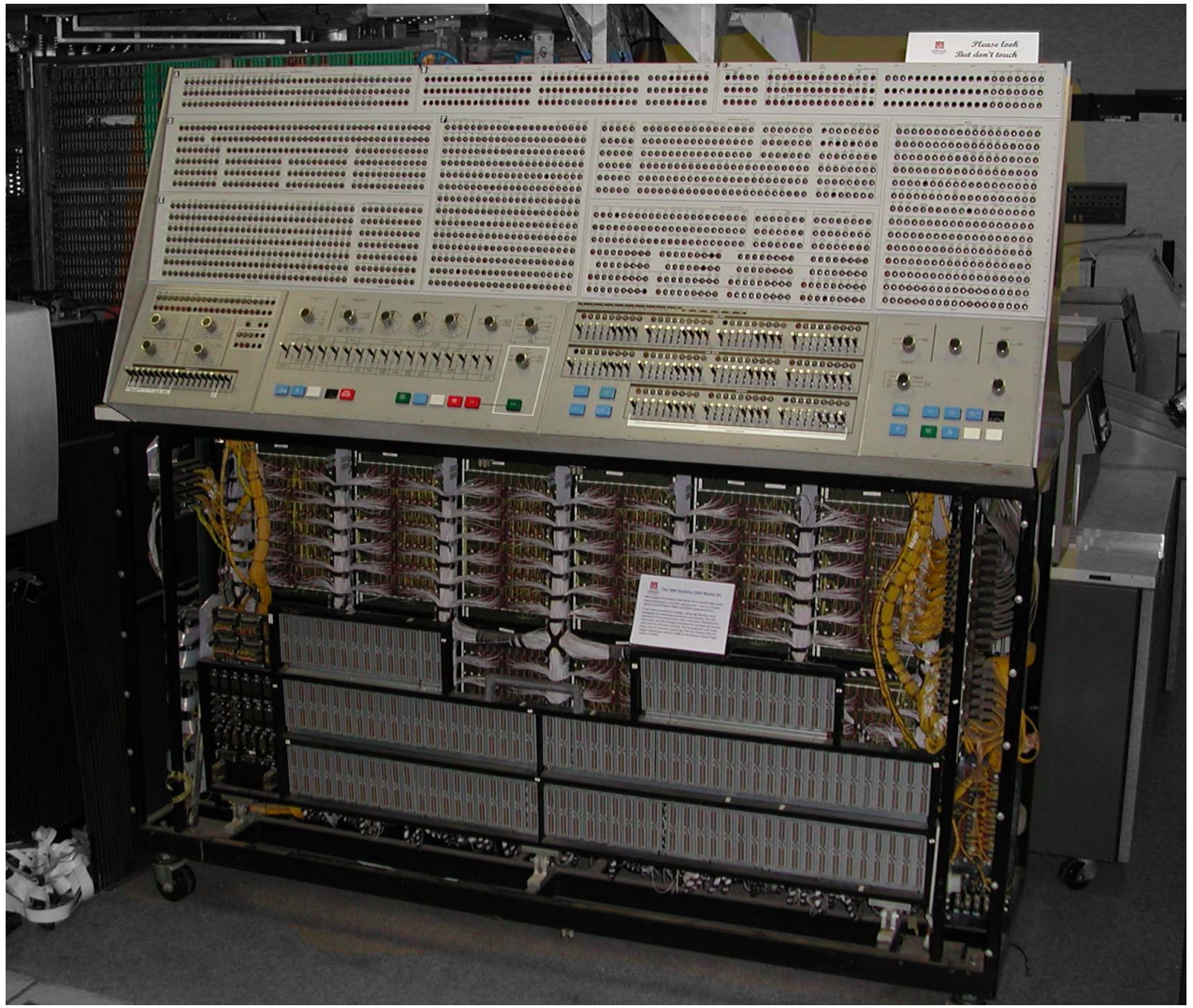
Tomasulo算法产生的背景



- **IBM 360/91比CDC6600晚三年推出**

- **整个IBM 360系列仅一个指令系统和一个编译器**
  - 要求具有很高的浮点性能，但不是通过高端机器的专用的编译器实现
  - 只有四个双精度浮点寄存器，编译器调度的有效性受到很大限制

IBM 360/91前面板



记分牌的问题→Tomasulo的改进



1. 解决冒险时不支持转发法，均采用阻塞法

Tomasulo改进1：采用转发法实现，优化性能

2. 每个部件仅有一个指令缓冲位置，会阻塞后续指令

Tomasulo改进2：采用分布式指令窗口及调度机构

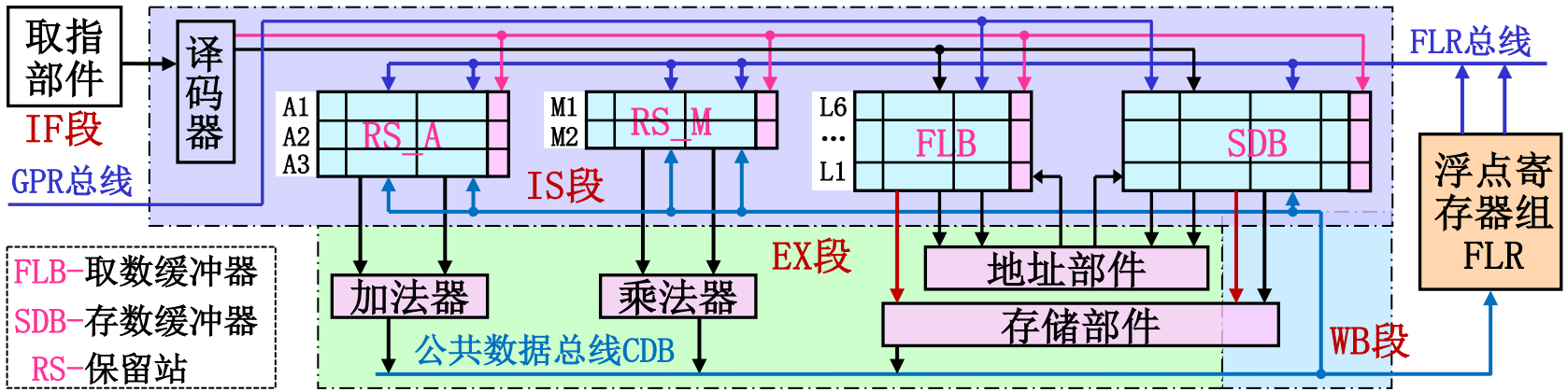
Tomasulo改进3：额外设置临时寄存器（寄存器重命名），来存放阻塞指令的寄存器内容，消除WAW、WAR冒险

Tomasulo流水线结构

支持指令例—Fd←(Fs)op(Ft)，Ft←M[(rs)+Imm]等　←同记分牌

流水线组成—IF＋IS＋EX＋WB，OF含在IS(读FLR)及WB(转发)中

1个/种部件(有窗口)→　└←同时处理1条指令(总线转发)



指令窗口—RS+FLB+SDB，均可存放多个操作及源OPD

(基于部件)　含OP+OPD状态→　└←REG重命名所需

转发通路—公共数据总线Common Data Bus（CDB）

调度机构—IS段检测冒险、处理结构及WAR/WAW冒险(REG重命名)，

RS/FLB/SDB独立地处理RAW冒险(乱序调度)

Tomasulo算法流程

流水线组成—IF＋IS＋EX＋WB，OF含在IS(读FLR)及WB(转发)中

1个/种部件(有窗口)→┘　　└←同时处理1条指令(总线转发)

- **流出**
  - 从浮点指令队列中取出一条指令
  - 如果存在一个空的保留站，就流出这条指令
  - 如果操作数在寄存器中，就送到该指令对应的保留站
  - 存储器取/存指令只要有空闲的缓存就可以流出
  - **如果没有空闲的保留站或者缓存，就存在结构相关，指令暂停，直到有空闲的保留站或者缓存**

站号(名)	Bs	Op	Vj	Vk	Qj	Qk	A
0001(L1)起		CA/MR	基REG				偏移/地址
1000(M1)起		运算	OPD1	OPD2	1站号	2站号	
1010(A1)起							
(SDB1)起		CA/MW	基REG	数据	基站号	数站号	偏移/地址

名称	Vi	Qi
F0	值	站号
F1		
…		

Tomasulo算法流程

流水线组成—IF＋IS＋EX＋WB，OF含在IS(读FLR)及WB(转发)中

1个/种部件(有窗口)→┘　　└←同时处理1条指令(总线转发)

- **执行（Execute）**
  1、如果缺少一个或者多个操作数，就监听CDB
    - 这个阶段实际是检测和自动维护RAW相关
  2、如果两个操作数都就绪，这条指令就可以执行

- **结果写回（Write result）**
  1、如果结果已经产生，将其写到CDB上
  2、通过CDB，把这个结果写到目标寄存器和等待这个结果的所有功能单元的保留站

指令序列如何工作

- **用以下代码段来说明Tomasulo算法原理**
  1. LD F6, 34(R2)
  2. LD F2, 45(R3)
  3. MULTD F0, F2, F4
  4. SUBD F8, F6, F2
  5. DIVD F10, F0, F6
  6. ADDD F6, F8, F2
- **假设**
  - 1、Load、加法、减法指令需要2个时钟周期
  - 2、乘法指令需要10个
  - 3、除法指令需要40个

Tomasulo算法: Cycle 0



指令流

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load缓冲，还应有3个Store缓冲，未画出

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

FU 局部计数器

3 FP Adder保留站.
2 FP Mult保留站

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
0									
FU									

时钟计数器

流水线组成—IF＋IS＋EX＋WB，OF含在IS(读FLR)及WB(转发)中

1个/种部件(有窗口)　　同时处理1条指令(总线转发)

Cycle 1

Instruction status:

Instruction		j	k	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
1	*FU*				Load1					

LD流出，IS段取操作数，放进Load1缓存中；更新寄存器状态表

Cycle 2



Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

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

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

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

第二条LD流出，IS段取操作数，放进Load2缓存中；更新寄存器状态表

Cycle 3

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

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

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		F4	Load2	
	Mult2	No					

Register result status:

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

MULTD流出，进入保留站Mult1，F4内容取出放进Vk， 另一OPD未就绪；第一条LD完成访存，下拍写结果

Cycle 4

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		F4	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(A1)	Add1				

A1=34+R2

SUBD指令流出；第一条LD WB写F6完成，清空Load1，同时CDB转发给需要的SUBD；

Cycle 5

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

一条指令的V域全部写满，代表其操作数全部准备好，可以开始执行

Register result status:

Clock 5

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	M(A2)		M(A1)	Add1	Mult2			

A2=45+R3

DIVD指令流出；第二条LD写回，CDB广播其结果，Add1和Mult1均更新至V域

Cycle 6

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	M(A2)		Add2	Add1	Mult2			

ADDD指令流出；

Cycle 7

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(A2)		Add2	Add1	Mult2			

SUBD执行完成，下一拍写结果

Cycle 8

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

SUBD写回F8，同时广播；Add2收到广播更新自己Vj，OPD均就绪，下拍开始执行

Cycle 9



Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

继续执行

Cycle 10



Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

ADDD指令EX结束

Cycle 11



Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2			

ADDD写回，从保留站退出，并广播CDB；但是没有等待ADDD结果的指令

Cycle 12

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	*FU*	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2			

Cycle 13



Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
13	*FU*	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2			

继续执行

Cycle 14



Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	*FU*	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2			

继续执行

Cycle 15



Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	F4		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2			

MULTD执行完毕，下一拍写结果

Cycle 16

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
16	*FU*	M*F4	M(A2)		(M-M+M)	(M-M)	Mult2			

MULTD写结果完毕，退出保留站，更新寄存器状态表，CDB广播；Mult2收到，更新Vj，OPD就绪，下拍执行

Cycle ->55



Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU	M*F4	M(A2)		(M-M+M)	(M-M)	Mult2			

DIVD继续执行

Cycle 56

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)		(M-M+M)	(M-M)	Mult2			

DIVD执行段结束，下拍写回

Cycle 57

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU	M*F4	M(A2)		(M-M+M)	(M-M)	Result			

DIVD写回，并向CDB广播结果

Tomasulo算法最终状态



Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	No					

- 指令流出是完全顺序的
- 执行是乱序的
- 写结果是乱序的

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU	M*F4	M(A2)		(M-M+M)	(M-M)	Result			

Tomasulo算法最终状态

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	No					

同样的程序段

记分牌（62 cycles）

- 1整数+1加+2乘+1除部件

Tomasulo（57 cycles）

- 1LD+1加+1乘部件

Register result status:

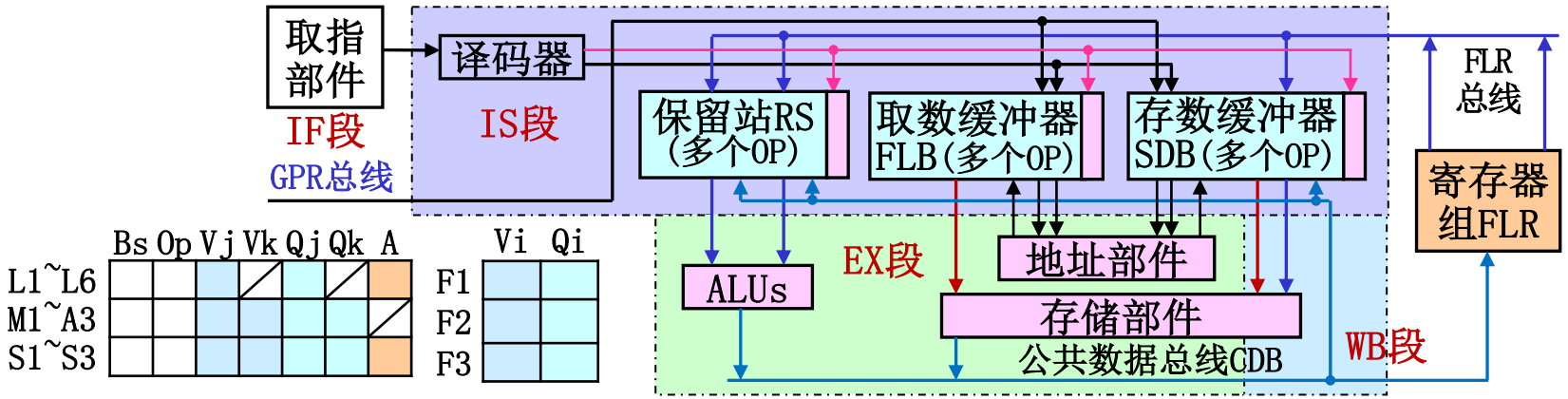
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU	M*F4	M(A2)		(M-M+M)	(M-M)	Result			

Tomasulo动态调度小结

*基本思想：RAW采用乱序执行＋转发法，WAR及WAW自动消除

*结构组织：流水线为IF+IS+EX+WB，OF含在IS(读FLR)及WB(转发)中；

指令窗口及调度机构为分布式(RS+FLB+SDB)



*指令发射：在IS段将指令信息写入指令窗口(RS/FLB/SDB+FLR)

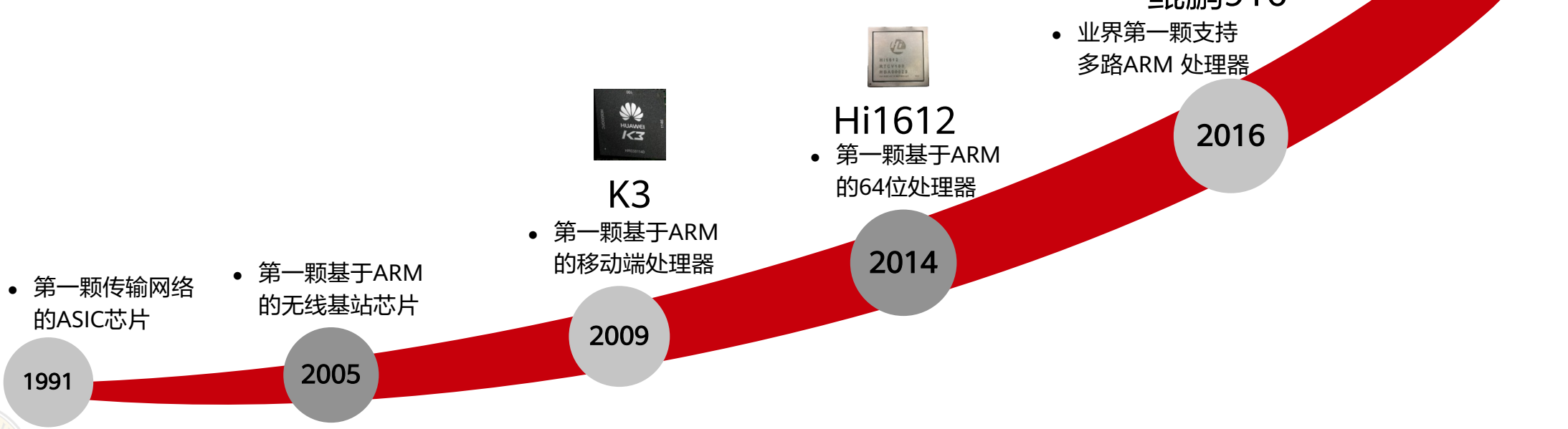
*指令调度：在IS段检测，基于子窗口处理RAW，消除WAR/WAW

OPD就绪的指令优先执行→┘　　└←REG重命名法

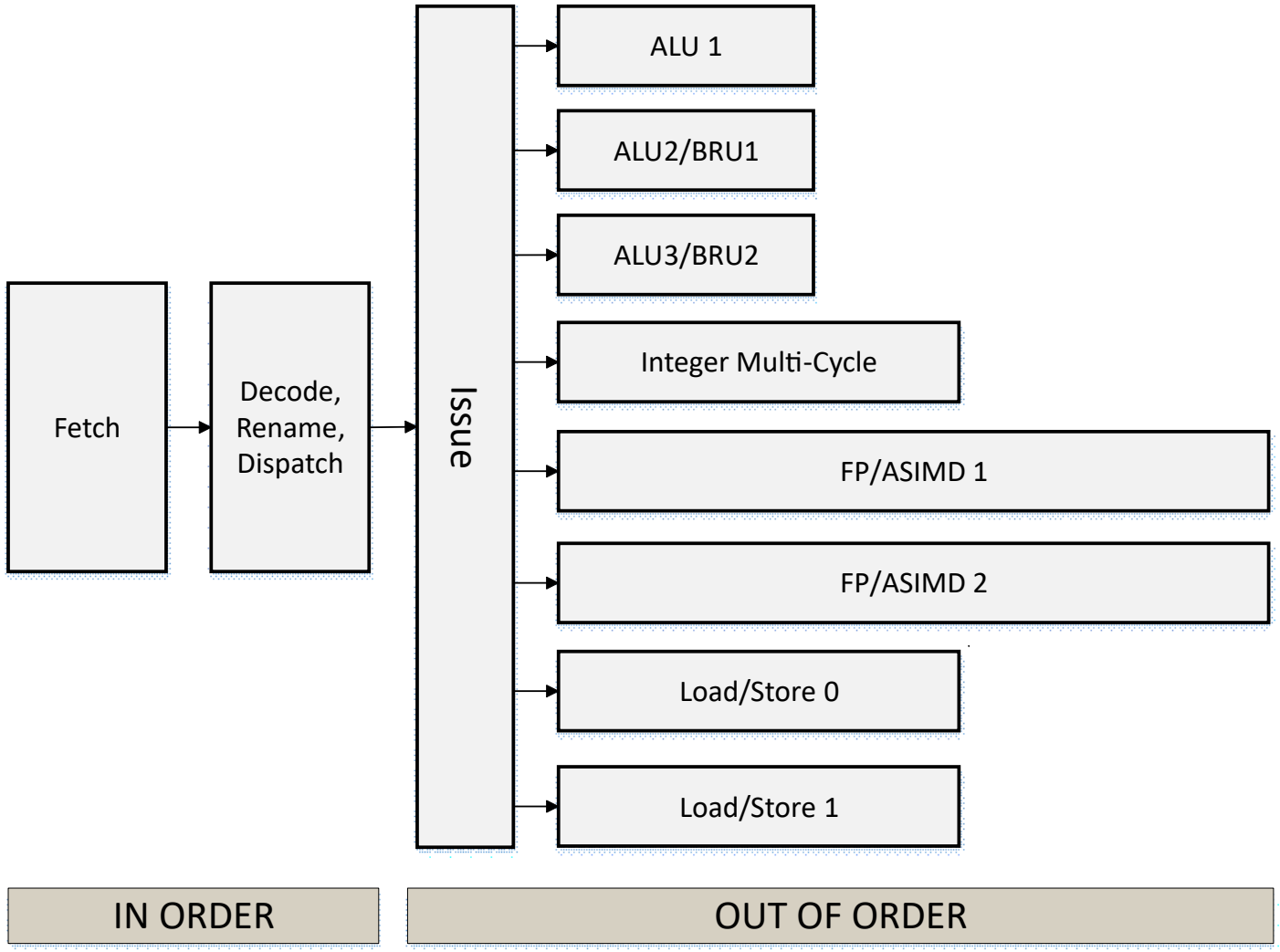
	冒险条件	处理方法 (设当前指令占用RS[c]行)
RAW WAR	F[s].Qi≠0	RS[c].Qj←F[s].Qi，指令被阻塞在IS段
	F[s].Qi=0	RS[c].Vj←F[s].Vi，RS[c].Qj←0，冒险消除
WAW	F[d].Qi≠0	F[d].Qi←c，冒险消除　(设F[d]为目的OPD)
结构	RS[c].Bs=1	指令被阻塞在IF段　(设F[s]为源OPD)

现代处理器中动态调度应用 (以鲲鹏为例)

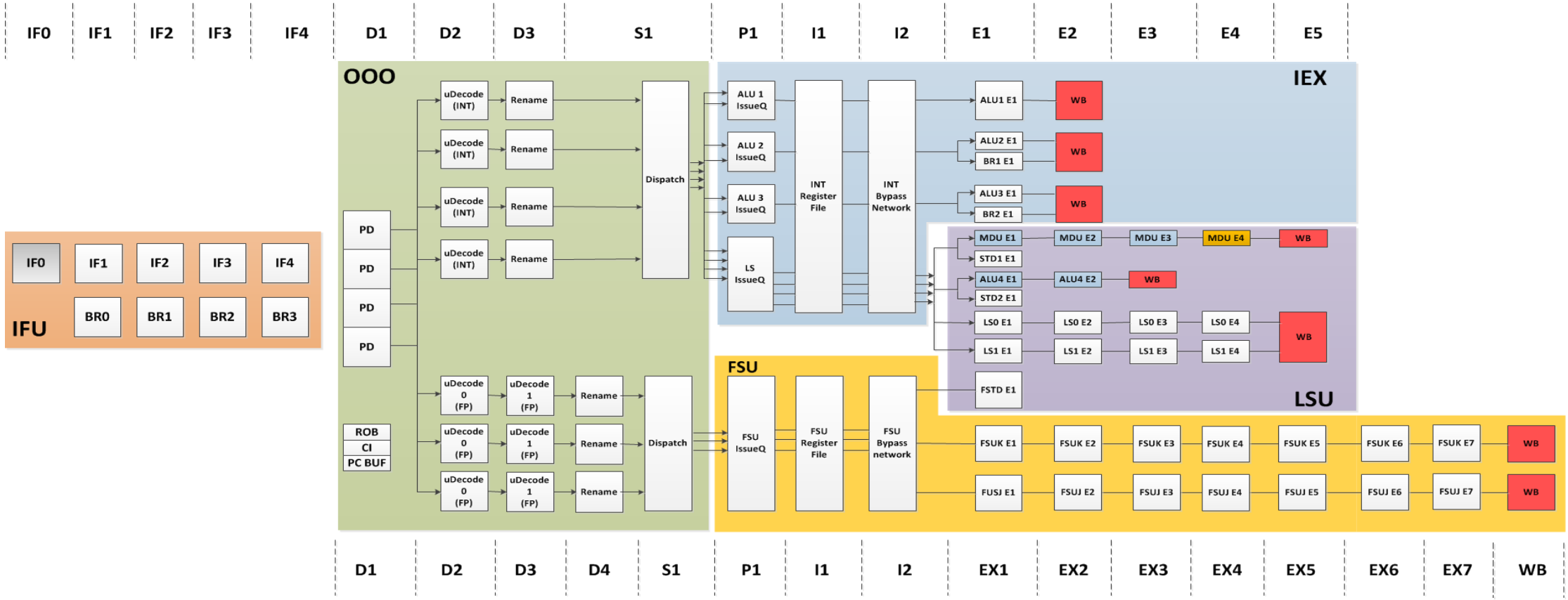
- **华为鲲鹏处理器**
  - ARM架构、面向数据中心/服务器场景
- **典型使用场景**
  - AI计算、边缘计算、HPC、Web应用等



现代处理器中动态调度应用 (以鲲鹏为例)



基于ARMv8的华为鲲鹏处理器流水线



Taishan coreV110 Pipeline Architecture

Tomasulo动态调度方法评价



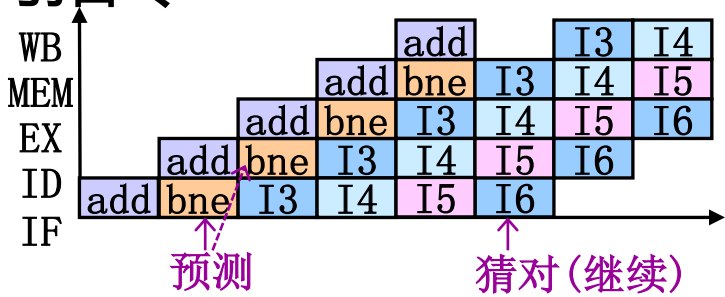
- **动态方法能够达到很高的性能**
- **主要缺陷**
  - 高复杂性：需要大量硬件
  - 存在瓶颈：单个公共数据总线（CDB）引发竞争
    - 解决方法：在每个保留站上需要为每条CDB设置重复的硬件接口



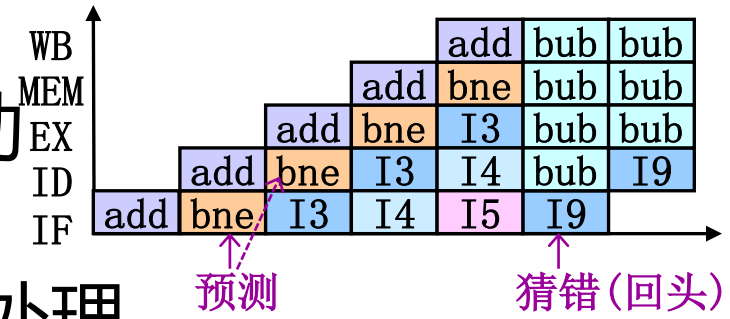
Part I. 为什么要学习指令级并行？
Part II. 什么是指令级并行？
Part III. 如何开发指令级并行？—数据相关动态调度
Part IV. 如何开发指令级并行？—动态分支预测

回顾：流水线的分支预测

- **预测法：** 预测转移方向(转/不转)，并执行该方向的指令
  - 猜对时继续执行后续指令，猜错时回头执行反方向指令
- **预测法的内涵**
  - 任务：预测是否转移、获得分支目标地址
  - 时机：IF段（不停顿）或ID段（停1拍）
  - 方法：静态预测（盲猜）、动态预测（基于转移历史）



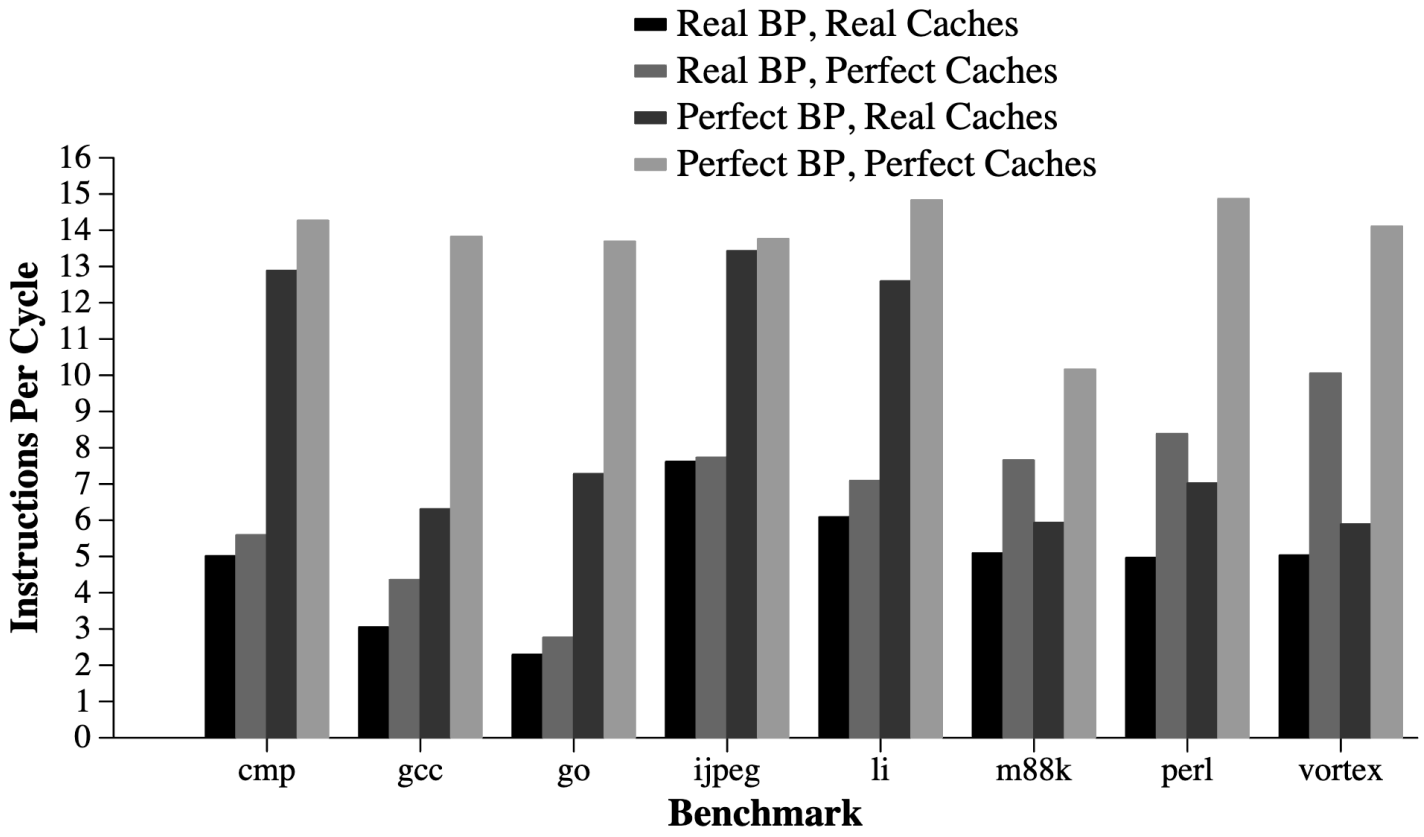
- **动态分支预测**
  - 预测算法：动态预测(按转移历史)、静态预测可辅助
  - 转移历史管理：历史保存硬件、历史信息管理
  - 预测处理流程：分支预测、历史更新、预测结果处理



优秀的分支预测带来的性能提升



- 100%的完美分支预测能带来~2x的性能提升 (SPECint95)



*相比自适应预测算法

Chappell, Robert S., et al. "Simultaneous subordinate microthreading (SSMT)." Proc. of International Symposium on Computer Architecture (ISCA). 1999.



Part I. 为什么要学习指令级并行？
Part II. 什么是指令级并行？
Part III. 如何开发指令级并行？—数据相关动态调度
Part IV. 如何开发指令级并行？—动态分支预测

分支预测算法→转移历史管理→预测处理流程

分支预测算法



- **静态预测算法**
  - 分类：定向预测（不转移/转移）、按指令内容预测
    - 定向预测 → 可在IF段预测
    - 按指令内容预测 → 需在ID段预测

Intel PII的预测算法(按内容)			预测转移	预测不转移	说明
CALL/RET/JMP指令			√		跳转型
其它指令	相对寻址	偏移量为负数	√		常为循环
		偏移量为正数		√	常为分支
	非相对寻址			√	未知

- **动态预测算法**
  - 分类：饱和计数预测法、相关预测法、自适应预测法等(均可在IF段预测)

- **要求：**掌握饱和计数预测法，其余了解



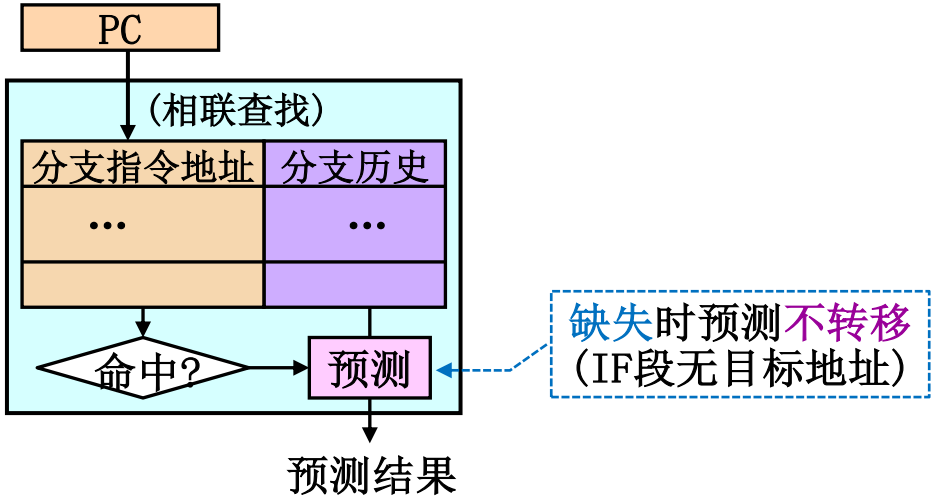
饱和计数预测器 又称基本预测器、(0,n)预测器

思想： 根据当前分支的历史来预测

组成： 分支历史表BHT + 预测器

├←Branch History Table，类似于Cache

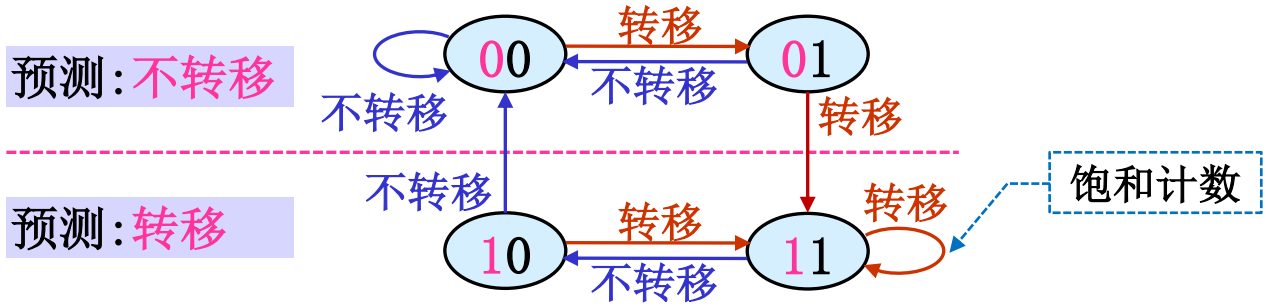
└←分支历史占2位：(0, 2)预测器



算法：

根据分支历史高位预测转移方向(1时转移)，

根据执行结果更新分支历史(饱和计数)



回忆：什么是相联查找？

- 根据存储内容来进行查询的存储器
- 可以实现高速Cache存放地址

特点：

预测方向在连续2次误预测改变； ←可优化预测性能

预测依据仅基于当前分支指令历史； ←未参考前驱分支指令

多分支预测(类似switch)性能较差 ←仅命中一行

饱和计数器的问题：分支结果相关性



- **局部分支相关性** Local branch correlation
  - 一条分支的结果和它前几次的执行结果有关

- **全局分支相关性** Global branch correlation
  - 一条分支的结果和其他分支的结果有关

```
if (cond1)
...
if (cond1 AND cond2)
```

if (x<1) . . .
if (x>1) . . .

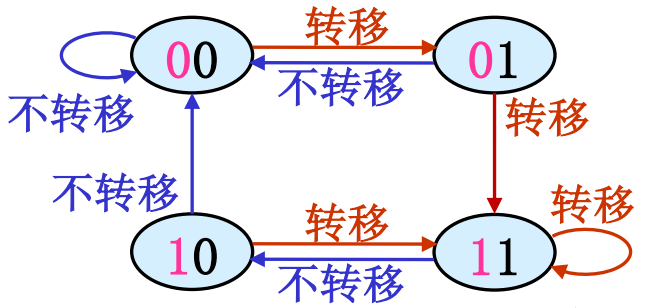
branch Y: if (cond1)
...
branch Z: if (cond2)
...
branch X: if (cond1 AND cond2)

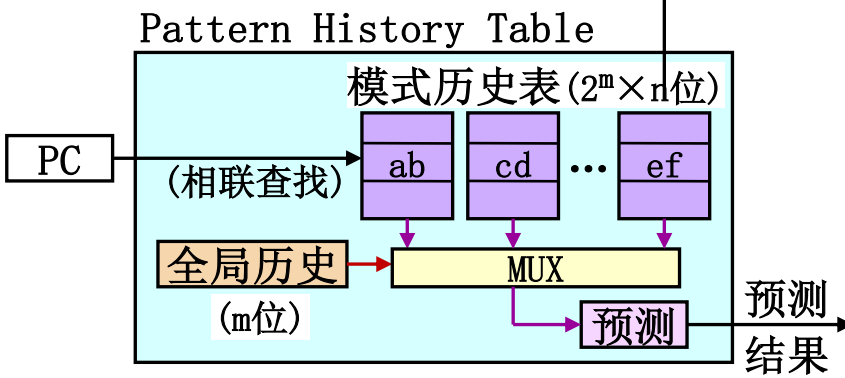
相关预测器（了解）　又称两级预测器

思想： 参考其他分支的行为来预测

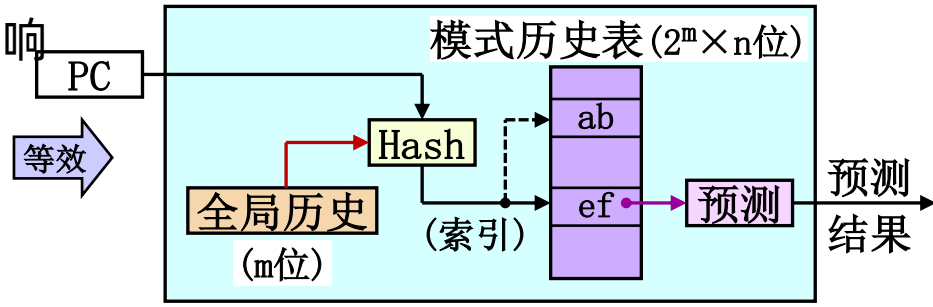
组成： 全局历史 + 模式历史表 + 预测器　←称为(m,n)预测器
└←前m个分支的行为　└←每个分支有2^m种转移模式(各n位)

算法： 根据全局历史(m位)，选择当前所用模式历史；
根据所选模式历史(n位)，预测转移方向；
根据执行结果，更新全局历史、所选模式历史



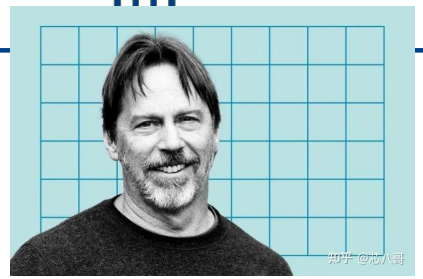


特点： 预测性能远超(0,n)预测器，　←当前分支结果受前驱分支影响
多分支预测性能较差　←仅命中一行



DEC Alpha 21264与Jim Keller

- 相对于IBM的大型机，DEC公司引领了小型机的浪潮
- 超级计算机神威•太湖之光就是从DEC Alpha衍生出来的





“硅仙人”AMD Ryzen之父吉姆·凯勒职业之路

吉姆·凯勒（Jim Keller）工作经历

时间	公司	职位	贡献
1980-1998	DEC	芯片设计师	负责Alpha系列芯片
1998-1999	AMD	首席架构师	负责K7/K8
1999-2000	SiByte	首席芯片架构师	负责MIPS芯片设计
2000-2004	Broadcom	首席芯片架构师	负责MIPS芯片设计
2004-2008	P.A.Semi	研发副总裁	负责低功耗手机芯片设计
2008-2012	Apple	研发副总裁	负责A4/A5芯片设计
2012-2015	AMD	首席核心架构师	领导并开发Zen架构
2016-2018	Tesla	自动驾驶工程副总裁	负责FSD自动驾驶芯片设计
2018-2020	Intel	硅工程高级副总裁	领导SoC的集成和开发
2021	Tenstorrent	总裁兼首席技术官	未完待续…

数据支持：网络公开信息，芯八哥整理

完整文章《美国“硅仙人”吉姆·凯勒的传奇之路》请扫码关注公众号阅读。

芯八哥

知乎 @芯八哥



Part I. 为什么要学习指令级并行？
Part II. 什么是指令级并行？
Part III. 如何开发指令级并行？—数据相关动态调度
Part IV. 如何开发指令级并行？—动态分支预测

分支预测算法→**转移历史管理→预测处理流程**

转移历史管理

预测的任务：预测是否转移、获得分支目标地址　←读取(IF段预测)或计算(ID段预测)

思考①：基于BHT，能在IF段预测吗？

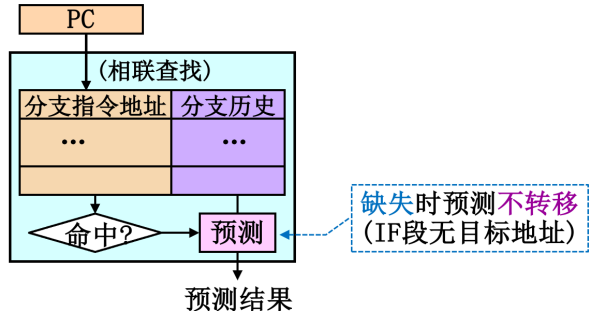
不行，得不到分支目标地址。

历史保存：使用分支目标缓冲器BTB(含转移历史位)

BTB结构：

有效	标记	转移历史	分支目标地址

BTB参数：容量＝几百行，组相联映射、LRU替换算法、无写策略



历史表示：历史含多次结果，用转移历史位表示

可用预测算法：基于转移历史的预测算法(如饱和计数法)

历史建立：BTB缺失、执行结果转移时，建立新行(假设为第k行)

如：k.有效←1、 k.标记←(PC)，

k.转移历史←11、 k.分支目标地址←执行结果

思考②：哪些指令会引起BTB缺失？

顺序型、首次/上次不转移的分支

思考③：不转移时不建新行的原因？

下次应预测不转移，不应该BTB命中

历史更新：BTB命中、执行结束时，改变匹配行中转移历史位

如：k.转移历史←k.转移历史＋r　(r取决于预测算法)

预测处理流程

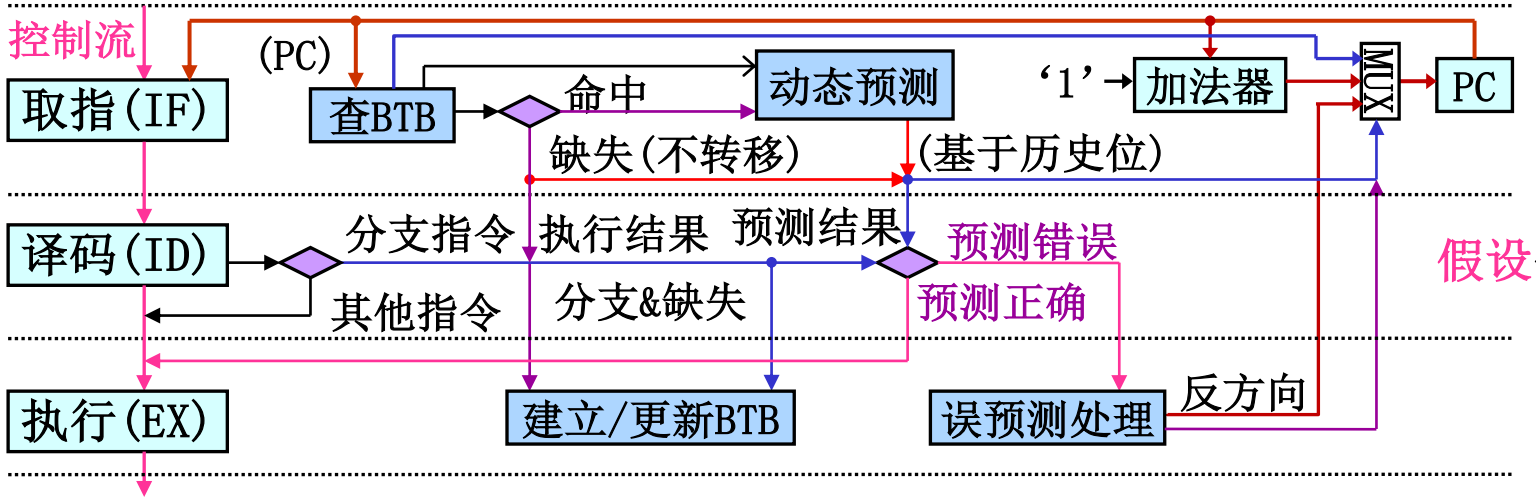
分支预测— IF段用PC查BTB，命中时预测基于历史位，缺失时不转移

预测结果处理— ID段(已有执行结果)进行判断

历史更新— EX段建立/更新转移历史(缺失&转移时/命中时)

预测结果处理— 猜对时，不写PC(IF段已写/后续指令继续执行)；

(可与历史更新同时)　　猜错时，清除已取指令，重置PC(反方向)



假设分支指令在ID段完成[写PC]

*性能分析： 预测正确时停0拍，预测错误时停2拍(比阻塞法多1拍)

*性能优化： BTB设置分支目标指令域(1～2条)　←避免回头时取指令



Part I. 为什么要学习指令级并行?
Part II. 什么是指令级并行?
Part III. 如何开发指令级并行? —数据相关动态调度
Part IV. 如何开发指令级并行? —动态分支预测
Part V. 前瞻执行技术

回顾：Tomasulo和预测执行的问题

- **传统的乱序执行不能跨基本块；但是分支指令平均6~7条就会出现**
  - 导致单纯乱序执行的效率不够高

- **Tomasulo的执行与结果写回是乱序的**
  - 导致：不精确异常

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

基于硬件的前瞻执行 Speculation

*目标：支持动态分支预测＋动态调度

- 调度范围选择：跨基本块
- 保持数据流：分支指令完成前，预测指令不写结果
- 保持异常行为：预测指令在分支指令之后按序产生

*基本思想：①用动态分支预测技术选择后续指令

②控制冒险消除前可执行后续指令 ←即预测的指令

③用动态调度技术处理数据冒险 ←跨基本块调度

指令串示例

I1：$1←$2－5

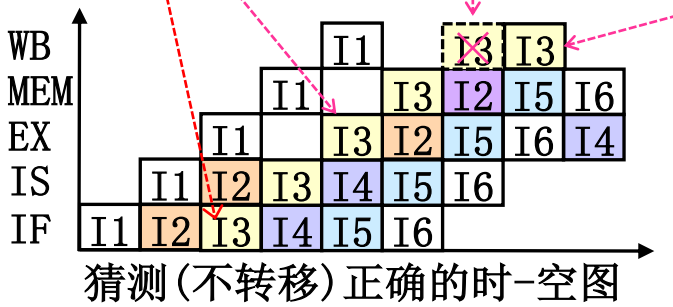
I2：$1=$0时PC←I9

I3：$3←$4＋$5

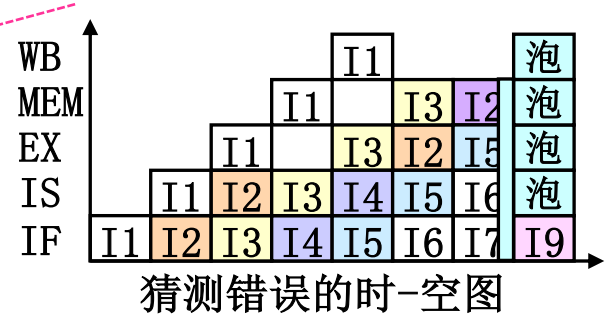
I4：$3=$0时PC←I7

I5：$4←$5－$6

可乱序执行、但不写结果 → 按序确认后才写结果



猜测(不转移)正确的时-空图



猜测错误的时-空图

核心—根据分支预测结果取指令，根据动态调度结果执行指令，

根据按序确认结果写结果；确认时才处理异常

流水线结构（改进Tomasulo）



部件组织— ①增设BTB(Branch Target Buffer，分支目标缓冲器)，实现动态分支预测

按序确认时需要

②增设ROB(ReOrder Buffer，再定序缓冲器)，实现指令排队及结果暂存

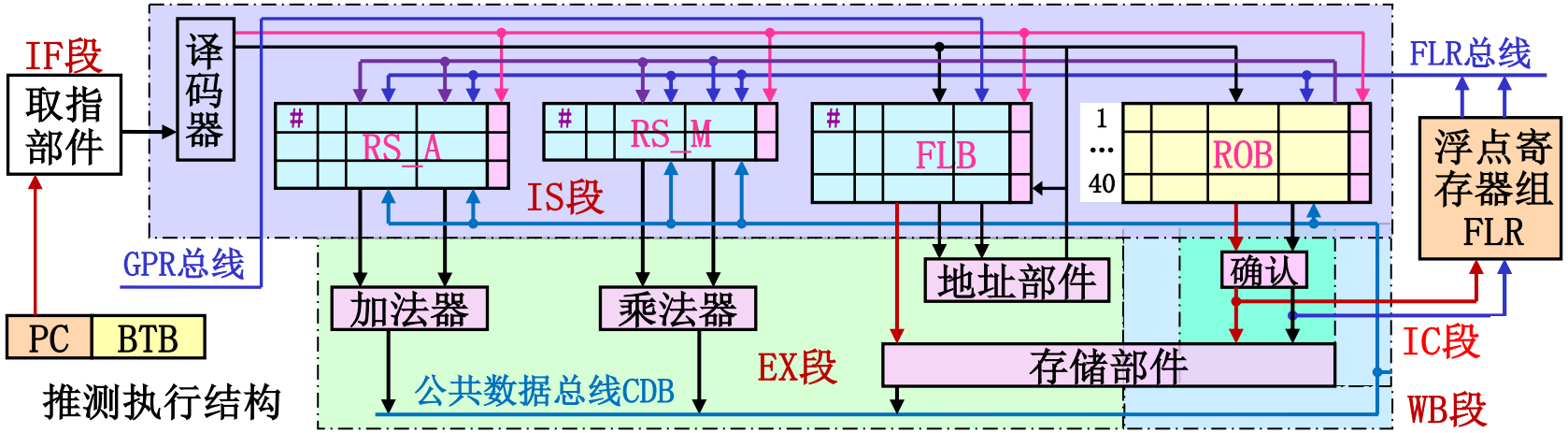
取消SDB←——用作临时FLR和SDB←┴→WB段写ROB

Instruction Commit

③增设确认段(IC)，实现确认后写结果及处理异常

└←IC段写FLR/MEM

流水线组成— IF+IS+EX+WB+IC



推测执行结构

思考①：Tomasulo的RS中源OPD来自FLR/CDB，现在呢？ FLR/CDB/ROB(临时FLR)

思考②：确认段的“按序”如何实现？ ROB按序流入&流出、IC段源自ROB

思考③：误预测原来需清除分支指令后的指令，现在呢？ 整个流水线(分支为队首)

状态表组成：RS+FLB+ROB+FLR

ROB组成—有4个域，指令按序存放　←不保存源OPD[在RS中]

再定序缓冲器（ROB）

项号	Bs	Type	TAddr	TVal	Stat
已确认完→ 1	0	ST	MEM地址	aa	
等待确认→ 2	1	OT	F4	bb	WB
3	1	BR		cc	IS
乱序执行→ 4	1	OT	F1	dd	EX
…	0				

说明：

Type—表示写结果类型(写PC/MEM/REG)，有分支BR/存数ST/其他OT

Stat—表示执行状态，如处于IS/EX/WB

TAddr—目的地址

TVal—目的值，Stat=WB时值有效(推测)

其余表组成—①Qi/Qj/Qk内容为ROB项号　←ROB的TVal为临时FLR

②RS/FLB不设置站号、增设ROB#项

└←站号已无用　└←表明结果产生位置

	Vi	Qi
Fd		x
Fs		y
Ft		z

以Fd←Fs+Ft为例

	Vi	Qi
Fd		x
Fs		y
Ft		z

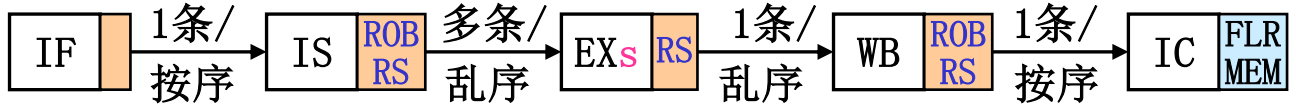
ROB#	Bs	类型	地址	值	状态
y	1	OT	Fs		
z	1	OT	Ft		
x	1	OT	Fd		

站号	Bs	Op	Vj	Vk	Qj	Qk	A	
	1		20		0		4	FLB
x	1				y	z		RS
无								SDB

Bs	Op	Vj	Vk	Qj	Qk	A	ROB#	
1		20		0		4		FLB
1				y	z		x	RS

(取消SDB)

指令流水组织



IS段—无结构冒险时，消除WAR/WAW、发射当前指令

└←ROB、RS均有空行　　├←填ROB、RS/FLB及FLR

└←OPD从FLR或ROB中取

EX段—运算部件在RS某行OPD就绪时，进行数据运算，

地址部件在FLB队首参数就绪时，进行地址计算，

└←含存数指令参数

存储部件在队首地址就绪时，进行MEM读操作

WB段—RS/FLB在CDB空闲时发送结果，等待部件接收数据，

并释放该行→┘　ROB对应行→┴←有RAW的RS行

IC段—在ROB队首指令完成时，进行确认和处理(写结果/处理异常)

存数/非分支指令：结果写入MEM/FLR，释放ROB行

预测正确的分支指令：释放ROB行　←不写PC

预测错误的分支指令：清空流水线，重置PC

硬件前瞻执行示例

示例1：若推测流水线为整数流水线，地址&存储部件、加减法器(具有+1功能)、分支部件JEU的时延分别为2、1、1T_C；BTB采用2位饱和计数预测算法。分析下列指令(入口地址为0)的执行过程及执行时间。

思路：分析各轮流水过程(注意轮间数据相关)，直到流出稳定为止

第1轮指令流水过程：

t1-t10，IF段含动态预测、IS段动态调度

└←均预测不转移(BTB缺失)

	1	2	3	4	5	6	7	8	9	10
I1: F1=F4-F2	IF	IS	EXi	WB	IC					
I2: F3=M[F1+4]		IF	IS	z(D)	EXm	EXm	WB	IC		
I3: F2=F2+1			IF	IS	EX	WB	z	z	IC	
I4: F2≠F4转I1				IF	IS	z(D)	EXj	WB	z	IC
(后续记为I5-I9)					IF	IS	EXi	z	WB	z

t2-t3：SUB指令完成IS、EX(t3)，LD指令完成IS(写ROB/RS/FLR、ROB按序使用)

	Bs	Op	Vj	Vk	Qj	Qk	A	ROB#
L1	0→1	CA			1		4	2
A1	0→1	SUB	35	5	0	0		1
A2	0							
J1	0							

	Vi	Qi
F1	10	0→1
F2	5	0
F3	0	0→2
F4	40	0

ROB#	Bs	Type	TA	TVal	Stat
1	0→1	OT	F1		IS→EX
2	0→1	OT	F3		IS
3	0				
4	0				

假设Vj用作EX的段间REG

ROB[F1.Qi].Stat=WB时为0

操作结果与目的OPD的关联

地址&存储部件、加减法器(具有+1功能)、分支部件JEU的时延分别为2、1、1T_C

I1: F1=F4-F2
I2: F3=M[F1+4]
I3: F2=F2+1
I4: F2≠F4转I1

t4：SUB完成WB(写ROB/转发)，LD被阻塞(有RAW)，INC指令进入IS段

	Bs	Op	Vj	Vk	Qj	Qk	A	ROB#
L1	1	CA	35	/	1→0	/	4	2
A1	1→0	SUB	35	0	0	0	/	1
A2	0→1	INC	5	/	0	/	/	3
J1	0							

	Vi	Qi
F1	10	1
F2	5	0→3
F3	0	2
F4	40	0

ROB#	Bs	Type	TA	TVal	Stat
1	1	OT	F1	35	EX→WB
2	1	OT	F3		IS
3	0→1	OT	F2		IS
4	0				

WB段释放相应RS行

下拍可用(节拍结束时写入)

IC段才写REG

t5：SUB完成IC(写结果/处理异常)，LD、INC进入EX段(不同部件)，BNE进入IS

L1	1	→MR	→39	/	0	/	4	2
A1	0						/	
A2	1	INC	5→6	/	0	/	/	3
J1	0→1	BNE		40	3	0	I1	4

F1	35	1→0
F2	5	3
F3	0	2
F4	40	0

1	1→0	OT	F1	35	WB
2	1	OT	F3		IS→EX
3	1	OT	F2		IS→EX
4	0→1	BR			IS

预测方向等未列出

IC段释放相应ROB行

t6：LD完成EX(2Tc)，INC完成WB(乱序)，BNE被阻塞(有RAW)，I5指令进入IS

L1	1	MR	→56	/	0	/	0	2
A1	0→1	I5	*	*	0	0	/	5
A2	1→0	INC	6	/	0	/	/	3
J1	1	BNE	6	40	3→0	0	I1	4

F1	30	0
F2	5	3
F3	0	2
F4	40	0

2	1	OT	F3		EX
3	1	OT	F2	6	EX→WB
4	1	BR			IS
5	0→1	OT		(I5)	IS

假设M[39]=56

下拍生效

节约版面(项1未列出)

地址&存储部件、加减法器(具有+1功能)、分支部件JEU的时延分别为2、1、1T_C

I1: F1=F4-F2
I2: F3=M[F1+4]
I3: F2=F2+1
I4: F2≠F4转I1



t7：LD完成WB，INC被阻塞(等待IC)，BNE完成EX(含判断预测结果)，
I5略，I6进入IS(ROB/FLR中未列出)

	Bs	Op	Vj	Vk	Qj	Qk	A	ROB#
L1	1→0	MR	56		0		0	2
A1	1	I5	*	*	0	0		5
A2	0→1	I6	*	*	0	0		6
J1	1	BNE	Err	40	0	0	I1	4

预测错误

	Vi	Qi
F1	30	0
F2	5	3
F3	0	2
F4	40	0

ROB#	Bs	Type	TA	TVal	Stat
2	1	OT	F3	56	EX→WB
3	1	OT	F2	6	WB
4	1	BR			IS→EX
5	1	OT		(I5)	IS→

t8：LD完成IC(按序)，INC被阻塞(同t7)，BNE完成WB(更新BTB)，I5-I7略

L1	0							
A1	1	I5	*	*	0	0		5
A2	1	I6	*	*	0	0		6
J1	1→0	BNE	Err	40	0	0	I1	4

	Vi	Qi
F1	30	0
F2	5	3
F3	56	2→0
F4	40	0

ROB#	Bs	Type	TA	TVal	Stat
2	1→0	OT	F3	56	WB
3	1	OT	F2	6	WB
4	1	BR	Err	I1	EX→WB
5	1	OT		(I5)	IS→

t9：INC完成IC，BNE被阻塞(等待IC)，I5-I8略

t10：BNE完成IC(清空流水线[含ROB及RS]/重写PC)

L1	0							
A1	1→0	I5	*	*	0	0		5
A2	1→0	I6	*	*	0	0		6
J1	0							

	Vi	Qi
F1	30	0
F2	6	3→0
F3	56	0
F4	40	0

ROB#	Bs	Type	TA	TVal	Stat
2	0				
3	1→0	OT	F2	6	WB
4	1→0	BR	Err	I1	WB
5	1→0	OT		(I5)	IS→

地址&存储部件、加减法器(具有+1功能)、分支部件JEU的时延分别为2、1、1T_C

I1: F1=F4-F2
I2: F3=M[F1+4]
I3: F2=F2+1
I4: F2≠F4转I1

第2-5轮指令流水过程：关注I1与上轮、I2~I4与I1的依赖关系

(第2轮)	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I1: F1=F4-F2	IF	IS	EXi	WB	IC										
I2: F3=M[F1+4]		IF	IS	z(D)	EXm	EXm	WB	IC							
I3: F2=F2+1			IF	IS	EXi	WB	z	z	IC						
I4: F2≠F4转I1				IF	IS	z(D)	EXj	WB	z	IC					
I1: F1=F4-F2					IF	IS	EXi	z	WB	z	IC				
I2: F3=M[F1+4]						IF	IS	z(D)	z(D)	EXm	EXm	WB	IC		
I3: F2=F2+1							IF	IS	EXi	WB	z	z	z	IC	
I4: F2≠F4转I1								IF	IS	z(D)	EXj	z	WB	z	IC

(第4轮)	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
I1: F1=F4-F2	IF	IS	EXi	z	z	WB	z	IC									
I2: F3=M[F1+4]		IF	IS	z(D)	z(D)	z(D)	EXm	EXm	WB	IC							
I3: F2=F2+1			IF	IS	EXi	z	WB	z	z	z	IC						
I4: F2≠F4转I1				IF	z(S)	IS	z(D)	EXj	z	WB	z	IC					
I1: F1=F4-F2					IF	IS	z(D)	EXi	z	z	WB	z	IC				
I2: F3=M[F1+4]						IF	IS	z(D)	z(D)	z(D)	z(D)	EXm	EXm	WB	IC		
I3: F2=F2+1							IF	IS	EXi	z	z	WB	z	z	z	IC	
I4: F2≠F4转I1								IF	z(S)	z(S)	IS	z(D)	EXj	z	WB	z	IC

特征—$T_{每轮流出}$受限于I1的WB&IC段(~上轮I4)，无关于I1的EX段(~上轮I3)

执行时间：$T=T_{第1轮}\times 2+(n-2)T_{每轮流出}=10*2+(35-2)*5=185Tc$

示例2：Pentium II CPU动态执行技术

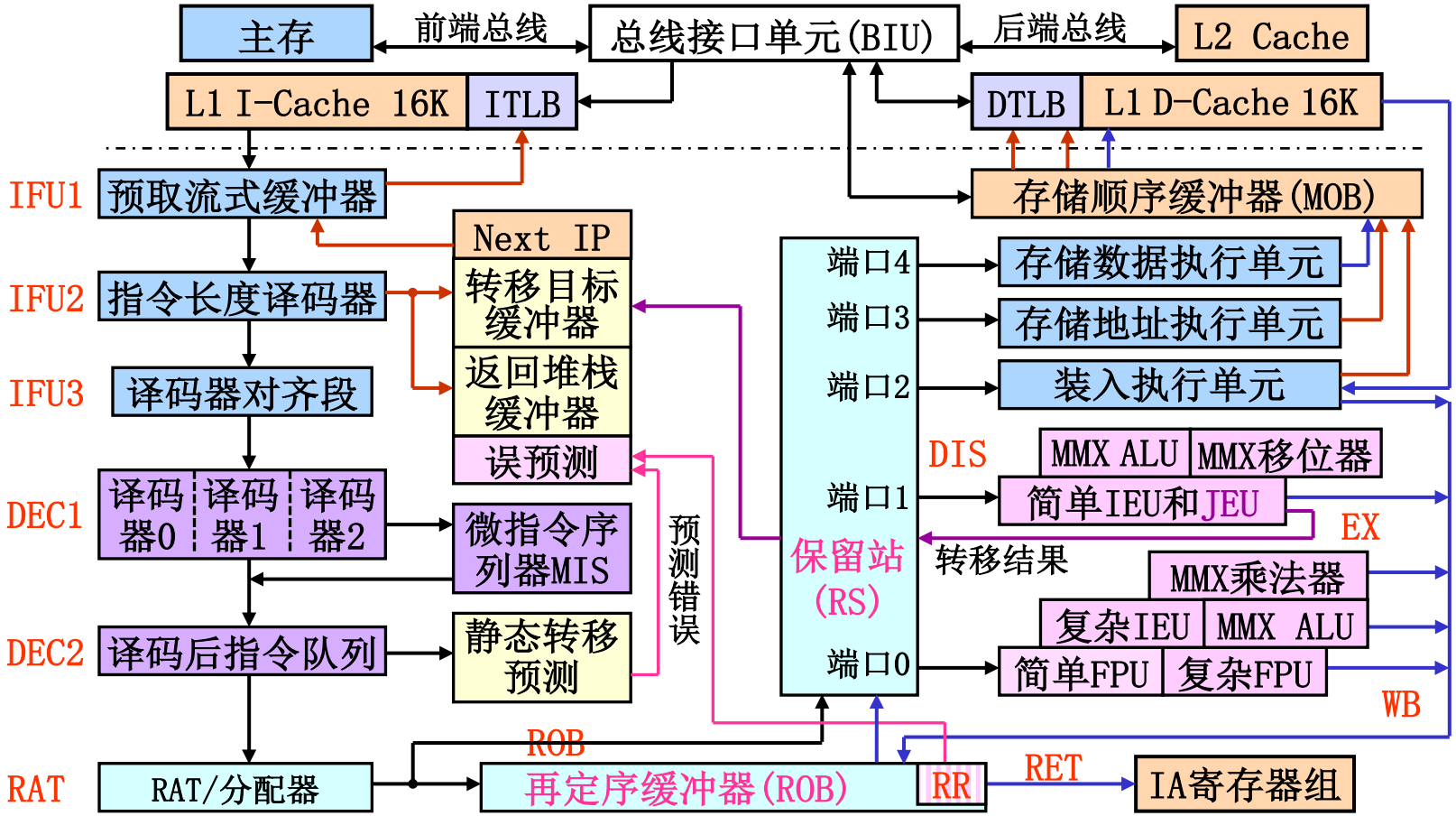
- **奔腾II〔Pentium II〕**
  - PII为英特尔于1997年5月7日推出的微处理器
  - 它基于Pentium Pro使用的P6架构，加入MMX (多媒体) 指令集

- **初代Pentium II**
  - 233MHz、266MHz、300MHz
  - 350nm制程
  - 继承了Pentium Pro的动态执行技术



https://cpu.zol.com.cn/259/2591441.html

示例2：Pentium II CPU动态执行技术(了解)



动态分支预测＋动态调度＋推测执行

思考：流水线如何处理访存冲突的？

L1$哈弗结构、增设MOB、增设指令缓冲器

流水线组成—12级





IF&ID段—实现取指&译码，形成μOP序列(微指令序列)　　(稍后讲)

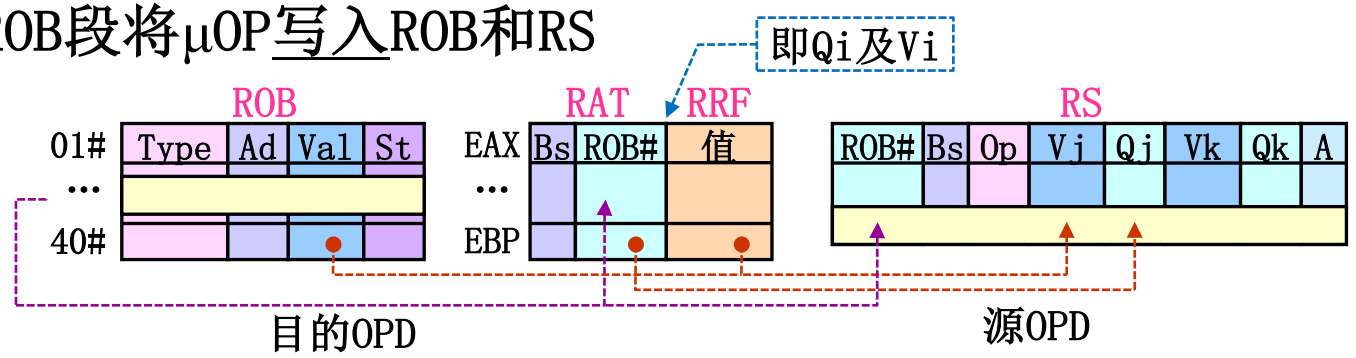
CISC→RISC→⊥←执行基于微指令

动态分支预测—IF段动态预测(BTB等)、ID段静态预测(BTB缺失时)，

EX段更新转移历史，ID&RR段处理预测结果

指令发射—RAT段分配ROB及RS行、实现REG重命名(消除WAR/WAW)，

ROB段将μOP写入ROB和RS



RAT—Register Alias Table
DIS—Dispatch
RRF—Retirement Register File，即GPRs

动态调度—DIS段从RS中输出就绪的μOP(处理RAW)　←Intel称为派遣

推测执行—按动态分支预测结果取指，按动态调度结果执行；

RR段按序确认、处理预测/异常，

RET段写结果(REG/MEM)

} Tomasulo仅1个段

RR—Retirement Ready
RET—Retirement



Part I. 为什么要学习指令级并行?
Part II. 什么是指令级并行?
Part III. 如何开发指令级并行? —数据相关动态调度
Part IV. 如何开发指令级并行? —动态分支预测
Part V. 前瞻执行技术
Part VI. 多指令流出技术

多指令流出技术（→CPI<1）

*多流出流水线：一个时钟周期可完成多条指令　　←需执行多条指令



发射与流出— 同义(Issue)，指ID段→EX段过程

IS段的任务— 指令打包、冒险处理(含数据/控制冒险)

　　└←有静态/动态2种

　　├←①组织发射包(可同时执行的指令集合)

　　└←②送入发射槽(即各执行部件的指令接收器)

*多流出处理器：　(基于数据冒险处理策略)　　VLIW—Very Long Instruction Word

静态多发射— 软件/硬件打包指令、静态调度处理冒险

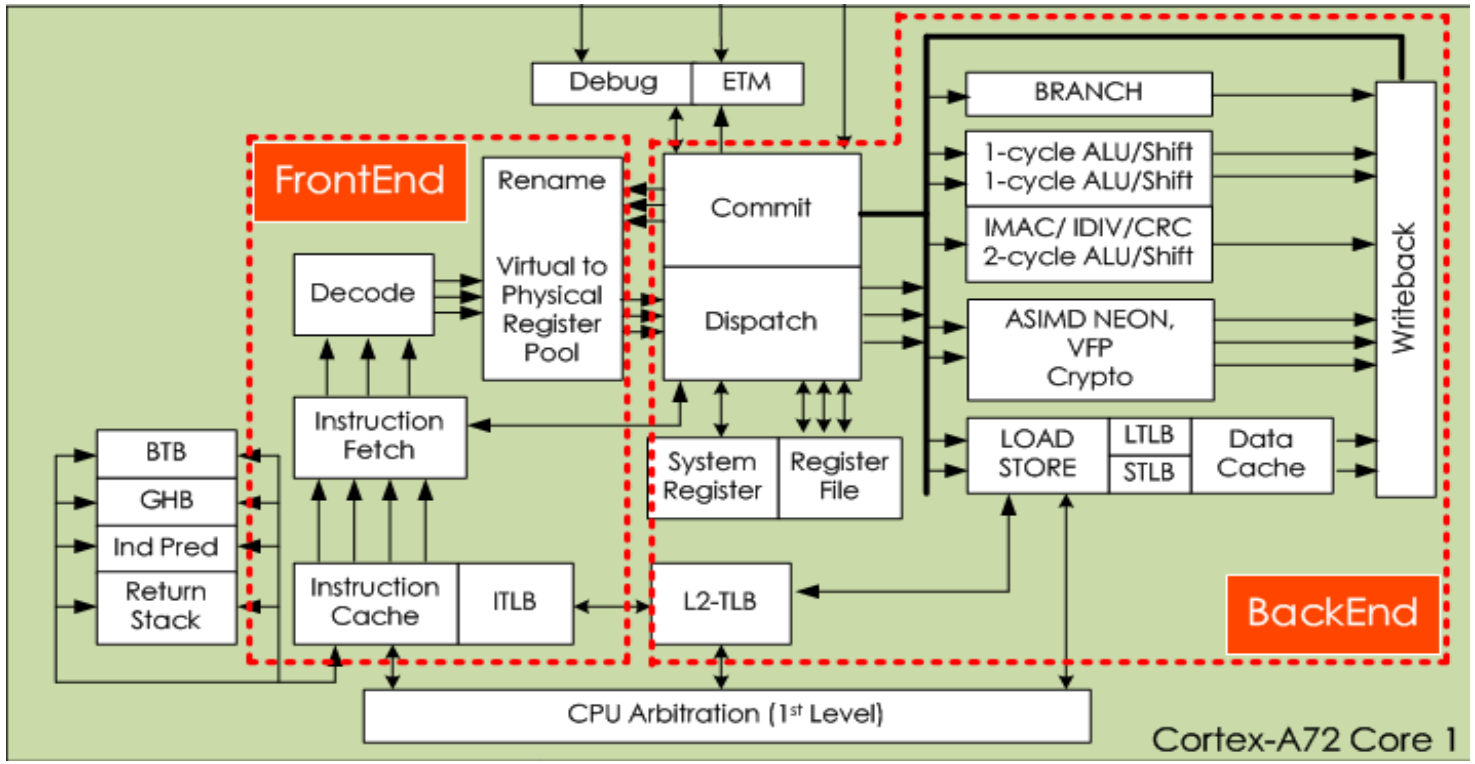
　　└←m固定/可变　　└←如VLIW/静态超标量

动态多发射— 硬件打包指令、动态调度处理冒险

　　└←m可变　　└←如动态超标量

鲲鹏CPU的超标量流水技术（了解）

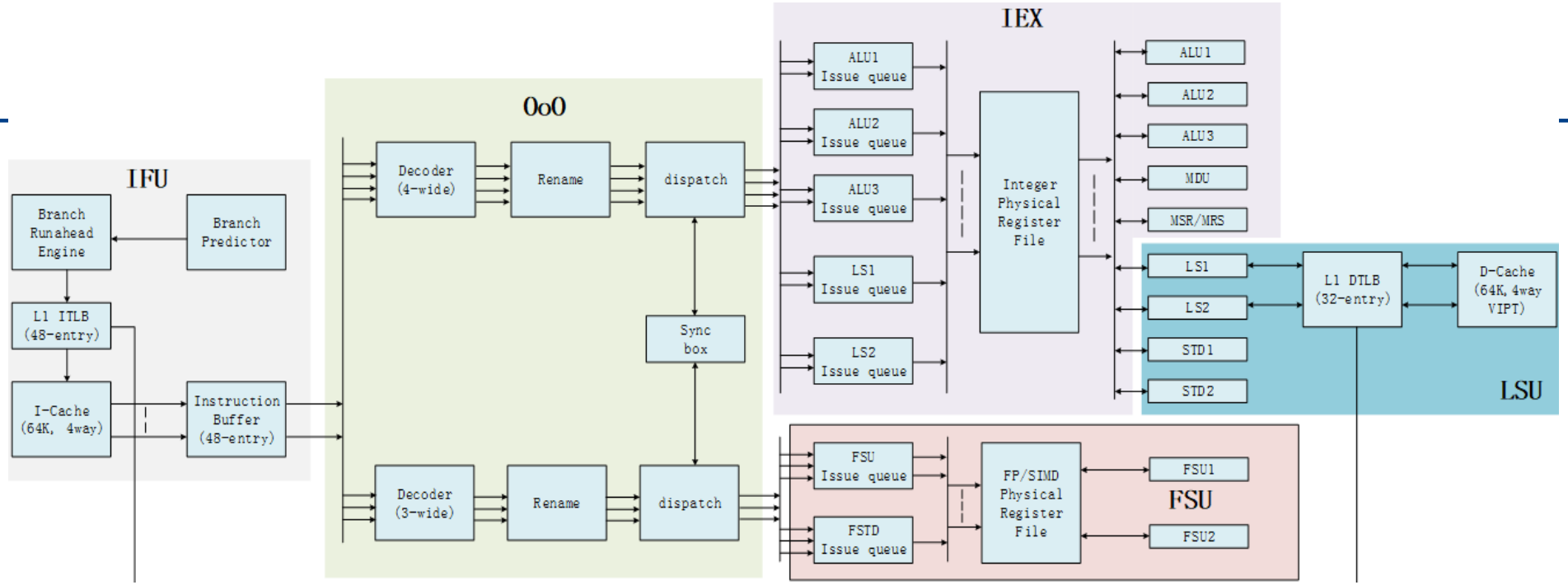
超标量特征—动态调度，IPC＝4条指令，流水线分前后端



流水线组成—

Fetch	Decode	Allocation	Issue	Execute	WriteBack	Commit

IFU0-IFU4	D1-D3	S1	P1	I1-I2	EX1-EX5	WB	IC	整数流水线
	D1-D4	S1	P1	I1-I2	EX1-EX7	WB	IC	浮点流水线

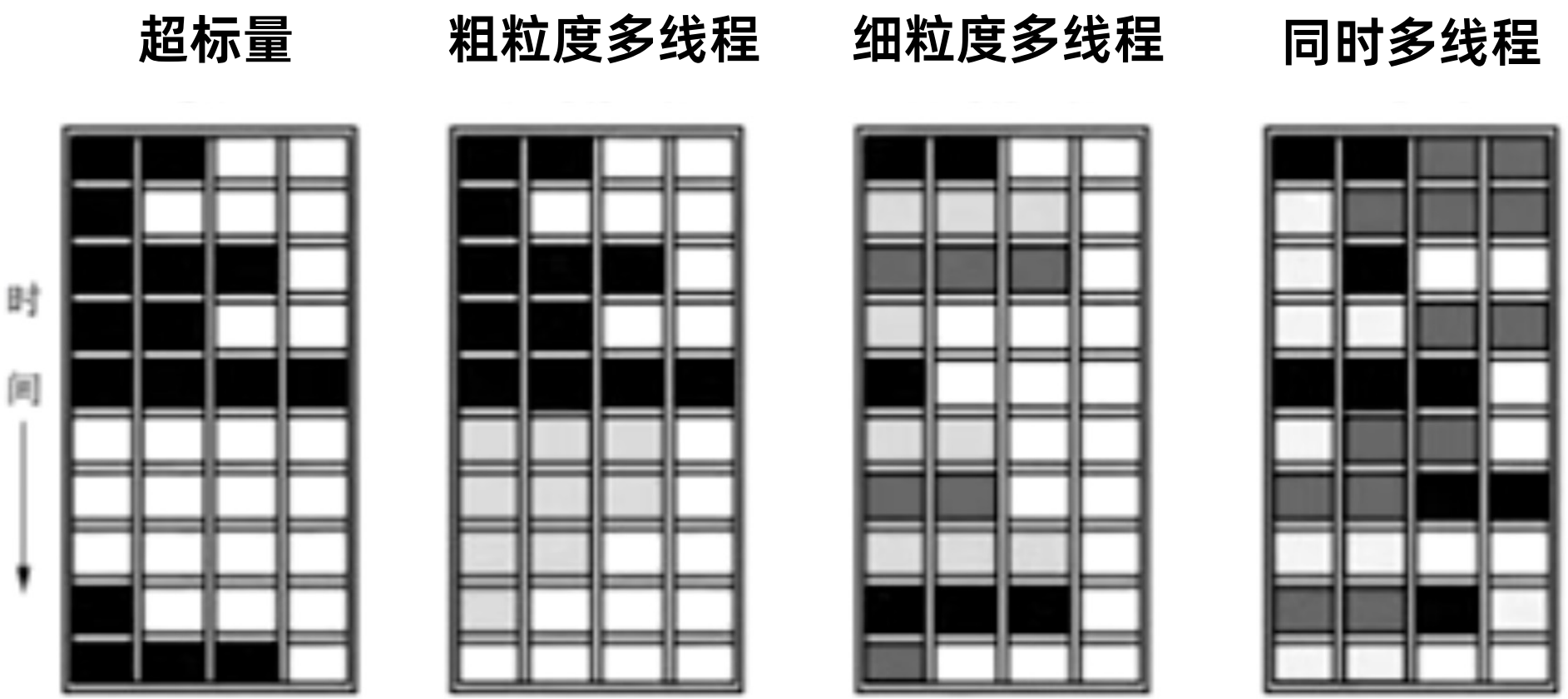


Fetch：取指令并产生下次Fetch地址，包括指令缓存、分支预测、BTB及RSB等
Decode： - 分解指令流到独立指令;
- 转换X86指令到类RISC uOp序列;
- 理解指令语义，包括指令类型(运算/访存/控制等)，OP类型、OPD地址等
Allocation：寄存器重命名+资源预留
Issue：发射指令到相应执行单元，从此处进入乱序执行阶段
Execute：指令执行阶段
Write Back：将执行结果写入ROB等
Commit：重整执行结果次序、确定推测执行的正确性，最后输出结果



Part I. 为什么要学习指令级并行?
Part II. 什么是指令级并行?
Part III. 如何开发指令级并行? —数据相关动态调度
Part IV. 如何开发指令级并行? —动态分支预测
Part V. 前瞻执行技术
Part VI. 多指令流出技术
Part VII. 多线程技术

将线程级并行TLP转换为指令级并行ILP



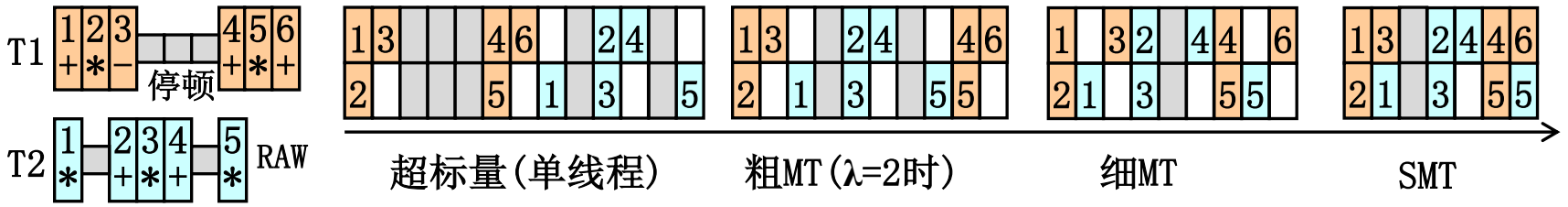
多线程的实现方法

*粗粒度MT： 线程有较长停顿(λ)时切换

特点—线程速度可保证，吞吐率提高有限(<λ时无效果)

*细粒度MT： 线程每个时钟周期切换 (时间片轮转法)

特点—吞吐率提高(停顿被隐藏)，线程速度变慢(轮转所致)



思考：粗MT、细MT的线程数一般是多少？说明理由

*同时MT(SMT)：细粒度MT的变体，可同时执行不同线程的指令

(又称超线程HT) 无需每个CLK切换←→部件不冲突时

特点—吞吐率提高幅度大(TLP转换为ILP)

线程切换策略—优先线程方法(1主/1辅) ←保证速度&提高利用率

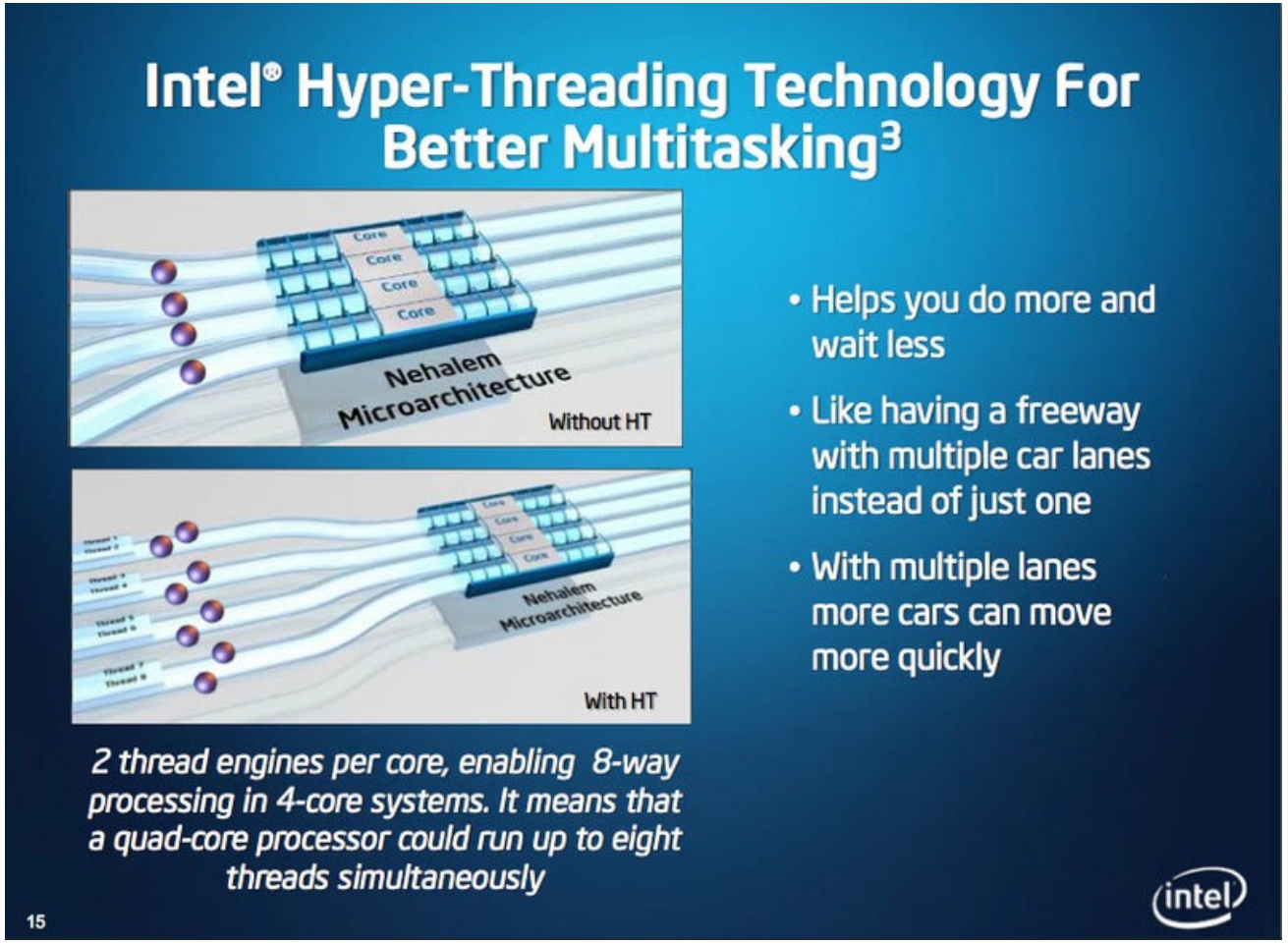
Intel的超线程技术

- 5%核心面积→15~30%的多线程性能

13代酷睿配置表

处理器型号	处理器内核(P 核+E 核)	处理器线程	英特尔® 智能高速缓存(L3)	处理器睿频频率			处理器基本频率		处理器显卡	CPU PCIe 通道总数
				英特尔®温度自适应睿频加速技术(英特尔® TVB)	英特尔® 睿频加速Max技术3.0频率	单 P 核/E 核睿频频率(GHz)	单 P 核/E 核基本频率(GHz)	未锁频		
i9-13900K	24 (8+16)	32	36 MB	高达 5.8	高达 5.7	高达 5.4/4.3	3.0/2.2	√	英特尔® 超核芯显卡770	20
i9-13900KF	24 (8+16)	32	36 MB	高达 5.8	高达 5.7	高达 5.4/4.3	3.0/2.2	√	不适用	20
i9-13900	24 (8+16)	32	36 MB	高达 5.6	高达 5.5	高达 5.2/4.2	2.0/1.5	-	英特尔® 超核芯显卡770	20
i9-13900F	24 (8+16)	32	36 MB	高达 5.6	高达 5.5	高达 5.2/4.2	2.0/1.5	-	不适用	20
i7-13700K	16 (8+8)	24	30 MB	-	高达 5.4	高达 5.3/4.2	3.4/2.5	√	英特尔® 超核芯显卡770	20
i7-13700KF	16 (8+8)	24	30 MB	-	高达 5.4	高达 5.3/4.2	3.4/2.5	√	不适用	20
i7-13700	16 (8+8)	24	30 MB	-	高达 5.2	高达 5.1/4.1	2.1/1.5	-	英特尔® 超核芯显卡770	20
i7-13700F	16 (8+8)	24	30 MB	-	高达 5.2	高达 5.1/4.1	2.1/1.5	-	不适用	20
i5-13600K	14 (6+8)	20	24 MB	-	-	高达 5.1/3.9	3.5/2.6	√	英特尔® 超核芯显卡770	20
i5-13600KF	14 (6+8)	20	24 MB	-	-	高达 5.1/3.9	3.5/2.6	√	不适用	20
i5-13400	10 (6+4)	16	20 MB	-	-	高达 4.6/3.3	2.5/1.8	-	英特尔® 超核芯显卡730	20
i5-13400F	10 (6+4)	16	20 MB	-	-	高达 4.6/3.3	2.5/1.8	-	不适用	20
i3-13100	4	8	12MB	-	-	高达 4.5/0	3.4/0	-	英特尔® 超核芯显卡730	20
i3-13100F	4	8	12MB	-	-	高达 4.5/0	3.4/0	-	不适用	20

F 系列	KF 系列	K 系列	KS 系列	无后缀
无核显 价更优 需配独显	无核显 可超频 需配独显 和散热器	支持超频 需配散热器	特别版 轻松超频 需配散热器	有核显 不可超频



https://orbitalcomputers.com/contents.php?page=hyperthreading

作业

- **PPT思考题**
  - 2、3、5

第4章思考题

- 1、动态调度的基本原理？Tomasulo算法的实现原理(指结构组织、冒险处理等方面)？
- 2、假设动态调度流水线的保留站结构如图5.7所示(或PPT的RS/FLB/SDB)，LD、ADD/SUB、MUL/DIV指令的操作延迟分别为3、2、5个CLK，写出下列指令序列中LD指令写结果完成时的保留站状态，计算序列的执行时间。
- I1: F1=F2+5　　I4: F1=F2/2
- I2: F3=M[F2]　　I5: F3=F1/4
- I3: F4=F1*F3
- 3、采用BTB进行动态分支预测后，流水线的IF、ID、EX段所进行的相关操作有哪些？
- 4、某32位CPU的BTB有512行，采用8路组相联映射、LRU替换算法组织，采用2级相关预测器预测，其中全局历史为3位、模式历史为2位，则BTB容量为多少？
- 5、推测执行的基本思想？相对于Tomasulo算法流水线，结构上有何变化？