Ch3--
ILP及其
探索

Ch3-1
•指令级并行

•暴露ILP的基础编译器技术

•动态调度——记分板

3.1, 3.2, 附录C7

什么是指令级并行?

□指令级并行

>指令间潜在的重叠执行能力

□基本块的指令级并行度非常有限

> 基本块：一种仅含单入口单出口的直线型代码序列，内部无分支结构

>平均动态分支频率15%至25% $\Rightarrow$ 4 每对分支间执行约7条指令

> 基本块内指令往往存在数据依赖

流水线回顾

开发指令级并行时的目标

是最大化CPI

□ 流水线 CPI = 理想流水线 CPI+ 结构停顿+数据冲突停顿+控制停顿

理想流水线CPI：衡量该实现可达到的最大性能

> 结构冲突：硬件不支持该指令组合

> 数据冒险：指令依赖于仍在流水线中的前一条指令结果

> 控制冒险：由指令取指与控制流变更决策（分支和跳转）之间的延迟引起

如何开发指令级并行?

□两大主流方法：

>基于硬件的动态方案-广泛应用于服务器/桌面处理器-在便携媒体处理器中应用有限

基于编译器的静态方法

•在科学计算领域之外成效有限

第三章

□指令级并行：概念与挑战

□开发ILP的基础编译器技术□通过动态调度克服数据冒险□利用动态分支预测降低分支成本 □ 基于硬件的推测执行

□多发射与静态调度开发指令级并行

□利用动态调度、多发射和推测开发指令级并行性

□ 高级指令交付与推测技术

□多线程：通过开发线程级并行性提升单处理器吞吐量

减少停滞的构想

ChC

Ch3

ChH

技术手段	降低	章节
转发与旁路	潜在数据冲突停顿	C.2,C.3
简单分支调度与预测	控制冲突停顿	C.3
基础编译器流水线调度	数据冲突停顿	C.2, 3.2
循环展开	控制冒险停顿	3.2
动态分支预测	控制停顿	3.3
动态调度（记分板）	数据冒险停顿	C.7
动态调度（Tomasulo算法）	由反相关和输出相关导致的停顿	3.4, 3.5
基于硬件的推测执行	控制停顿	3.6
每周期发射多条指令	理想CPI	3.7
动态调度+多指令发射+推测执行	数据与控制停顿	3.8
多线程	数据并行	3.11
编译器依赖分析、软件流水、踪迹调度	理想CPI与数据冒险停顿	H.2, H.3
编译器推测的硬件支持	理想CPI与数据冒险停顿、分支冒险停顿	H.4, H.5

指令级并行性(ILP)

□要获得显著的性能提升，必须在多个基本块间开发ILP

□最简方法：利用循环迭代间的并行性开发循环级并行

> 向量化与GPU加速是一种途径

> 若非向量化，则需通过分支预测实现动态并行或依赖编译器循环展开实现静态并行

数据相关性与冒险

□相关性是程序的固有属性，存在相关性即表明潜在冒险可能

流水线结构决定是否检测到相关性及是否引发停顿，实际冒险与停顿时长取决于流水线特性

□ 数据依赖性表示：

> 潜在冲突可能性（寄存器与内存位置）

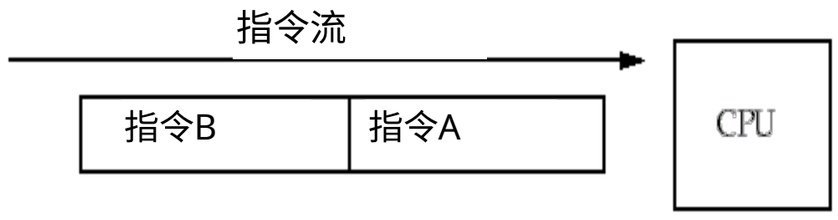
> 计算结果必须遵循的执行顺序

> 可开发指令级并行性的上限

□ 经由内存位置传递的依赖关系难以检测

回顾：数据冲突类型

□考虑两条指令A和B，A在B之前执行



□ RAW（写后读）真数据相关 > 指令A写入Rx，指令B读取Rx □ WAW（写后写）输出相关 > 指令A写入Rx，指令B写入Rx □ WAR（读后写）反相关 > 指令A读取Rx，指令B写入Rx

□ 流水线必须保持的排序决定了危害的命名

真实数据依赖与危害

□真实数据依赖：

> Instr_1 对 Instr_1 存在数据依赖
> Instr_J 试图在 Instr_J 写入操作数前读取它

I: 加法指令 r1,r2,r3
J: 寄存器r4等于r1减r3

> 或 Instr_J 数据依赖于 Instr_K ，后者又依赖于指令

□由"真依赖"引起（编译器术语）

□若真依赖导致流水线冲突，称为写后读(RAW)冲突

名称依赖1：反依赖

□ 名称依赖：当两条指令使用相同寄存器或内存位置（称为名称）时，但指令间不存在与该名称相关的数据流

□指令」在 Instr_1 读取操作数前写入该操作数

I：sub r4,r1,r3
J：add r1,r2,r3
K：mul r6,r1,r7

编译器开发者称之为"反依赖"，源于对寄存器"r1"名称的重复使用

□若反依赖导致流水线冲突，则称为读后写(WAR)冲突

名称依赖 2：输出依赖

□指令」在另一指令写入操作数前先写入该操作数。

I：sub r1,r4,r3
J：add r1,r2,r3
K：mul r6,r1,r7

□编译器开发者称之为"输出依赖" 该现象也源于重复使用名称"r1"

□若反依赖导致流水线冲突 则称为写后写(WAW)冲突

名称依赖

□ 两条指令使用相同名称但不存在信息流

> 非真实数据依赖，但指令重排时会产生问题

> 反依赖：指令 j 写入指令 i 所读取的寄存器或内存位置

•必须保持初始顺序（i在j之前）

> 输出依赖：指令 i 与指令 j 写入相同寄存器或内存位置-必须保持执行顺序

指令级并行与数据冲突

□硬件/软件必须保持程序顺序：即指令在原始源代码程序中确定的单步顺序执行方式

□$\mathrm{HW/SW}$ 目标：仅在影响程序结果的情况下保持程序顺序 以开发并行性

□ 若修改指令中使用的名称以避免冲突 则涉及名称依赖的指令可并行执行

> 寄存器重命名通过编译器或硬件解决寄存器名称依赖问题 >

控制依赖

□每条指令都控制依赖于某些分支集，通常必须保持这些控制依赖才能维持程序顺序

```
若p1成立 {
    S1;
};
若p2成立 {
    S2;
}
```

□$S1$ 控制依赖于 $p1$ ，且 $S2$ 控制依赖于 p2 而非 p1。

控制依赖可忽略

□无需保持控制依赖 > 若不影响程序正确性，允许执行本不该执行的指令，从而违反控制依赖关系

□程序正确性的两个关键属性实为异常行为与数据流

示例

•示例1：

- 加法指令 x1,x2,x3
- 条件分支指令 x4,x0,L
- sub x1,x1,x6
- L: …
- or x7,x1,x8

□ 或取决于指令

加法与减法

- 示例2：
- add x1,x2,x3
- 若x12等于x0，则跳转至skip
- x4寄存器值等于x5减x6
- x5寄存器值等于x4加x9
- skip标记：
- 或 x7,x8,x9

□假设x4后续未被使用

跳过

＞ 可尝试将减法指令前移分支指令之前

异常行为

□保持异常行为

⇒ 指令执行顺序的任何变更均不得改变程序中异常触发方式(=>不产生新异常)>示例:DADDU R2,R3,R4BEQZ R2,L1LW R1,0(R2)L1: ……

□将LW指令移至BEQZ前的问题?

简要概述

OILP

＞ 指令间潜在重叠

□减少由以下原因导致的停顿

＞ 结构冲突

＞ 数据冲突

＞ 控制冒险

为保持程序正确性，必须

＞ 保持数据流

＞ 保持异常行为

ILP专题讲座：软件实现方法

□开发ILP的基础编译技术 ＞ 循环展开

□静态分支预测

□静态多指令发射：超长指令字

□高级编译器支持：揭示与利用指令级并行 ＞ 软件流水线>全局代码调度□硬件支持：在编译时暴露更多并行性

＞ 条件或谓词指令

＞ 基于硬件支持的编译器推测执行

浮点循环：何处存在数据冲突?

循环：LD F0,0(R1) ;F0=向量元素

ADDD F4,F0,F2 ;与F2标量相加

SD 0(R1),F4 ;存储结果

SUBI R1,R1,8 ;指针递减8字节(双字)

BNEZ R1,Loop ;若R1非零则跳转

NOP ;延迟分支槽

指令生成结果	指令使用结果	执行周期数	延迟周期数
浮点ALU操作	另一浮点ALU操作	**4**	**3**
浮点ALU操作	存储双精度	**3**	**2**
加载双精度	浮点运算操作	**1**	**1**
加载双精度	存储双精度	**1**	**0**
整数运算	整数运算	**1**	**0**

摊位在哪里?

延迟规范

□算术逻辑单元 F1,-,-: 取指 译码 取数 取数 取数 取数

□算术逻辑单元 -, F1,-: 取指 译码 s s S 取数 取数 取数

□算术逻辑单元: 取指 译码 取数 取数 取数 取数 回写□存储指令: 取指 译码 s s 执行 数据存储器

□取数指令 F1, - 取指 译码 执行 数据存储器 回写□存储指令: F1,8(R1): 取指 译码 执行 数据存储器 回写存储器/回写.加载存储器 - ⟶ 数据存储器输入端口

减少调度BB和延迟分支导致的停顿

循环：加载 F0, 0(R1)

浮点加 F4, F0, F2

将浮点寄存器F4存入内存(R1+0)

寄存器R1加立即数8

若R1非零则跳转至Loop

FDXMW

$FD\dot{S}A_1A_2A_3A_4W$

$F \le D \le S^* \times MW$

F s s D XMW

F s D X M

W6 CC F S D X M W10 CC F F 1 .23

循环：加载 F0, 0(R1)

寄存器R1减立即数8

浮点加法 F4=F0+F2

若R1非零则跳转至Loop

标准偏差 $+8\,(R1), F4$

F D X M W

F D X M W

$F_1 D A_1\ A_2\ A_3\ A_4\ W$

FDXMW

FDsXMW

循环展开四次(直接方式)

1 循环:	LD	F0,0(R1)	1 周期停顿	重写循环至最小化停顿?
2	ADDD	F4,F0,F2	2周期停顿	
3	SD	0(R1),F4	移除SUBI与BNEZ指令	
4	LD	F6,-8(R1)		
5	ADDD	F8,F6,F2		
6	SD	-8(R1),F8	移除SUBI与BNEZ指令	
7	LD	F10,-16(R1)		
8	ADDD	F12,F10,F2		
9	SD	-16(R1),F12		
10	LD	F14,-24(R1)	1周期停顿	
11	ADDD	F16,F14,F2	1周期停顿(等待F16)	
12	SUBI	R1,R1,#32	修改为 $4*8$	
13	SD	+8(R1),F16		
14	BNEZ	R1,LOOP	1周期控制停顿	
15	NOP			

$14 + 3x\,(1+2) + 1 + 1 + 1 = 26$ 时钟周期，或每次迭代6.5周期假设R1是4的倍数

最小化停顿的循环展开

```
1 循环:   LD     F0,0(R1)
2         LD     F6,-8(R1)
3         LD     F10,-16(R1)
4         LD     F14,-24(R1)
5         ADDD   F4,F0,F2
6         ADDD   F8,F6,F2
7         ADDD   F12,F10,F2
8         ADDD   F16,F14,F2
9         SD     0(R1),F4
10        SD     -8(R1),F8
11        SUBI   R1,R1,#32
12        SD     +16(R1),F12
13        BNEZ   R1,LOOP
14        SD     8(R1),F16      ; 8-32 = -24
```

14个时钟周期，或每次迭代3.5周期

□移动代码时做了哪些假设？ ＞即使修改了寄存器，将存储移到SUBI后仍可行 ＞将加载移到存储前是否安全：能否获取正确数据？ ＞编译器何时可安全执行此类变更?

减少停顿的设计思路

	技术手段	减少	章节
ChC	转发与旁路	潜在数据冲突停顿	C.2,C.3
	简单分支调度与预测	控制冲突停顿	C.3
Ch3	基础编译器流水线调度	数据冲突停顿	C.2, 3.2
	循环展开	控制冒险停顿	3.2
	动态分支预测	控制停顿	3.3
	动态调度（记分板）	数据冒险停顿	C.7
	动态调度（Tomasulo算法）	由反相关和输出相关导致的DH停顿	3.4, 3.5
	基于硬件的推测执行	控制停顿	3.6
	每周期发射多条指令	理想CPI	3.7
	动态调度+多指令发射+推测执行	数据与控制停顿	3.8
ChH	多线程	数据并行	3.11
	编译器依赖分析、软件流水、踪迹调度	理想CPI与数据冒险停顿	H.2, H.3
	编译器推测的硬件支持	理想CPI与数据冒险停顿、分支冒险停顿	H.4, H.5

为何需要动态调度?

□ **示例1：**

＞ DIVD $F0, F2, F4$
ADDD F10,F0,F8SUBD
F12,F8,F14

□示例2：结构冲突DIVD F2,F2,F4

```
ADDD F10,F0,F8          ; 非流水线浮点加法器
加法指令 F12, F0,F4
乘法指令 F16, F14,
F4
```

•问题：指令(减法指令、乘法指令)因无关前序指令而阻塞

硬件方案：动态调度

□ 核心思想：允许停顿后指令继续执行。重排指令顺序以减少停顿，同时保持数据流

□支持乱序执行
并允许乱序完成

□需区分指令开始执行与完成执行的两个时间点，其间为指令执行阶段

□在动态调度流水线中，所有指令按序通过发射阶段（顺序发射）

动态调度的优势

□ 处理编译时依赖关系未知的情况
＞(例如涉及内存引用时)

□简化编译器设计，编译器无需具备微架构知识

- Allows code that compiled for one pipeline to run efficiently on a different pipeline
- Hardware speculation, a technique with significant performance advantages, that builds on dynamic scheduling

动态调度第一步

□简单流水线用1个阶段（指令译码ID，亦称指令发射）同时检测结构冲突与数据冲突

□将简单五级流水线的ID段拆分为两个阶段：

□问题解码指令，检查结构风险

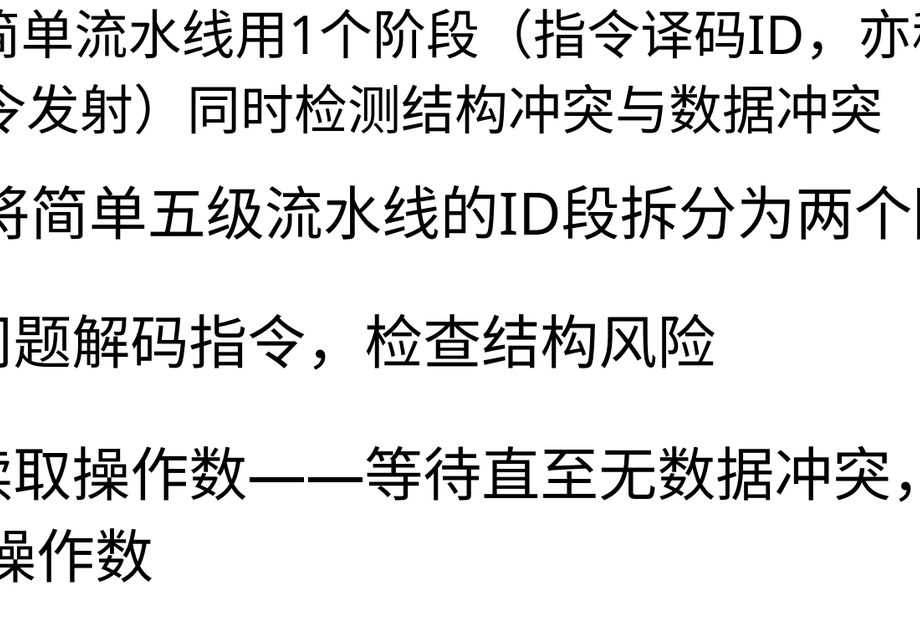
□读取操作数——等待直至无数据冲突，再读取操作数

采用计分板的动态调度

□ 计分板技术
>得名于CDC6600计分板
＞ 当资源充足且无数据依赖时，允许指令乱序执行。

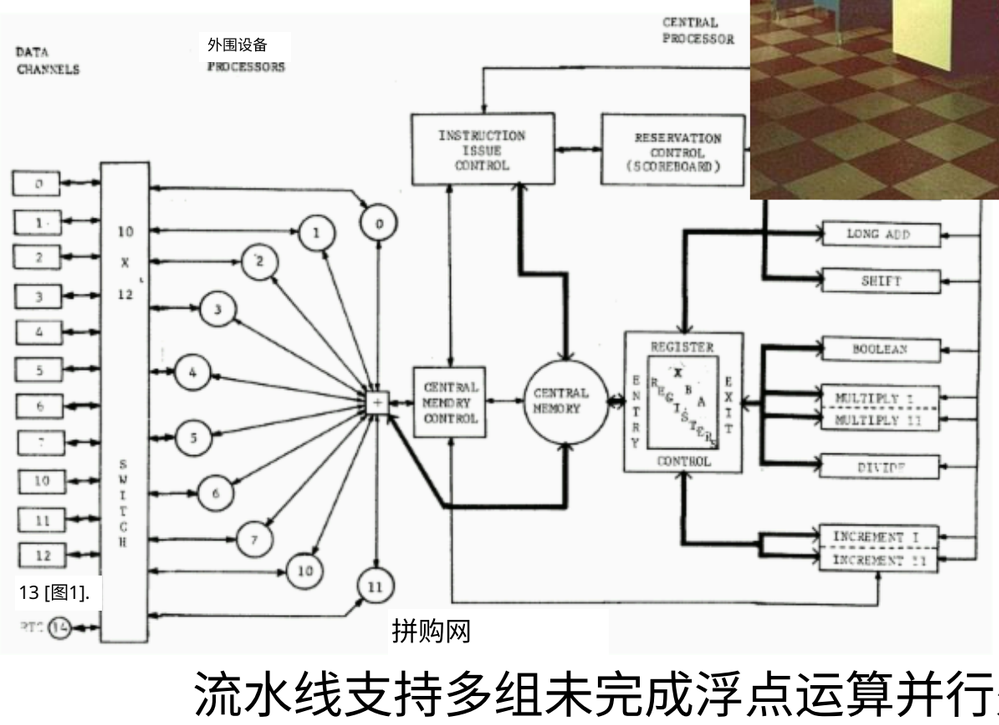
顺序发射
＞ 乱序完成
>尽早执行指令

配备计分板的流水线处理器基本结构



CDC6600 - 首台超级计算机

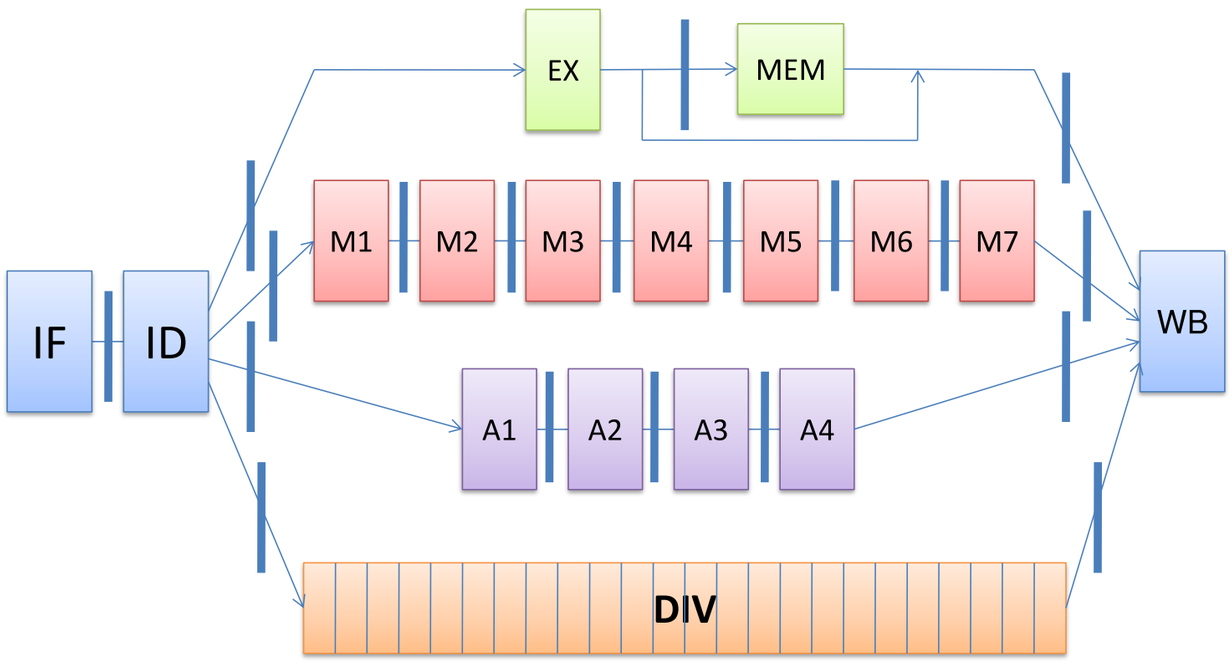
1964-1969年性能榜首



13 [图1].

拼购网

流水线支持多组未完成浮点运算并行处理



带记分牌的流水线阶段

五个阶段：取指、译码、执行、访存、写回

取指：所有指令相同

译码：拆分为两个阶段：发射和读取操作数

执行：无变化

>MEM：省略了仅针对JFJPEM操作的foxronly集中离子

>WB：无变化

□流水线阶段分为：取指(IF)、发射(IS)、读操作数(RO)、执行(EX)、写回(WB)

记分牌流水线阶段说明

□ 发射阶段：当 > 功能单元可用且

> 无其他活跃指令占用相同目标寄存器时，指令被发射。 > 避免结构冲突和写后写冲突□读操作数(RO) > 读取操作将延迟至所有操作数就绪。 > 这意味着先前发射但未完成的指令不会将该操作数作为目标。 > 动态解决读后写冲突□执行(EX) > 执行完成后通知记分牌以释放功能单元□写回(WB) > 记分牌检查写后读冲突，必要时暂停完成指令

计分板算法

□计分板全权负责指令派发与执行 > 建立依赖记录 > 决定操作数获取时机

> 决定执行启动时机

> 决定结果写入寄存器文件的时机□三大数据结构

> 指令状态：

•指令当前所处的四阶段状态

功能单元状态：忙碌、操作中、Fi、Fj、Fk、Qj、Qk、Rj、Rk > 寄存器结果状态：标记由哪个功能单元写入该寄存器

示例：指令状态

```
LD      F6, 34(R2)
LD      F2, 45(R3)
MULTD   F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

指令	指令状态			
	IS	RO	EX	WB
LD	✓	✓	✓	✓
LD	✓	✓	✓	
MULTD	✓			
SUBD	✓			
DIVD	✓			
ADDD				

记分板示例

指令状态：

指令		j	k	问题	Oprand	Comp	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				

读取 Exec 写入

Function Unit Statous:

时间	名称	忙碌	Op	Fi	Fj	Fk	Qi	Qk	Rj	Rk
时钟周期 计数器	整数	No								
	乘法器1	No								
	乘法器2	No								
	加法	No								
	除法	No								

描述 S1 S2 RS RS

寄存器结果状态：

时钟		F0	F2	F4	F6	F8	F10	F12	...
0	FU								

记分板周期1

指令状态：

指令		j	k	问题	操作数比较结果
LD	F6	34+	R2	✓	
LD	F2	45+	R3		
MULTD	F0	F2	F4		
SUBD	F8	F6	F2		
DIVD	F10	F0	F6		
ADDD	F6	F8	F2		

读取 执行 写入

Function Unit Statous:

时间	名称	忙碌	Op	des	S1 Fj	S2 Fk	RS Qj	RS Qk	Rj	Rk
时钟周期 计数器	整数	是	加载	F6	R2				是	
	乘法器1	No								
	乘法器2	No								
	加法器	No								
	除法	No								

F3(寄存器结果状态：

时钟		F0	F2	F4	F6	F8	F10	F12	...	F3(
0	FU				中断					

计分板周期2

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 Exec 写入

发射 操作数比较R esult

√

Function Unit Statous:

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Clock cycle	整数	是	加载	F6	R2				No	
计数器	多选1	No								
	乘法器2	No								
	加法器	No								
	除法器	No								

描述 S1 S2 RS RS

寄存器状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU			整数					

计分板周期3

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 执行 写入

问题 Oprand Comp Result

√

计算地址

功能单元状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
时钟周期	整数	是	加载	F6	R2				No	
计数器	乘法1	No								
	乘法2	No								
	加法	No								
	除法	No								

描述 S1 S2 RS RS

寄存器状态结果：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU			整数					

计分板周期4

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 执行 写入

问题 操作数比较结果

✓

访问 数据 缓存

功能单元状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
时钟周期	整数	是	加载	F6	R2				No	
计数器	乘法器1	No								
	乘法2	No								
	加法	No								
	除法	No								

描述 S1 S2 RS RS

寄存器状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU			整数					

计分板周期5

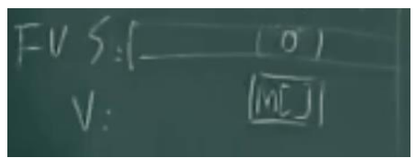
指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 Exec Write

问题 操作数比较结果

√



Function Unit Statous:

时间	名称	忙碌	Op	Vi	Vj	Vk	Qj	Qk	Rj	Rk
时钟周期	整数	No	负载	F6	V_{R2}				No	
计数器	乘法器1	No								
	乘法器2	No								
	加法	No								
	除法	No								

描述 S1 S2 RS RS

寄存器结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU	M[R2+34]							

计分板周期6

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 Exec 写入

发射 操作数比较结果 sult

√

功能单元状态：

时间	名称	忙碌	Op	Vi	Vj	Vk	Qj	Qk	Rj	Rk
时钟周期	整数	是	加载	F2	R3				是	
counter	乘法器1	No								
	乘法2	No								
	加法	No								
	除法	No								

描述 S1 S2 RS RS

寄存器状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU		整数	M[R2+34]					

计分板周期7

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 执行 写入

发射 操作数比较结果

√

√

功能单元 Statous:

时间	Name	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Intege	Yes	load	F2	R3				No	
	多路1	是	乘法	F0	F2	F4	整数		No	是
	乘2	No								
	加法	No								
	除法	No								

设计 S1 S2 RS RS

寄存器结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F3
0	FU	乘法器I	整数	M[R2+34]					

计分板周期8

指令状态：

Instruction		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 Exec 写入

Issue 操作数比较结果

√

√

√

计算地址

功能 Unit 状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
时钟周期计数器	整数	是	加载	F2	R3				No	
	乘法器1	是	乘	F0	F2	F4	整型		No	是
	乘法器2	No								
	加	是	减	F8	F6	F2		整型	是	No
	除法	No								

描述 S1 S2 RS RS

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU	乘法器1	中断47	M[R2+3	加法				

记分板周期9

指令状态：

指令		j	k
LD	F6	34+	R2
LD	F2	45+	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

读取 执行 写入

发射 运算与比较结果 ult

✓

✓

√

√

访问数据缓存

函数 Unit 状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
时钟周期	整数	是	加载	F2	R3				No	
计数器	乘法器1	是	乘法	F0	F2	F4	整数		No	是
	乘法器2	No								
	加法	是	减法	F8	F6	F2		整数	是	No
	除法	是	除	F10	F0	F6	乘1		No	是

描述 S1 S2 RS RS

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU	乘法I	整型	M[R2+3	加法	除法			

示例：功能单元状态及寄存器状态

名称	功能单元状态								
	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
整数	是	加载	F2	F3				No	
乘法单元1	是	多	F0	F2	F4	整数		No	是
多2	No								
加	是	子项	F8	F6	F2		整数	是	No
除	是	除	F10	F0	F6	乘1		No	是

	注册结果状态								
	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	乘法	整数			加法	除法		…	

记分板周期10

指令状态：

指令		j	k	发射	操作数比较	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			

Function Unit 状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
时钟周期计数器	乘法器1	是	多	F0	F2	F4	整		是	是
	多2	No								
	添加	是	减法	F8	F6	F2		整数	是	是
	除	是	除法	F10	F0	F6	乘1		No	是

寄存器状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	Mult1 /(R3+45)			M[R2+34	Add	除法		

记分板周期11

指令状态：

指令		j	k	发射	Oprand	Comp	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				

功能单位状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
时钟周期计数器	乘法器1	是	乘	F0	F2	F4			No	No
	乘2	No								
	加	是	减	F8	F6	F2			No	No
	除	是	除法	F10	F0	F6	乘法1		No	是

寄存器结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	Mult1 M[R3+45]		M[R2+34	加法	除法			

Scoreboard Cycle 12

指令状态：

指令		j	k	发射	操作数比较结果	
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4		√	假设乘法耗时7个周期
SUBD	F8	F6	F2		√	假设减法耗时3个周期
DIVD	F10	F0	F6	√		
ADDD	F6	F8	F2			

函数单元状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
	乘法器1	是	乘	F0	F2	F4			No	No
	乘法器2	No								
	加	是	子	F8	F6	F2			No	No
	划分	是	除	F10	F0	F6	乘1		No	是

20 寄存器结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	Mult1 M[R3+45]	M[R2+3-		加法	除法			

Scoreboard Cycle 15

指令状态：

指令		j	k	问题	操作数比较结果		
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4		✓		假设需7个周期
SUBD	F8	F6	F2			√	Assum减法运算需3个周期
DIVD	F10	F0	F6	√			
ADDD	F6	F8	F2				

I函数单元状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
	乘数1	是	乘	F0	F2	F4			No	No
	乘数2	No								
	加	No	子	F8	F6	F2			No	No
	除	是	除	F10	F0	F6	乘1		No	是

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	Mult1 除以 (R3+45)		M[R2+3+	V-	除法			

Scoreboard Cycle 16

指令状态：

指令		j	k	发射	操作数比较结果	
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4		√	假设乘法运算占用7个时钟周期
SUBD	F8	F6	F2			
DIVD	F10	F0	F6	✓		
ADDD	F6	F8	F2	√		

Fi提及 Unit Statous:

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
	乘法器1	是	乘	F0	F2	F4			No	No
	乘法器2	No								
	加	是	ADD	F6	F8	F2			是	是
	分割	是	除法	F10	F0	F6	多路复用1		No	是

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	Mult1 除以 I [R3 + 45]		加法	V-	除法			

Scoreboard Cycle 17

指令状态：

指令		j	k	问题	操作数比较结果		
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4		✓		Ass乘法运算耗时7个周期
SUBD	F8	F6	F2				
DIVD	F10	F0	F6	√			
ADDD	F6	F8	F2		√		

功能单位状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
	乘法器1	是	乘	F0	F2	F4			No	No
	乘法器2	No								
	加	是	ADD	F6	F8	F2			No	No
	除法	是	除	F10	F0	F6	乘1		No	是

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	乘法器1 / I [R3 + 45]		添加	V-	除法			

计分板循环 18

指令状态：

指令		j	k	问题	操作数比较结果	
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4		√	乘法未周期
SUBD	F8	F6	F2			
DIVD	F10	F0	F6	✓		
ADDD	F6	F8	F2		√	

功能单位状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	整数	No	加载	F2	R3				No	
	乘数1	是	乘	F0	F2	F4			No	No
	乘数2	No								
	添加	是	ADD	F6	F8	F2			No	No
	除法	是	除	F10	F0	F6	乘1		No	是

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	…	F30
0	FU	乘法器1 /(R3+45)		加法	V-	除法			

计分板周期19

指令状态：

指令		j	k	读取	执行	写入
				发射	操作数	计算 结果
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			√
SUBD	F8	F6	F2			
DIVD	F10	F0	F6	√		
ADDD	F6	F8	F2		✓	

函数单元状态：

时间	名称	忙碌	Op	Fi	Fj	Fk	RS Qj	RS Qk	Rj	Rk
		描述		S1	S2					
	整数	No	加载	F2	R3				No	
	乘数1	No	乘法	F0	F2	F4			No	No
	乘法2	No								
	加法	是	ADD	F6	F8	F2			No	No
	除法	是	除	F10	F0	F6	多选1		是	是

注册结果状态：

时钟	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU	V*	M[R3+45]	添加	V-	除法			

示例：动态调度□动态调度包含：＞乱序执行＞乱序完成□例1：浮点除法指令f0=f2/f4浮点加法指令 f10=f0+f8浮点减法指令 f12=f8-f14＞fsub.d无依赖关系，可在fadd.d之前发射 □ 例2：浮点除法指令f0=f2/f4浮点乘法指令 f6=f0*f8浮点加法指令f0=f10+f14＞fadd.d虽无数据依赖，但反依赖关系使得必须通过寄存器重命名才能提前发射

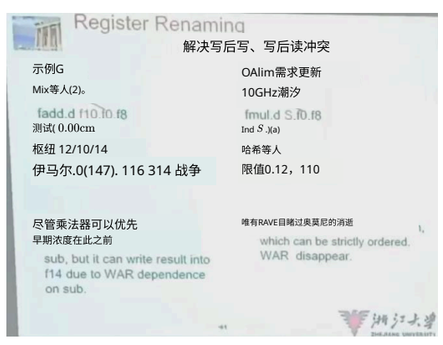
寄存器重命名

□ 示例3：

浮点除法 f0,f2,f4
浮点加法 f6,f0,f8

fsd f6,0(x1)

fsub.d
f8,f10,f14fmul.d
f6,f10,f8



寄存器重命名

□ 例3：

浮点除法指令 f0=f2/f4

浮点加法指令 S=f0+f8

浮点存储指令 [x1+0]=S

浮点减法双精度 T,f10,f14

浮点乘法双精度 f6,f10,T

□现在仅剩RAW冒险，可通过严格排序解决

记分板的局限性-1

OILP

＞若无法找到独立指令执行，记分板（或任何动态调度方案）的助益微乎其微。

“已发射”指令队列的大小

这决定了CPU能前瞻多远以寻找可并行执行的指令。
＞它被称为指令窗口。
＞目前我们假设指令窗口不能跨越分支指令。

＞换言之，该窗口仅包含基本块内的指令。

记分板机制的局限性-2

□功能单元的数量、类型及速度

>>这决定了结构冒险导致停顿的频率

□存在反依赖与输出依赖

>WAR和WAW冲突比分频器更易引发停顿，导致WAR和WAW阻塞
＞RAW冲突是所有技术都面临的问题
＞但WAR和WAW冲突可通过非分频器方式解决

记分牌算法与Tomasulo算法对比

□特点

＞乘法器等功能单元	功能较少，非流水线
＞按序发射，完成OOO	•按序发射，完成乱序执行
＞若 → 发射，Ro	浮点操作队列，保留站，加载/存储缓冲，公共数据总线
＞4 多级流水线	•寄存器重命名 → 消除WAW,WAR冲突
＞记分牌集中控制缺点	•降低结构冒险
＞WAW/WAR冲突时停顿	•分散式保留站的RAW冲突检测 公共数据总线 → 转发路径