中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

计算机体系结构

主讲教师：胡淼

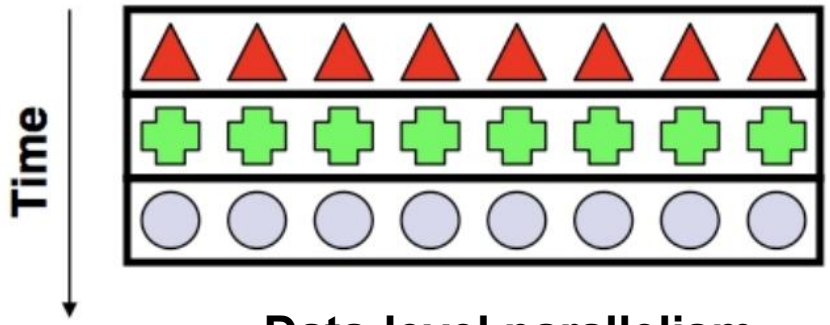
中山大学计算机学院

2024 年 秋季

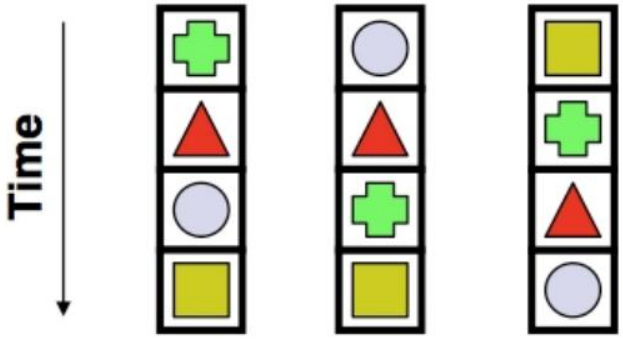
- 第 1 章 绪论
- 第 2 章 基准评测集
- 第 3 章 并行计算机的体系结构
- **第 4 章 高性能处理器的并行计算技术**
- 第 5 章 并行计算机的存储层次
- 第 6 章 并行计算机的互连网络
- 第 7 章 异构计算体系结构
- 第 8 章 领域专用体系结构

Classes of Parallelism

- **Basically two kinds of parallelism in applications:**
  - **Data-level parallelism (DLP)**
    - there are many data items that can be operated on at the same time
  - **Task-level parallelism (TLP)**
    - tasks of work are created that can operate independently and largely in parallel



Data-level parallelism



Task-level parallelism

Classes of Parallelism

- **Computer hardware exploits these two kinds of application parallelism in four major ways:**
  - **Instruction-level parallelism**
    - exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution
  - **Vector architectures, GPUs, and multimedia instruction sets**
    - exploit data-level parallelism by applying a single instruction to a collection of data in parallel
  - **Thread-level parallelism**
    - exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction between parallel threads
  - **Request-level parallelism**
    - exploits parallelism among largely decoupled tasks specified by the programmer or the operating system

第 4 章 高性能处理器的并行计算技术

4.1 指令级并行 part1

参考资料

- Computer Architecture, A Quantitative Approach, 6th Edition.
  - Appendix C. “Pipelining: Basic and Intermediate Concepts”
  - Chapter 3. “Instruction-Level Parallelism and Its Exploitation”

内容纲要

- 指令级并行简介
- 基于循环展开的指令调度
- 基于计分板的指令调度
- 基于Tomasulo算法的指令调度

什么是指令级并行

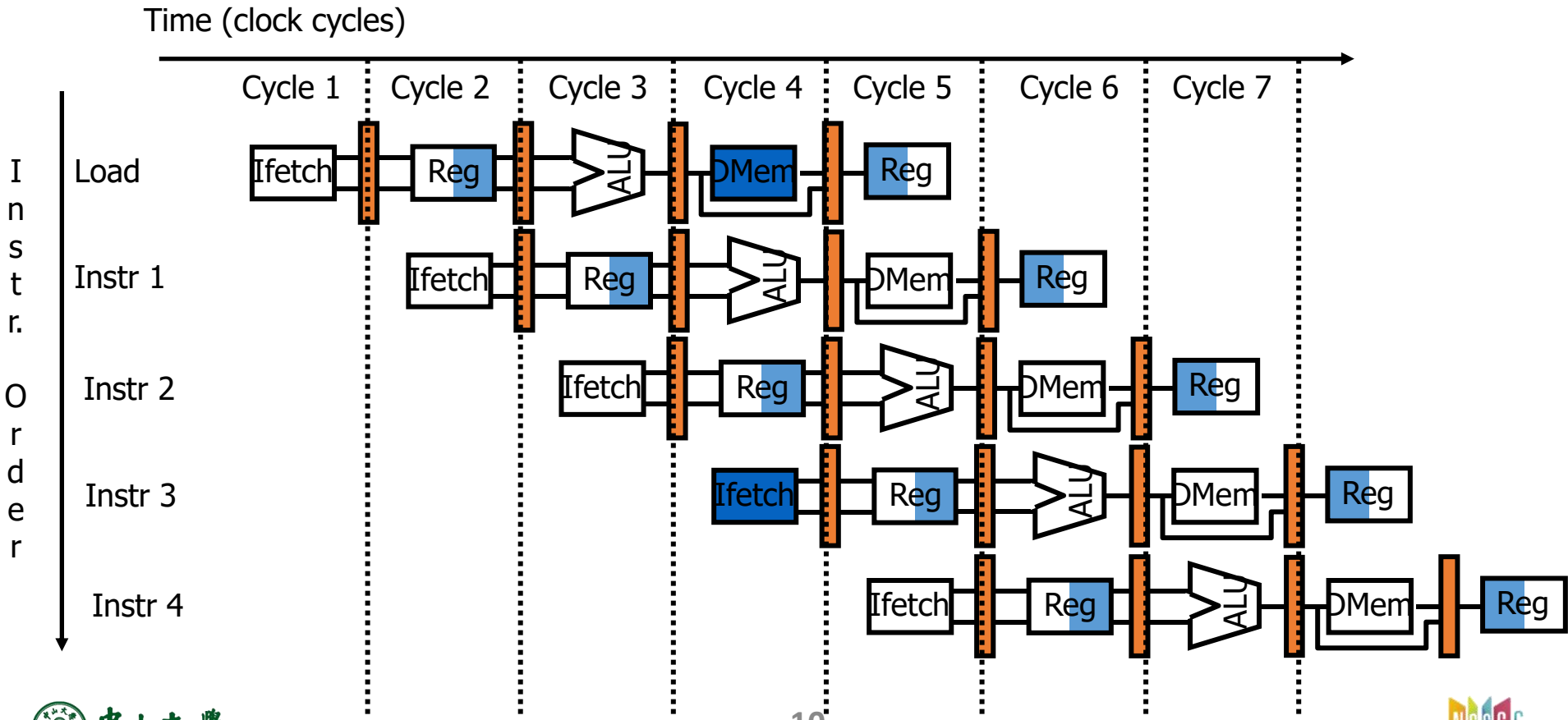
- 现代处理器的典型特征
  - 深度流水线
    - 十多个功能段的流水线
  - 多功能部件
    - 多个浮点乘法器、加法器等
  - 一个时钟周期能够流出多条指令
- 发掘指令之间的并行性已成为现代处理器性能优化的重要方面
  - 软件技术
    - 循环展开
    - VLIW
  - 硬件技术
    - 分支预测
    - 推测执行
    - 动态调度

指令级并行的挑战

- 指令之间存在一定的竞争或者依赖关系，导致指令级并行存在一定的挑战
  - 结构冒险
    - 多条指令争用同一个功能部件
  - 数据冒险
    - 数据之间存在真假依赖
  - 控制冒险
    - 分支语句执行的不确定性

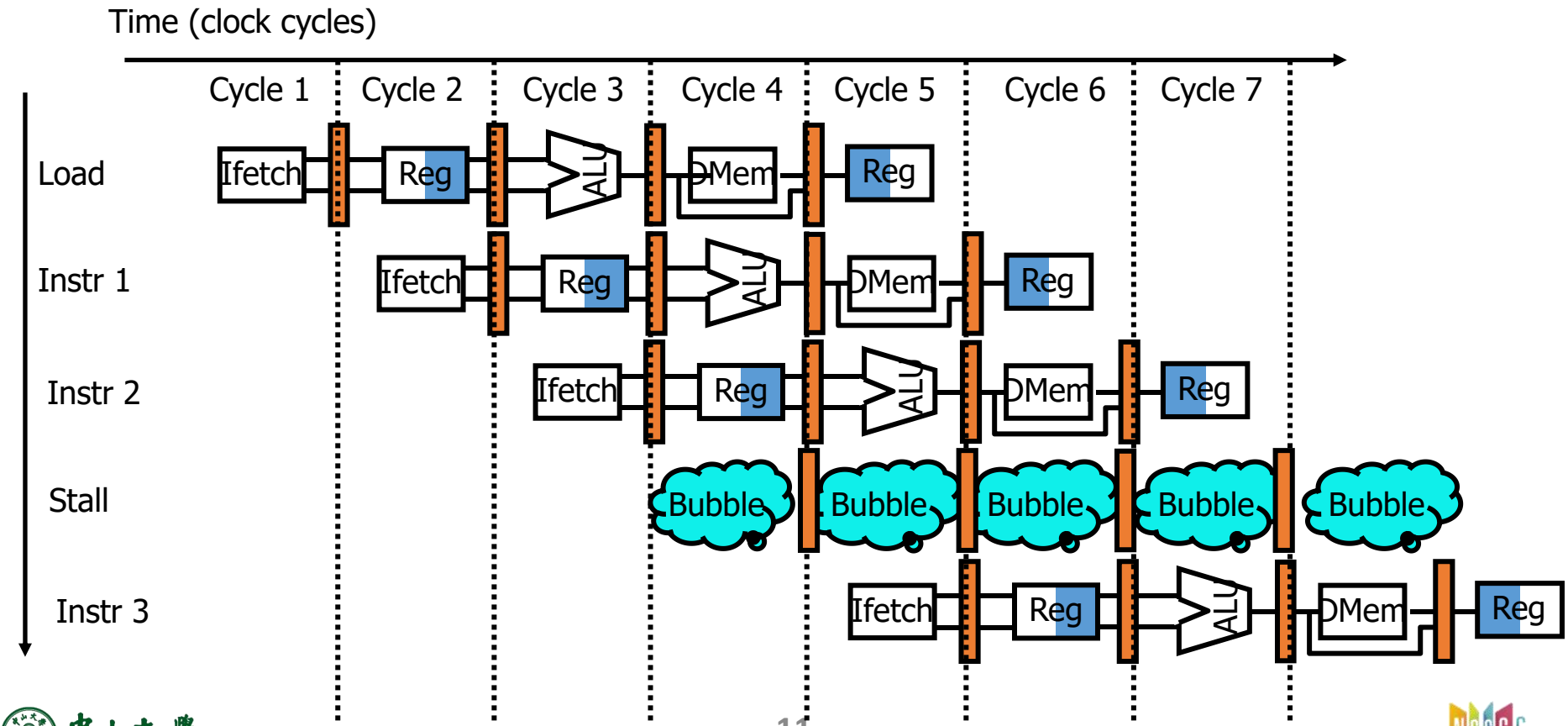
指令级并行的挑战

- 典型的结构冒险：只有一个内存端口，内存争用
  - Cycle4中，访存和取指冲突



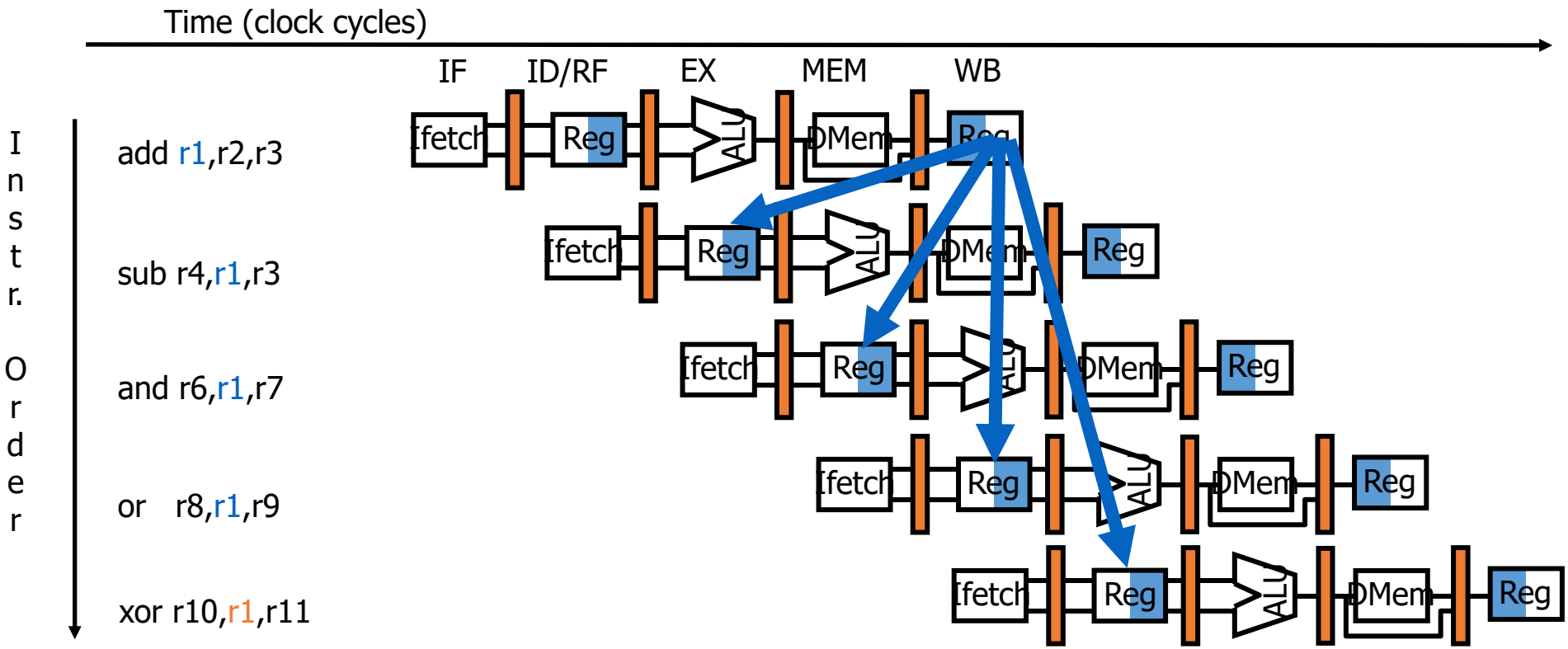
指令级并行的挑战

- 典型的结构冒险：只有一个内存端口，内存争用
  - 第四条指令需要延迟发射
  - 解决方法：L1缓存分成数据Cache和指令Cache



指令级并行的挑战

- 典型的数据冒险：数据之间存在依赖
  - 后续指令对R1寄存器的结果依赖



指令级并行的挑战

- 数据相关的类型
  - Read After Write (RAW)（写后读）
    - 真相关

    I: add r1,r2,r3

    J: sub r4,r1,r3
  - Write After Read (WAR)（读后写）
    - 名字相关、反相关

    I: sub r4,r1,r3

    J: add r1,r2,r3
  - Write After Write (WAW)（写后写）
    - 名字相关、输出相关

    I: sub r1,r4,r3

    J: add r1,r2,r3
  - Read After Read (RAR)

指令级并行的挑战

- 数据相关的几点说明
  - 数据相关具有传递性
    - i->j，j->h，则i->h
  - 数据相关不一定导致数据冒险（Hazard）
    - 数据相关是程序的属性
    - 是否会发生冒险，还取决于处理器的体系结构和功能部件
      - 所以合理利用功能部件能避免冒险
    - 两条存在相关的指令，在指令序列中距离较远，也不会发生冒险
      - 所以调度指令序列能避免冒险
  - 数据相关给出了可发掘的指令级并行的上限

指令级并行的挑战

- 典型的控制冒险：分支目标不确定，无法确定后续指令
  - 如果一条指令是否执行依赖于一条分支指令的执行结果，则不能把这条指令提到分支指令之前
    - 使这条指令的执行不再受控于分支指令
  - 如果一条指令与一个分支指令没有控制相关，则不能把这条指令放在分支指令之后
    - 使这条指令受控于这个分支
- 控制冒险在很大程度上限制了指令级并行
  - 对于典型的MIPS程序，分支频率约15%~25%，平均3~6条指令就有一个分支出现
  - 控制冒险不解决，指令级并行的上限很低

指令级并行的挑战

- 典型的控制冒险

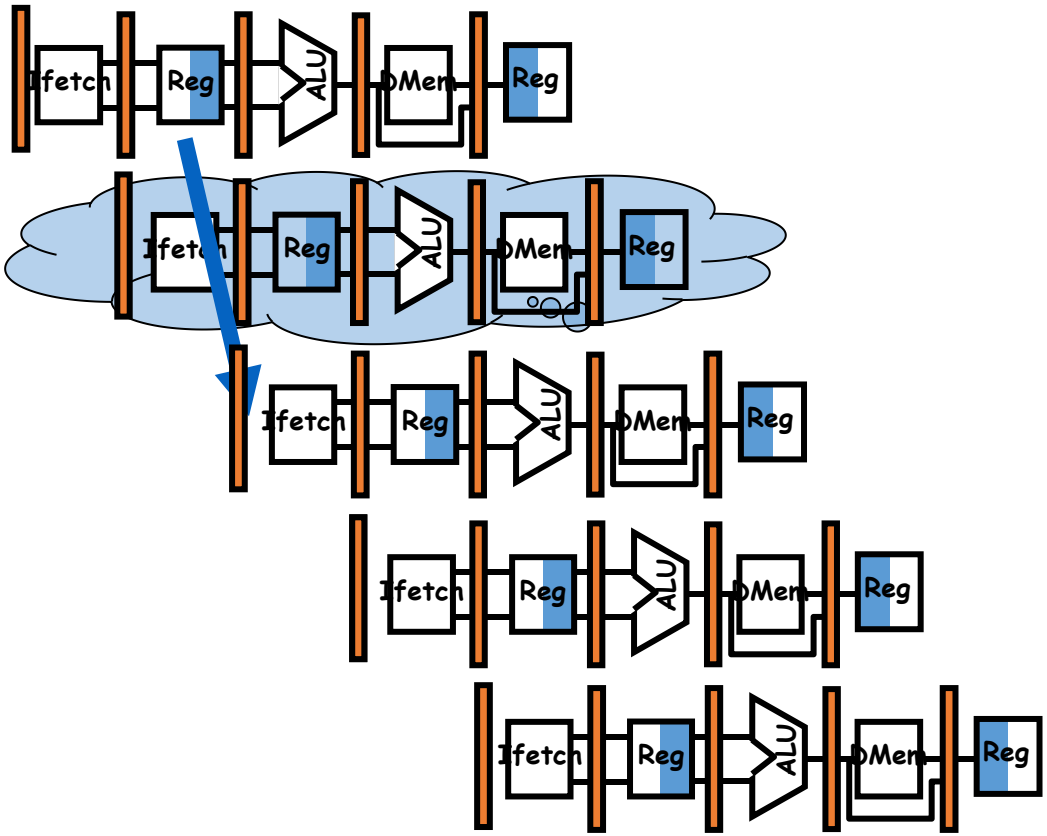
10: beq r1,r3,36

14: and r2,r3,r5

18: or r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11



发掘指令级并行的目的

- 降低程序执行的平均CPI

Pipeline CPI = Ideal pipeline CPI
+ Structural stalls
+ Data hazard stalls
+ Control stalls

数据相关分析

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的RAW相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的RAW相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的RAW相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的RAW相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的WAR相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的WAR相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 典型的WAW相关

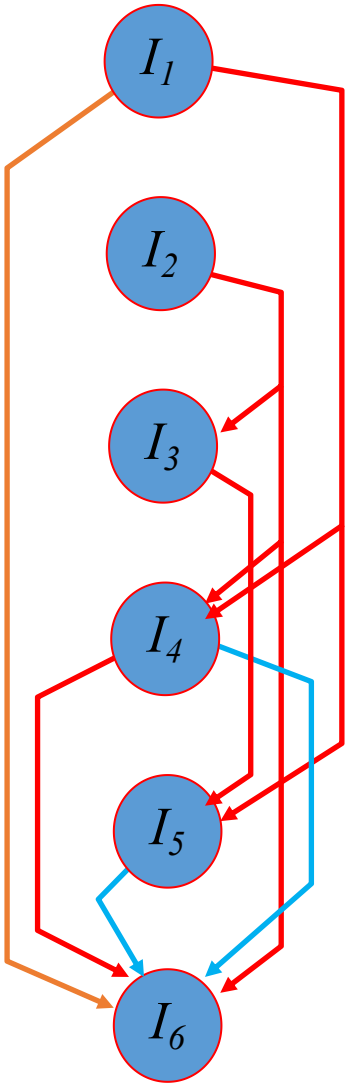
I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

数据相关分析

- 所有的数据相关

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

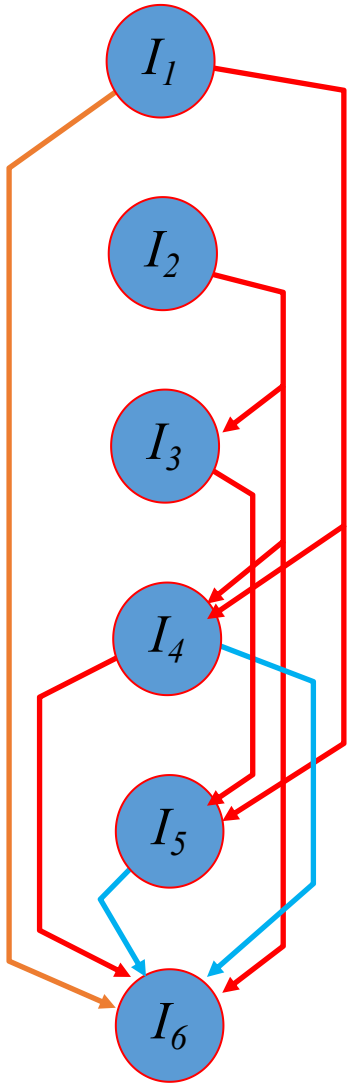
- Execuation order
  - In-order
  - Out-of-order (o-o-o)



数据相关分析

- 指令执行的顺序

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2

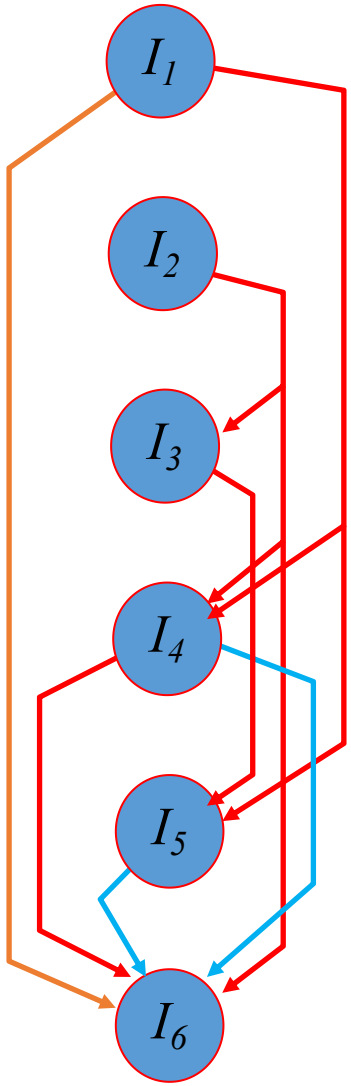


in-order	I1	I2	I3	I4	I5	I6
o-o-o						

数据相关分析

- 指令执行的顺序

I1	FDIV.D	f6,	f6,	f4
I2	FLD	f2,	45(x3)	
I3	FMUL.D	f0,	f2,	f4
I4	FDIV.D	f8,	f6,	f2
I5	FSUB.D	f10,	f0,	f6
I6	FADD.D	f6,	f8,	f2



in-order	I1	I2	I3	I4	I5	I6
o-o-o	I2	I1	I3	I5	I4	I6

数据相关分析

- 假定各种指令的额外延迟
  - 相对标准五段流水所需要的额外时钟数
  - 即：如果一条指令想要使用本指令的结果，必须延后几个周期发射

					Latency*
I1	FDIV.D	f6,	f6,	f4	4
I2	FLD	f2,	45(x3)		1
I3	FMUL.D	f0,	f2,	f4	3
I4	FDIV.D	f8,	f6,	f2	4
I5	FSUB.D	f10,	f0,	f6	1
I6	FADD.D	f6,	f8,	f2	1

数据相关分析

- 按序发射/按序执行指令所需要的时间分析

I1	FDIV.D	f6,	f6,	f4	4
I2	FLD	f2,	45(x3)		1
I3	FMUL.D	f0,	f2,	f4	3
I4	FDIV.D	f8,	f6,	f2	4
I5	FSUB.D	f10,	f0,	f6	1
I6	FADD.D	f6,	f8,	f2	1

clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
in/in	1	2			1	2	3	4		3	5	4	6	5	6

数据相关分析

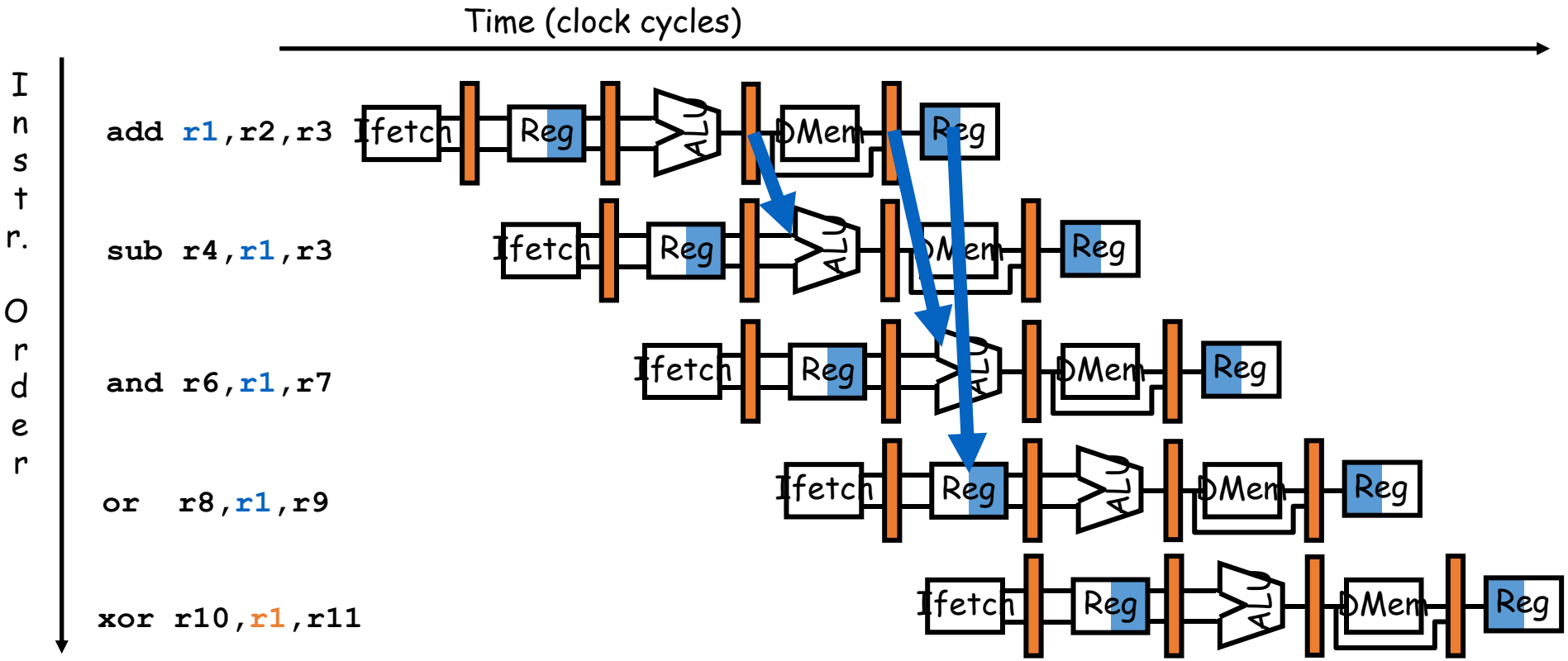
- 按序发射/乱序执行指令所需要的时间分析

I1	FDIV.D	f6,	f6,	f4	4
I2	FLD	f2,	45(x3)		1
I3	FMUL.D	f0,	f2,	f4	3
I4	FDIV.D	f8,	f6,	f2	4
I5	FSUB.D	f10,	f0,	f6	1
I6	FADD.D	f6,	f8,	f2	1

clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
in/out	1	2	2	3	1	4	3	5	5	4	6	6			

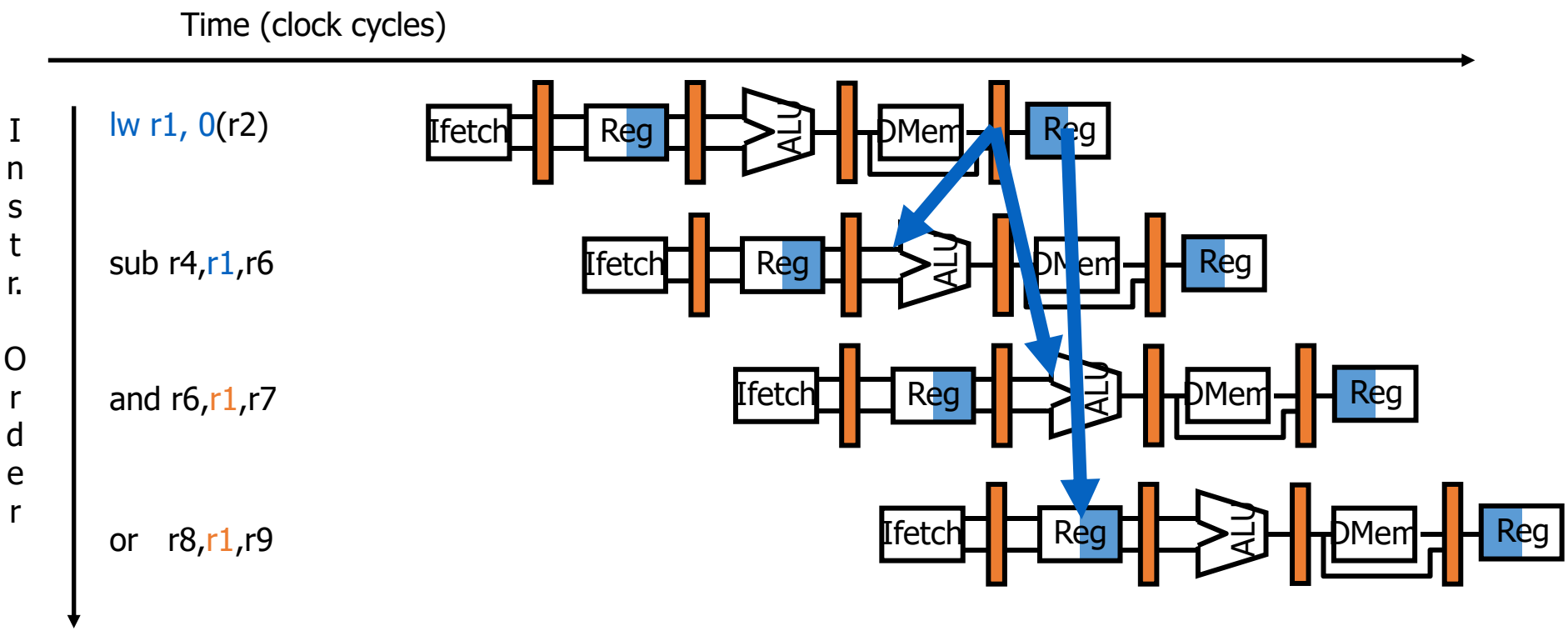
初级的数据相关解决方法

- 转发（Forward）
  - 将计算出的结果尽早的发送到等待该结果的部件



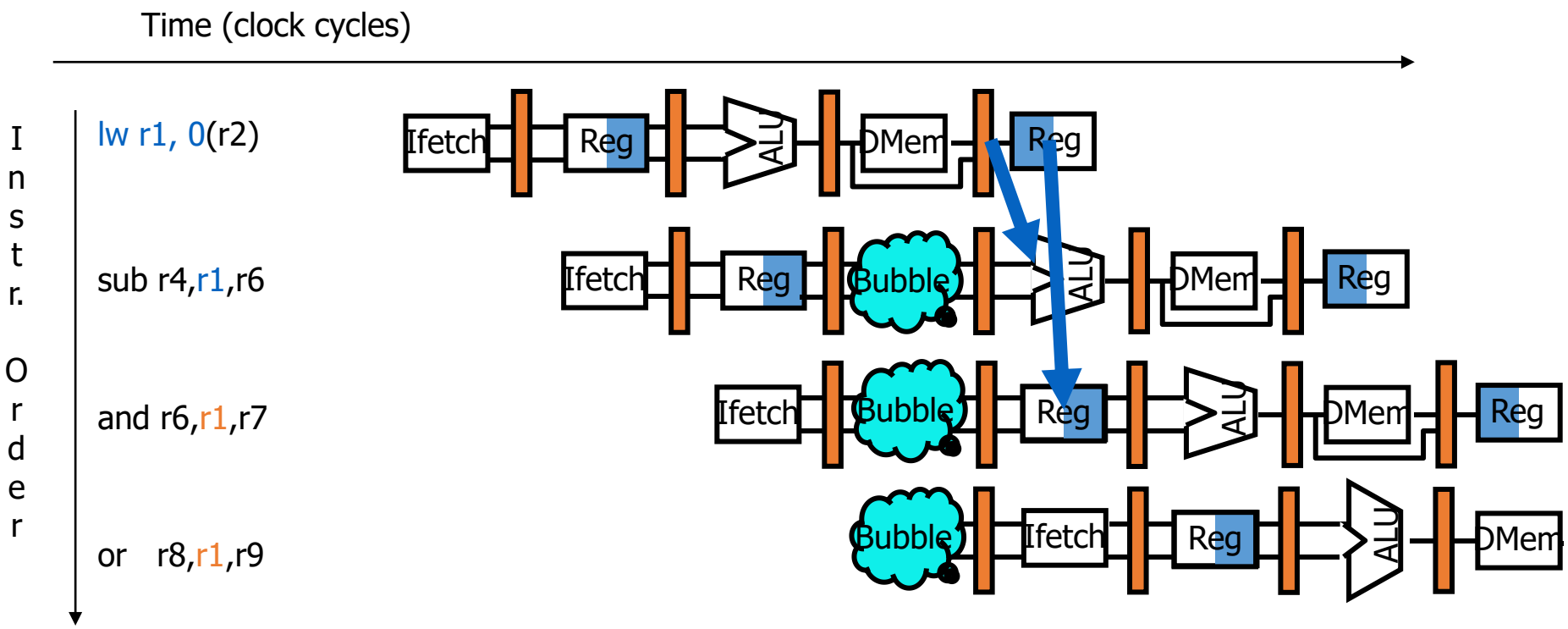
初级的数据相关解决方法

- 转发（Forward）
  - 转发并不能解决所有的问题



初级的数据相关解决方法

- 转发（Forward）
  - 转发并不能解决所有的问题



内容纲要

- 指令级并行简介
  - 乱序执行能够带来收益
  - 转发不能解决所有问题
- 基于循环展开的指令调度
- 基于计分板的指令调度
- 基于Tomasulo算法的指令调度

循环展开

- 循环展开的优势
  - 降低循环条件判断在代码中的比例
    - 分支指令本身会带来额外的延迟
    - 分支指令会导致控制冒险
  - 提升循环体内可发掘的指令级并行性
    - 循环体内指令变多，能够充分占据功能部件
    - 乱序执行调度的余地更大
- 一些基本的假设
  - 五段流水
  - 分支延迟为1个额外时钟
  - 定点计算的额外延迟为0
  - 浮点单元足够多，能够充分流水

产生结果指令	等待结果指令	延迟
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

循环展开

```
for (i=999; i>=0; i=i–1)
        x[i] = x[i] + s;
```

```
Loop:  fld       f0,0(x1)       //f0=array element
       fadd.d    f4,f0,f2       //add scalar in f2
       fsd       f4,0(x1)       //store result
       addi      x1,x1,-8       //decrement pointer
                                //8 bytes (per DW)
       bne       x1,x2,Loop     //branch x1 != x2
```

- R1: 数组的上界
- R2: 数组的下界

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	? fadd.d f4,f0,f2

读浮点数后面跟浮点数ALU指令，需要等待1个时钟

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	? fsd f4,0(x1)

浮点ALU运算后面跟写内存指令，需要等待2个时钟

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	*Stall*
5	*Stall*
6	fsd f4,0(x1)

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	*Stall*
5	*Stall*
6	fsd f4,0(x1)
7	? addi x1,x1,-8

addi指令试图修改x1的值时，fsd指令早就取数了

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	*Stall*
5	*Stall*
6	fsd f4,0(x1)
7	addi x1,x1,-8
8	? bne x1,x2,Loop

addi指令的计算结果，bne指令需要延迟一个时钟才能拿到

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	*Stall*
5	*Stall*
6	fsd f4,0(x1)
7	addi x1,x1,-8
8	*Stall*
9	bne x1,x2,Loop

未调度的循环执行

Clock	Unscheduled
1	fld f0,0(x1)
2	*Stall*
3	fadd.d f4,f0,f2
4	*Stall*
5	*Stall*
6	fsd f4,0(x1)
7	addi x1,x1,-8
8	*Stall*
9	bne x1,x2,Loop
10	*Stall*

未调度的循环执行

- 每次循环需要10个时钟周期
- 有效负荷较少：只有fld、fadd、fsd三条指令
  - 有效负荷占30%
  - 50%的时钟停顿
  - 20%的时钟用于循环控制
- 还有进一步的提升空间吗
  - 尽可能消除停顿

循环体的指令调度

Clk	Unscheduled	scheduled
1	fld f0,0(x1)	fld f0,0(x1)
2	*Stall*	
3	fadd.d f4,f0,f2	
4	*Stall*	
5	*Stall*	
6	fsd f4,0(x1)	
7	addi x1,x1,-8	
8	*Stall*	
9	bne x1,x2,Loop	
10	*Stall*	

循环体的指令调度

Clk	Unscheduled	scheduled
1	fld f0,0(x1)	fld f0,0(x1)
2	*Stall*	addi x1,x1,-8
3	fadd.d f4,f0,f2	
4	*Stall*	
5	*Stall*	
6	fsd f4,0(x1)	
7	addi x1,x1,8	
8	*Stall*	
9	bne x1,x2,Loop	
10	*Stall*	

循环体的指令调度

Clk	Unscheduled	scheduled
1	fld f0,0(x1)	fld f0,0(x1)
2	*Stall*	addi x1,x1,-8
3	fadd.d f4,f0,f2	fadd.d f4,f0,f2
4	*Stall*	
5	*Stall*	
6	fsd f4,0(x1)	
7	addi x1,x1,8	
8	*Stall*	
9	bne x1,x2,Loop	
10	*Stall*	

循环体的指令调度

Clk	Unscheduled	scheduled
1	fld f0,0(x1)	fld f0,0(x1)
2	*Stall*	addi x1,x1,-8
3	fadd.d f4,f0,f2	fadd.d f4,f0,f2
4	*Stall*	*? Stall*
5	*Stall*	*? Stall*
6	fsd f4,0(x1)	fsd f4,8(x1)
7	addi x1,x1,8	
8	*Stall*	
9	bne x1,x2,Loop	
10	*Stall*	

循环体的指令调度

Clk	Unscheduled		scheduled	
1	fld	f0,0(x1)	fld	f0,0(x1)
2	*Stall*		addi	x1,x1,-8
3	fadd.d	f4,f0,f2	fadd.d	f4,f0,f2
4	*Stall*		*Stall*	
5	*Stall*		bne	x1,x2,Loop
6	fsd	f4,0(x1)	fsd	f4,8(x1)
7	addi	x1,x1,8		
8	*Stall*			
9	bne	x1,x2,Loop		
10	*Stall*			

循环体的指令调度

- 每次循环需要6个时钟周期
  - 加速比：10/6=1.67
- 有效负荷仍然较少：只有fld、fadd、fsd三条指令
  - 有效负荷占50%
  - 一个时钟停顿，占16.67%
  - 两个时钟用于循环控制，占33.33%
- 还有进一步的提升空间吗
  - 尽可能增加有效负荷
  - 减少循环控制开销

进一步的循环展开

```
Loop:   fld f0,0(x1)
        fadd.d f4,f0,f2
        fsd f4,0(x1)          //drop addi & bne
        fld f6,8(x1)
        fadd.d f8,f6,f2
        fsd f8,8(x1)          //drop addi & bne
        fld f10,16(x1)
        fadd.d f12,f0,f2
        fsd f12,16(x1)        //drop addi & bne
        fld f14,24(x1)
        fadd.d f16,f14,f2
        fsd f16,24(x1)
        addi x1,x1,32
        bne x1,x2,Loop
```

进一步的循环展开

- 如果不做调度，仍然很差

		start waiting, …,execute
Loop:	fld f0,0(x1)	1
	fadd.d f4,f0,f2	2,3
	fsd f4,0(x1)	4,5,6
	fld f6,8(x1)	7
	fadd.d f8,f6,f2	8,9
	fsd f8,8(x1)	10,11,12
	fld f10,16(x1)	13
	fadd.d f12,f10,f2	14,15
	fsd f12,16(x1)	16,17,18
	fld f14,24(x1)	19
	fadd.d f16,f14,f2	20,21
	fsd f16,24(x1)	22,23,24
	addi x1,x1,32	25
	bne x1,x2,Loop	26,27
	Stall	28

未加调度的循环展开

- 每次循环需要28个时钟周期
  - 发射14条指令
  - 12条指令的有效负荷
  - 14个时钟周期的停顿
  - 平均每个原始循环7个时钟
  - 需要更多的寄存器
- 还有进一步的提升空间吗
  - 循环展开 & 指令调度

指令调度的循环展开

```
1        Loop:  fld     f0,0(x1)
2               fld     f6,8(x1)
3               fld     f10,16(x1)
4               fld     f14,24(x1)
5               fadd.d  f4,f0,f2
6               fadd.d  f8,f6,f2
7               fadd.d  f12,f0,f2
8               fadd.d  f16,f14,f2
9               fsd     f4,0(x1)
10              fsd     f8,8(x1)
11              addi    x1,x1,32
12              fsd     f12,-16(x1)
13              bne     x1,x2,Loop
14              fsd     f16,-8(x1)
```

指令调度的循环展开

- 每次循环需要14个时钟周期
  - 发射14条指令
  - 12条指令的有效负荷，占比85.7%
  - 0个时钟周期的停顿
  - 平均每个原始循环3.5个时钟
- 加速比计算
  - 未调度未展开：10个时钟
  - 有调度未展开：6个时钟
  - 未调度有展开：7个时钟
  - 有调度有展开：3.5个时钟

循环展开总结

- 循环展开的步骤
  - 确认循环迭代是不相关的，从而能够展开
  - 使用不同的寄存器，避免名字相关
  - 去除多余的分支与条件指令，调整循环终止与迭代代码
  - 分析是否存在关于存储地址的相关性
  - 对代码进行调度
- 缺点
  - 代码量显著增长，并由此可能引起频繁的指令缓存缺失
  - 寄存器消耗较多，并由此可能引起不必要的访存操作

内容纲要

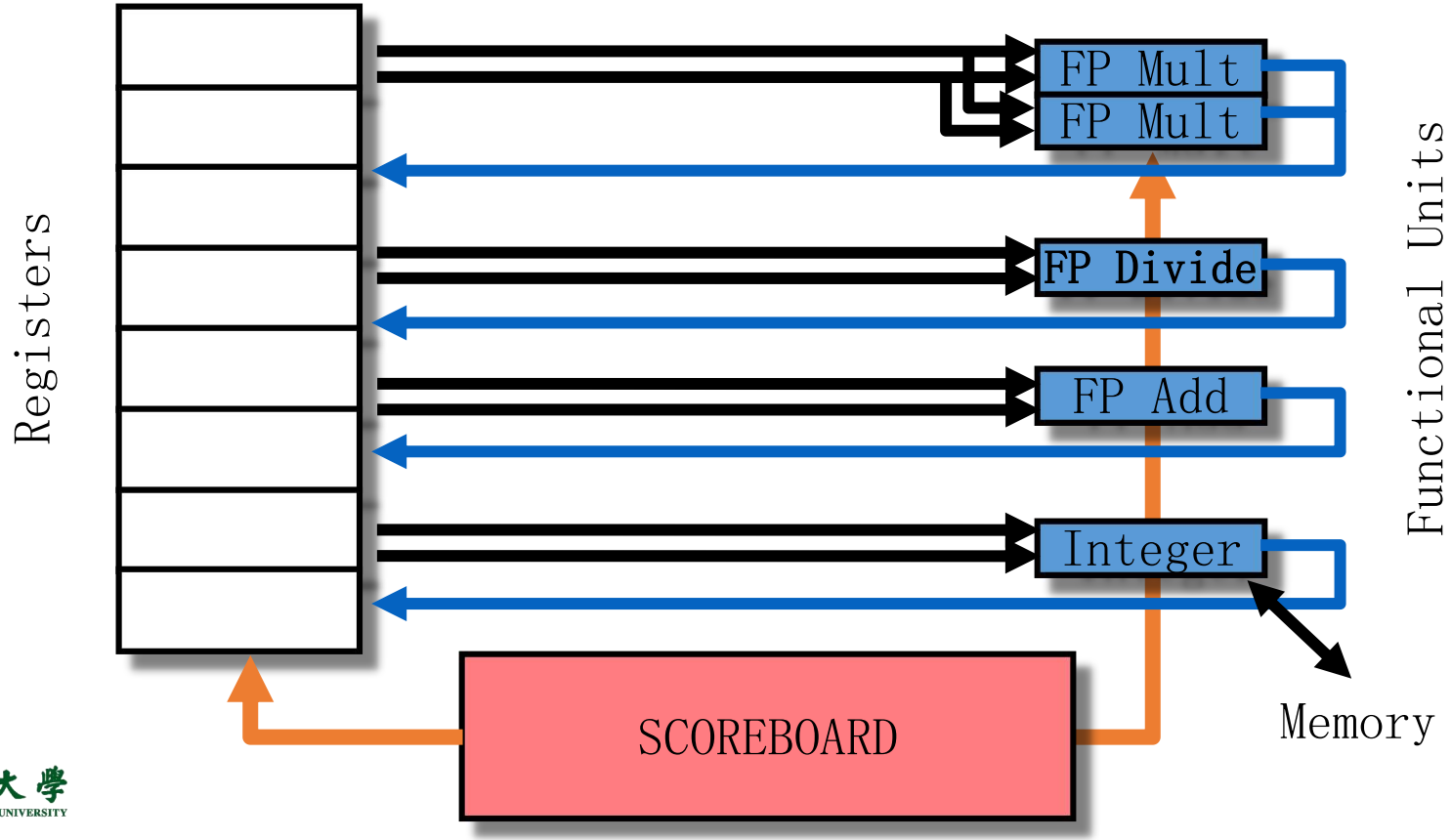
- 指令级并行简介
- 基于循环展开的指令调度
- 基于计分板的指令调度

静态调度的缺陷

- 以循环展开为代表的静态调度的缺陷
  - 需要对每个流水段的执行时间有明确的预期
- 实际上，指令执行有很多的不确定性
  - 同一种指令集体系结构具有不同的组成和实现方式，从而具有不同的微体系结构
    - 例如：流水线不同，针对一种流水线的静态调度难以在另一种流水线上高效执行
    - 不可能要求所有的软件为所有的微体系结构编译出一个可执行文件版本
  - 一些相关性在编译阶段难以发现
    - 存储地址带来的相关性
    - 大量的分支指令
  - 一些不可控的额外延迟，如：缓存缺失

基于计分板的动态调度

- 计分板（ScoreBoard）的结构
  - 记录已发射指令的状态
  - 记录各个寄存器的使用和等待情况
  - 记录各个功能单元的使用和等待情况



基于计分板的动态调度

- 计分板的动态调度机制
  - 针对多个功能部件和流水线展开调度
  - 针对RAW相关
    - 设置数据就绪位，强制等待数据就绪
  - 针对WAR相关
    - 在一个寄存器被读之前，不允许后续的指令写
  - 针对WAW相关
    - 延迟寄存器写回，避免覆盖

流水线的各个阶段

- 第一阶段-发射阶段：译码、检查结构冒险
  - 按指令的原始顺序发射
  - 有结构冒险则不能发射
  - 有WAW冒险则不能发射
- 第二阶段-读操作数：在没有冒险时读操作数
  - 在RAW消除后读操作数
  - 计分板不考虑转发（Forward）
- 第三阶段-执行：对操作数做运算
  - 执行指令，并向计分板通知指令执行完成
- 第四阶段-写回：写回结果
  - 在没有WAR冒险时写回结果

计分板的运作机制

- 记录指令状态
  - 每条已发射指令处于四个阶段的哪一个
- 记录寄存器的状态
  - 记录每个寄存器是否会被某个功能单元写入
  - 如果是，记录将会被哪个功能单元写入
- 记录功能单元的状态
  - 记录各个功能单元的使用和等待情况
    - 9个字段的标记

计分板的运作机制

- 9个字段的含义
  - Busy：当前单元是否空闲
  - Op：当前单元执行的操作
  - Fi：目的寄存器
  - Fj、Fk：源寄存器编号
  - Qj、Qk：为源寄存器产生数据的功能单元
  - Rj、Rk：源寄存器中数据是否就绪的标志位

基于计分板的动态调度

- 找出RAW相关与WAR相关

L.D	**F6, 34(R2)**
L.D	**F2, 45(R3)**
MUL.D	**F0, F2, F4**
SUB.D	**F8, F2, F6**
DIV.D	**F10, F0, F6**
ADD.D	**F6, F8, F2**

基于计分板的动态调度

- RAW相关

L.D	**F6, 34(R2)**
L.D	**F2, 45(R3)**
MUL.D	**F0, F2, F4**
SUB.D	**F8, F2, F6**
DIV.D	**F10, F0, F6**
ADD.D	**F6, F8, F2**

基于计分板的动态调度

- WAR相关

L.D	**F6, 34(R2)**
L.D	**F2, 45(R3)**
MUL.D	**F0, F2, F4**
SUB.D	**F8, F2, F6**
DIV.D	**F10, F0, F6**
ADD.D	**F6, F8, F2**

- 一些延迟假设
  - ADD：读到操作数后2时钟计算出结果
  - MULTIPLY：读到操作数后10时钟计算出结果
  - DIVIDE：40时钟

基于计分板的动态调度（初始化）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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									

基于计分板的动态调度（第1时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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					

基于计分板的动态调度（第2时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

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					

基于计分板的动态调度（第2时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

- 第二个LD可发射吗
  - 结构冒险

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					

基于计分板的动态调度（第3时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

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					

基于计分板的动态调度（第3时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

- 可发射MULT吗
  - F2不确定从哪里得到

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								

基于计分板的动态调度（第4时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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								

基于计分板的动态调度（第5时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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							

基于计分板的动态调度（第6时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	Mult	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*	Mult1	Integer							

基于计分板的动态调度（第7时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

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

Register result status:

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

基于计分板的动态调度（第7时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

- Read multiply operands?

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

Register result status:

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

基于计分板的动态调度（第8时钟前半段）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

- Read operands for MULT & SUB?

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	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			

基于计分板的动态调度（第8时钟后半段）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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			

基于计分板的动态调度（第9时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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:

Note Remaining

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

基于计分板的动态调度（第9时钟）

- 可发射ADDD吗

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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:

Note Remaining

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

基于计分板的动态调度（第10时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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			

基于计分板的动态调度（第11时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	Mult1				Add	Divide			

基于计分板的动态调度（第12时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

基于计分板的动态调度（第12时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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				

- Read operands for DIVD?
- Issue ADDD?

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

Register result status:

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

基于计分板的动态调度（第13时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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			

基于计分板的动态调度（第14时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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			

基于计分板的动态调度（第15时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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:

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

基于计分板的动态调度（第16时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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:

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

基于计分板的动态调度（第17时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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								
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		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
17	*FU*	Mult1			Add		Divide			

基于计分板的动态调度（第17时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	

- Why not write result of ADDD?

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

基于计分板的动态调度（第17时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	

- Why not write result of ADDD?

WAR Hazard!

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

基于计分板的动态调度（第18时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
18	*FU*	Mult1			Add		Divide			

基于计分板的动态调度（第19时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	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		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
19	*FU*	Mult1			Add		Divide			

基于计分板的动态调度（第20时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

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

基于计分板的动态调度（第21时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	

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	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

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

基于计分板的动态调度（第21时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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 Hazard is now gone...

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	Add	F6	F8	F2			No	No
40	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

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

基于计分板的动态调度（第22时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	Div	F10	F0	F6			No	No

Register result status:

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

基于计分板的动态调度（第61时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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	Div	F10	F0	F6			No	No

Register result status:

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

基于计分板的动态调度（第62时钟）

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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								
	Divide	No								

Register result status:

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

基于计分板的调度过程回顾

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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								
	Divide	No								

Register result status:

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

基于计分板的调度过程回顾

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*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

- 按序发射
- 乱序执行
- 乱序写回

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*
62	*FU*									

CDC6600上的计分板

- CDC6600上的计分板不足
  - 没有转发（Forward）
  - 没有考虑分支指令
    - 只能处理基本块内的调度，窗口较小
  - 发生结构冒险时，不发射指令
    - 实际上可以发射，因为有流水线
  - 没有重命名
    - 等待WAR冒险
    - 阻止WAW冒险
  - 写寄存器与读该寄存器的数据不能同时进行，需要一个额外时钟

内容纲要

- 指令级并行简介
- 基于循环展开的指令调度
- 基于计分板的指令调度
- 基于Tomasulo算法的指令调度

数据相关分析

- **真数据相关(True Data Dependence)**
  - 只有当指令的操作数可用时，才能执行指令，从而避免RAW冒险
  - 计分板提供

- WAR和WAW冒险源于**名字相关（Name Dependence）**
  - 通过寄存器重命名(Register Renaming)消除

Register Renaming

- **Example:**

```
fdiv.d f0,f2,f4
fadd.d f6,f0,f8
fsd f6,0(x1)
fsub.d f8,f10,f14
fmul.d f6,f10,f8
```

antidependence

output dependence

- name dependence

Register Renaming

- **Example:**

```
fdiv.d f0,f2,f4
fadd.d S,f0,f8
fsd S,0(x1)
fsub.d T,f10,f14
fmul.d f6,f10,T
```

```
fdiv.d f0,f2,f4
fadd.d f6,f0,f8
fsd f6,0(x1)
fsub.d f8,f10,f14
fmul.d f6,f10,f8
```

- **Now only RAW hazards remain, which can be strictly ordered**

另一个动态算法: 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
  - IBM的每条指令有两个寄存器描述符(register specifiers)，而CDC 6600有三个
  - IBM有四个浮点寄存器，而CDC 6600有八个

- 为什么要学习Tomasulo算法？
  - 由此产生了Alpha 21264、HP 8000、MIPS 10000、 Pentium II、PowerPC 604，…

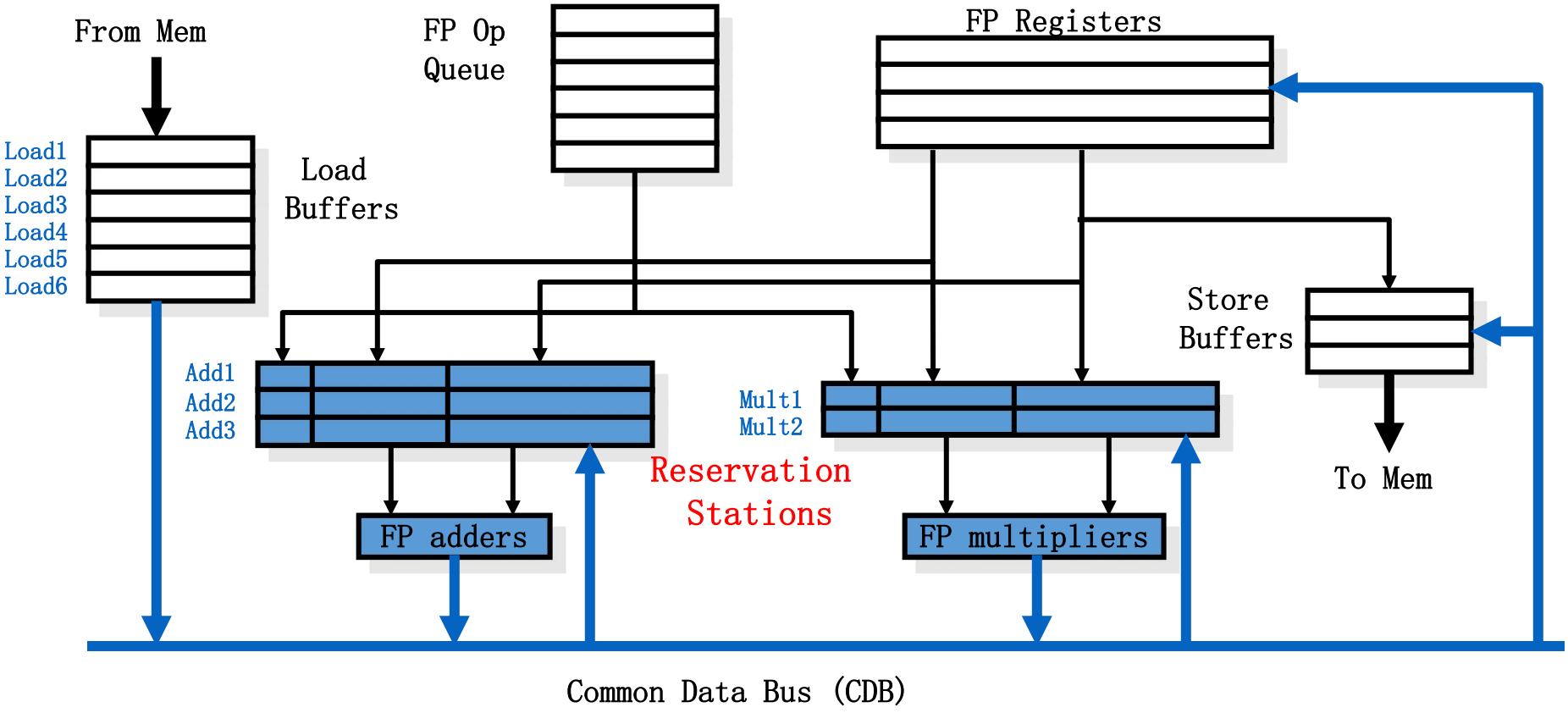
Tomasulo Algorithm

- **Tomasulo's Approach**
  - Tracks when operands are available
  - Introduces register renaming in hardware
    - avoid WAW and WAR hazards

- **Register renaming** is provided by **reservation stations (RS)**, which contains:
  - The instruction
  - Buffered operand values (when available)
  - Reservation station number of instruction providing the operand values

基于Tomasulo算法的指令调度

- Tomasulo的组织结构
  - 引入保留站：Reservation Stations



Tomasulo Algorithm

- RS fetches and buffers an operand **as soon as** it becomes available
  - not necessarily involving register file
- Pending instructions **designate the RS** to which they will send their output
  - Result values broadcast on a result bus, called the **Common Data Bus (CDB)**
- Only the **last** output updates the register file
- As instructions are issued, the register specifiers are **renamed** with the reservation station
  - May be more reservation stations than registers
- Load and store buffers
  - Contain data and addresses, act like reservation stations

Tomasulo算法 与 计分板

- 控制&缓冲器 分布于 功能部件(FU) 与 集中于计分板
  - 功能部件缓冲器称为 “保留站(reservation stations)”
  - 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 寄存器重命名(register renaming)
  - 消除WAR、WAW冒险
  - 保留站 比 实际寄存器 多，因而可以完成优化编译器所不能完成的一些工作结果
  - 从 RS 直接 到 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FU
  - 载入（Load） 和 存储（Stores）也像 其他功能部件一样使用保留站

Tomasulo's Algorithm

- **Three Steps:**
  - **1) Issue**
    - Get next instruction from FIFO queue
    - If available RS (no structural hazard), issue the instruction to the RS with operand values if available (renames registers)
    - If operand values not available, stall the instruction
      - This stage is sometimes called *dispatch* in a dynamically scheduled processor

Tomasulo's Algorithm

- **Three Steps:**
  - **2) Execute**
    - When operand becomes available, store it in any reservation stations waiting for it
    - When all operands are ready, the operation can be executed
    - When both operands ready, then execute
    - If not ready, watch Common Data Bus for result

Tomasulo's Algorithm

- **Three Steps:**
  - **3) Write result**
    - Write result on CDB, and from there into registers, reservation stations and store buffers waiting for this result
    - Stores must wait until store address and value to be stored are available
    - Mark reservation station available

Common Data Bus

- **Normal data bus:**
  - data + destination ("go to" bus)
- **Common data bus:**
  - data + source ("come from" bus)
  - 64 bits of data + 4 bits of Functional Unit source address
  - Write if matches expected Functional Unit (produces result)
  - Does the broadcast

基于Tomasulo算法的指令调度

- 保留站中记录的信息
  - Op：操作类型
  - Busy：是否占用
  - Vj，Vk:源操作数的值
    - 已经就绪的源操作数保留在此
  - Qj，Qk:如果源操作数的值还未就绪，它依赖保留站的哪一项
    - Qj，Qk = 0 => ready
    - 对于一个源操作数，V和Q只有一个有效
  - A：Load、Store指令对应的地址
    - 如果地址还未算出来，则保留立即数
    - 如果地址已经计算，则保留访存目标地址的值
- 寄存器组保留的信息
  - Qi：i号寄存器的值由哪项保留站算出

基于Tomasulo算法的指令调度

L.D	F6, 34(R2)
L.D	F2, 45(R3)
MUL.D	F0, F2, F4
SUB.D	F8, F2, F6
DIV.D	F10, F0, F6
ADD.D	F6, F8, F2

基于Tomasulo算法的指令调度

Instruction stream

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/Buffers

Reservation Stations:

FU count down

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

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

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

基于Tomasulo算法的指令调度

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					

基于Tomasulo算法的指令调度

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					

注意：可发射多条Load指令

基于Tomasulo算法的指令调度

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		R(F4)	Load2	
	Mult2	No					

Register result status:

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

- 注意：MULT已发射，其两个源操作数保存在保留站中，寄存器不再用了
- Load1执行结束，可广播其结果

基于Tomasulo算法的指令调度

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		R(F4)	Load2	
	Mult2	No					

Register result status:

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

- **Load2执行结束，可广播其结果**

基于Tomasulo算法的指令调度

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)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

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

- **Add1，Mult1可同时执行**

基于Tomasulo算法的指令调度

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)	R(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仍然可以发射，尽管F6存在名字相关**

基于Tomasulo算法的指令调度

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)	R(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			

- **Add1 (SUBD) 执行结束，谁在等待其结果?**

基于Tomasulo算法的指令调度

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)	R(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			

基于Tomasulo算法的指令调度

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)	R(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			

基于Tomasulo算法的指令调度

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)	R(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			

- **Add2 (ADDD) 执行结束**

基于Tomasulo算法的指令调度

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)	R(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			

- **Write result of ADDD here?**
- **All quick instructions complete in this cycle!**

基于Tomasulo算法的指令调度

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)	R(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+N	(M-M)	Mult2			

基于Tomasulo算法的指令调度

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)	R(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+N	(M-M)	Mult2			

基于Tomasulo算法的指令调度

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)	R(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+N	(M-M)	Mult2			

基于Tomasulo算法的指令调度

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)	R(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+N	(M-M)	Mult2			

- **Mult1 (MULTD) completing; what is waiting for it?**

基于Tomasulo算法的指令调度

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+N	(M-M)	Mult2			

- **Just waiting for Mult2 (DIVD) to complete**

基于Tomasulo算法的指令调度

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+N	(M-M)	Mult2			

基于Tomasulo算法的指令调度

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 **56**

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

- **Mult2 (DIVD) is completing; what is waiting for it?**

基于Tomasulo算法的指令调度

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					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock 57

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

- **Once again: In-order issue, out-of-order execution and out-of-order completion.**

Tomasulo算法的优势

- 分布式的冒险检测
  - 分布式的保留栈：每个保留栈可以自主判断冒险
  - CDB的使用：一个结果被多个功能单元等待，这些功能单元能同时得到该结果的广播消息
    - 如果结果保存在寄存器中，寄存器只能由一个功能单元访问
- 能够消除WAW、WAR冒险
  - 利用保留栈可实现寄存器的重命名
  - WAR：一旦数据从寄存器读到保留栈，则不再依赖该寄存器
  - WAW：两条指令的目标寄存器相同，但他们的结果由不同的功能单元等待，仍然不会发生WAW
    - 只要按需提交到寄存器组即可

Tomasulo算法的劣势

- The major drawback of this approach is the complexity of the Tomasulo scheme
  - requires a large amount of hardware
  - reservation station must contain an associative buffer, which must run at high speed, as well as complex control logic
  - The performance can also be limited by the single CDB, which interacts with each RS

Imprecise Exceptions

- **Dynamically scheduled processors could generate imprecise exceptions:**
  - The pipeline may have already completed instructions that are later in program order than the instruction causing the exception

```
fadd.d f10,f0,f8
```

  - The pipeline may have not yet completed some instructions that are earlier in program order than the instruction causing the exception

```
fadd.d f10,f0,f8
```

以循环为例的进一步分析

```
LOOP:   L.D      F0,0(R1)
        MUL.D    F4,F0,F2
        S.D      0(R1),F4
        DADDUI   R1,R1,#-8
        BNE      R1,R2,LOOP
```

假定循环无限展开，或者分支每次都选择执行下一次循环迭代

以循环为例的进一步分析

- 指令发射情况
  - 两个迭代的指令都发射了
  - 但是只有两条Load指令正在执行

Instruction		Instruction status			
		From iteration	Issue	Execute	Write result
L.D	F0,0(R1)	1	√	√	
MUL.D	F4,F0,F2	1	√		
S.D	F4,0(R1)	1	√		
L.D	F0,0(R1)	2	√	√	
MUL.D	F4,F0,F2	2	√		
S.D	F4,0(R1)	2	√		

以循环为例的进一步分析

- 功能单元和寄存器的使用情况

	Reservation stations						
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	yes	Load					Regs[R1]+0
Load2	yes	Load					Regs[R1]-8
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL		Regs[F2]	Load1		
Mult2	yes	MUL		Regs[F2]	Load2		
Store1	yes	Store	Regs[R1]			Mult1	
Store2	yes	Store	Regs[R1]-8			Mult2	

	Register status								
Field	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
Qi	Load2		Mult2						

以循环为例的进一步分析

- Load、Store、Add等指令各自判断冒险
  - 无集中控制单元
- Load与Store的地址已经保存在保留栈中
  - 关于R1的WAR冒险不会发生
- 两条Load指令的目标寄存器都是F0，但是不会发生WAW
  - 等待两条Load指令结果的功能单元分别是Mult1和Mult2
  - Load指令执行完，直接将结果送到对应的Mult单元，绕过F0

以循环为例的进一步分析

- 第二次迭代的Load可以在第一次迭代的Store之前执行
  - Tomasulo算法可判断出它们的目标地址不一样，没有RAW
- Load指令执行时
  - 在Store的保留栈中检查是否有相同地址的Store指令未完成
- Store指令执行时
  - 在Store和Load保留栈中检查是否有相同地址的指令未完成

CPI接近于1

Tomasulo算法总结

- 基于保留栈的动态调度
  - 允许发射很多指令，只要保留栈够用
  - 允许将寄存器的旧值保存在保留栈，避免WAR冒险
- 基于保留栈的寄存器重命名
  - 解决寄存器不够用的问题
- 解决访存时的地址冲突问题
  - 发射Load、Store指令时，需检查是否有关于访存的WAW、WAR
- Tomasulo算法在代码执行过程中建立数据流依赖关系图
- Tomasulo算法适用于
  - 寄存器较少的体系结构
  - 代码难以调度的场景

内容纲要

- 指令级并行简介
- 基于循环展开的指令调度
- 基于计分板的指令调度
- 基于Tomasulo算法的指令调度