计算机体系结构

周学海

xhzhou@ustc.edu.cn

0551-63601556, 63492271

中国科学技术大学

第5章 指令级并行

- 指令集并行的基本概念及挑战
- 软件方法挖掘指令集并行
  - 基本块内的指令集并行
- 硬件方法挖掘指令集并行
  - Tomasulo方法
- 跨越基本块的指令集并行
- 基于硬件的推测执行
- 以多发射和静态调度来挖掘指令集并行
- 以动态调度、多发射和推测执行来挖掘指令集并行

并行及并行体系结构

- 应用程序中的并行:
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)

- 软硬件挖掘应用程序的DLP或TLP的方式
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

Review: 基本流水线

- 流水线提高的是指令带宽（吞吐率），而不是单条指令的执行速度
- 相关限制了流水线性能的发挥
  - 结构相关：需要更多的硬件资源
  - 数据相关：需要定向，编译器调度
  - 控制相关：尽早检测条件，计算目标地址，延迟转移，预测
- 增加流水线的级数会增加相关产生的可能性
- 异常，浮点运算使得流水线控制更加复杂
- 编译器可降低数据相关和控制相关的开销
  - Load 延迟槽
  - Branch 延迟槽
  - Branch预测

5.1 指令级并行的基本概念及挑战

- ILP: 无关的指令重叠执行
- 流水线的平均CPI

  Pipeline CPI = Ideal Pipeline CPI + Struct Stalls + RAW Stalls + WAR Stalls + WAW Stalls + Control Stalls
- 本章研究：减少停顿（stalls)数的方法和技术
- 基本途径
  - 软件方法：
    - Gcc: 17%控制类指令，5 instructions + 1 branch；
    - 在基本块上，得到更多的并行性
    - 挖掘循环级并行
  - 硬件方法
    - 动态调度方法
  - 以DLX（MIPS）的浮点数操作为例

采用的基本技术

Technique	Reduces	Section
Forwarding and bypassing	Potential data hazard stalls	A.2
Delayed branches and simple branch scheduling	Control hazard stalls	A.2
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences	A.8
Dynamic scheduling with renaming	Data hazard stalls and stalls from antidependences and output dependences	3.2
Dynamic branch prediction	Control stalls	3.4
Issuing multiple instructions per cycle	Ideal CPI	3.6
Speculation	Data hazard and control hazard stalls	3.5
Dynamic memory disambiguation	Data hazard stalls with memory	3.2, 3.7
Loop unrolling	Control hazard stalls	4.1
Basic compiler pipeline scheduling	Data hazard stalls	A.2, 4.1
Compiler dependence analysis	Ideal CPI, data hazard stalls	4.4
Software pipelining, trace scheduling	Ideal CPI, data hazard stalls	4,3
Compiler speculation	Ideal CPI, data, control stalls	4.4

本章遵循的指令延时

产生结果的指令	*使用结果的指令*	*所需延时*
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**
Integer op	**Integer op**	**0**

(当使用结果的指令为BRANCH指令时除外)

5.2 基本块内的指令级并行

- 基本块的定义：
  - 直线型代码，无分支；单入口；程序由分支语句连接基本块构成

- 循环级并行

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

- 计算x(i)时没有相关；可以并行产生1000个数据；这里没有相关是指没有数据相关
- 问题：在生成代码时会有Branch指令—控制相关
- 预测比较容易，但我们必须有预测方案

  - 向量处理机模型

- load vectors x and y (up to some machine dependent max)
- then do *result-vec* = *xvec* + *yvec* in a single instruction

简单循环及其对应的汇编程序

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

```
Loop:   LD      F0,0(R1)    ;F0=vector element
        ADDD    F4,F0,F2    ;add scalar from F2
        SD      0(R1),F4    ;store result
        SUBI    R1,R1,8     ;decrement pointer 8B (DW)
        BNEZ    R1,Loop     ;branch R1!=zero
        NOP                 ;delayed branch slot
```

FP 循环中的Stalls

```
1 Loop:  LD     F0,0(R1)    ;F0=vector element
2        stall
3        ADDD   F4,F0,F2    ;add scalar in F2
4        stall
5        stall
6        SD     0(R1),F4    ;store result
7        SUBI   R1,R1,8     ;decrement pointer 8B (DW)
8        stall                ;
9        BNEZ   R1,Loop     ;branch R1!=zero
10       stall              ;delayed branch slot
```

产生结果的指令	*使用结果的指令*	*所需的延时*
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
Integer op	Integer op	0

10 clocks: 是否可以通过调整代码顺序使stalls减到最小

FP 循环中的最少Stalls数

```
1 Loop:LD      F0,0(R1)
2       SUBI    R1,R1,8
3       ADDD    F4,F0,F2
4       stall
5       BNEZ    R1,Loop          ;delayed branch
6       SD    8(R1),F4           ;altered when move past SUBI
```

Swap BNEZ and SD by changing address of SD

6 clocks: 通过循环展开4次是否可以提高性能?

循环展开4次(straightforward way)

```
1 Loop: LD     F0,0(R1)    stall
2       ADDD   F4,F0,F2    stall    stall
3       SD     0(R1),F4             ;drop SUBI & BNEZ
4       LD     F6,-8(R1)   stall
5       ADDD   F8,F6,F2    stall stall
6       SD     -8(R1),F8            ;drop SUBI & BNEZ
7       LD     F10,-16(R1) stall
8       ADDD   F12,F10,F2  stall stall
9       SD     -16(R1),F12          ;drop SUBI & BNEZ
10      LD     F14,-24(R1) stall
11      ADDD   F16,F14,F2  stall  stall
12      SD     -24(R1),F16
13      SUBI   R1,R1,#32   stall    ;alter to 4*8
14      BNEZ   R1,LOOP
15      NOP
```

Rewrite loop to minimize stalls?

15 + 4 x (1+2) + 1 = 28 cycles, or 7 per iteration

Assumes R1 is multiple of 4

Stalls数最小的循环展开

```
1 Loop:  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 clock cycles, or 3.5 per iteration

- 代码移动后
  - SD移动到SUBI后，注意偏移量的修改
  - Loads移动到SD前，注意偏移量的修改

循环展开示例小结

- 循环展开对循环间无关的程序是有效降低stalls的手段(对循环级并行）.
- 指令调度，必须保证程序运行的结果不变
- 注意循环展开中的Load和Store,不同次循环的Load 和Store是相互独立的。需要分析对存储器的引用，保证他们没有引用同一地址.
- 不同次的循环，使用不同的寄存器
- 删除不必要的测试和分支后，需调整循环步长等控制循环的代码.
- 移动SD到SUBI和BNEZ后，需要调整 SD中的偏移

从编译器角度看代码移动（1/5）

- 编译器分析程序的相关性依赖于给定的流水线
- 编译器进行指令调度来消除相关
- （True）数据相关（Data dependencies）
  - 对于指令i和j，如果Instruction j使用指令i产生的结果 ，或Instruction j 与instruction k相关，并且instruction k 与instruction i有数据相关.
- 如果相关，不能并行执行
- 对于寄存器比较容易确定(fixed names)
- 但对memory的引用，比较难确定:
  - 100(R4) = 20(R6)?
  - 在不同次的循环中，20(R6) = 20(R6)?

下列程序哪里有数据相关?

```
1 Loop:   LD    F0,0(R1)
2         ADDD  F4,F0,F2
3         SUBI  R1,R1,8
4         BNEZ  R1,Loop     ;delayed branch
5         SD    8(R1),F4    ;altered when move past SUBI
```

从编译器角度看代码移动(2/5)

- 另一种相关称为名相关（ name dependence）：
  两条指令使用同一个名子(register or memory location) 但不交换数据
  - 反相关（Antidependence） (WAR if a hazard for HW)
    - Instruction j 所写的寄存器或存储单元，与 instruction i 所读的寄存器或存储单元相同，注instruction i 是先执行
  - 输出相关(Output dependence) (WAW if a hazard for HW)
    - Instruction i 和instruction j 对同一寄存器或存储单元进行写操作，必须保证两条指令的写顺序

下列是否有名相关?

```
1 Loop: LD     F0,0(R1)
2       ADDD   F4,F0,F2
3       SD     0(R1),F4        ;drop SUBI & BNEZ
4       LD     F0,-8(R1)
5       ADDD   F4,F0,F2
6       SD     -8(R1),F4       ;drop SUBI & BNEZ
7       LD     F0,-16(R1)
8       ADDD   F4,F0,F2
9       SD     -16(R1),F4      ;drop SUBI & BNEZ
10      LD     F0,-24(R1)
11      ADDD   F4,F0,F2
12      SD     -24(R1),F4
13      SUBI   R1,R1,#32       ;alter to 4*8
14      BNEZ   R1,LOOP
15      NOP
```

如何消除名相关?

从编译器角度看代码移动（3/5）

- 访问存储单元时，很难判断名相关
  - 100(R4) = 20(R6)?
  - 不同次的循环，20(R6) = 20(R6)?
- 我们给出的示例要求编译器知道假设R1不变，因此:

0(R1) ≠ -8(R1) ≠ -16(R1) ≠ -24(R1)

因此loads和stores之间相互无关可以移动

从编译器角度看代码移动（4/5）

- 最后一种相关称为控制相关(control dependence)
- Example

```
if p1 {S1;};
if p2 {S2;};
```

S1 依赖于P1的测试结果，S2依赖于P2的测试。

从编译器角度看代码移动（5/5）

- 处理控制相关的原则：
  - 受分支指令控制的指令，不能移到控制指令之前，以免该指令的执行不在分支指令的控制范围.
  - 不受分支指令控制的指令，不能移到控制指令之后，以免该指令的执行受分支指令的控制.
- 减少控制相关可以提高指令的并行性

下列程序段的控制相关

```
1 Loop:   LD     F0,0(R1)
2         ADDD   F4,F0,F2
3         SD     0(R1),F4
4         SUBI   R1,R1,8
5         BEQZ   R1,exit
6         LD     F0,0(R1)
7         ADDD   F4,F0,F2
8         SD     0(R1),F4
9         SUBI   R1,R1,8
10        BEQZ   R1,exit
```

```
11    LD     F0,0(R1)
12    ADDD   F4,F0,F2
13    SD     0(R1),F4
14    SUBI   R1,R1,8
15    BEQZ   R1,exit
....
```

循环展开（1/3）

Example: 下列程序段存在哪些数据相关?
(A,B,C 指向不同的存储区且不存在覆盖区)

```
for (i=1; i<=100; i=i+1) {
        A[i+1] = A[i] + C[i];        /* S1 */
        B[i+1] = B[i] + A[i+1];      /* S2 */
}
```

1. S2使用由S1在同一循环计算出的 A[i+1].

2. S1 使用由S1在前一次循环中计算的值，同样S2也使用由S2在前一次循环中计算的值. 这种存在于循环间的相关，我们称为“loop-carried dependence”这表示循环间存在相关，不能并行执行，它与我们前面的例子中循环间无关是有区别的

循环展开 (2/3)

- Example:A,B,C,D distinct & nonoverlapping

```
for (i=1; i<=100; i=i+1) {
    A[i] = A[i] + B[i];    /* S1 */
    B[i+1] = C[i] + D[i];} /* S2 */
```

1. S1和S2没有相关，S1和S2互换不会影响程序的正确性
2. 在第一次循环中，S1依赖于前一次循环的B[1].

循环展开（3/3）

OLD:

```
for (i=1; i<=100; i=i+1) {
        A[i] = A[i] + B[i];    /* S1 */
        B[i+1] = C[i] + D[i];} /* S2 */
```

NEW:

```
A[1] = A[1] + B[1];
for          (i=1; i<=99; i=i+1) {
    B[i+1] = C[i] + D[i];
    A[i+1] = A[i+1] + B[i+1];
}
B[101] = C[100] + D[100];
```

Review

- 指令级并行(ILP)
- 流水线的平均CPI

  Pipeline CPI =

  Ideal Pipeline CPI

  + Struct Stalls + RAW Stalls + WAR Stalls + WAW Stalls + Control Stalls
- 提高指令级并行的方法
  - 软件方法：指令流调度，循环展开，软件流水线，trace scheduling
  - 硬件方法
- 循环展开
  - 指令调度，必须保证程序运行的结果不变
  - 偏移量的修改
  - 寄存器的重命名
  - 循环步长的调整

5.3 硬件方案：指令级并行

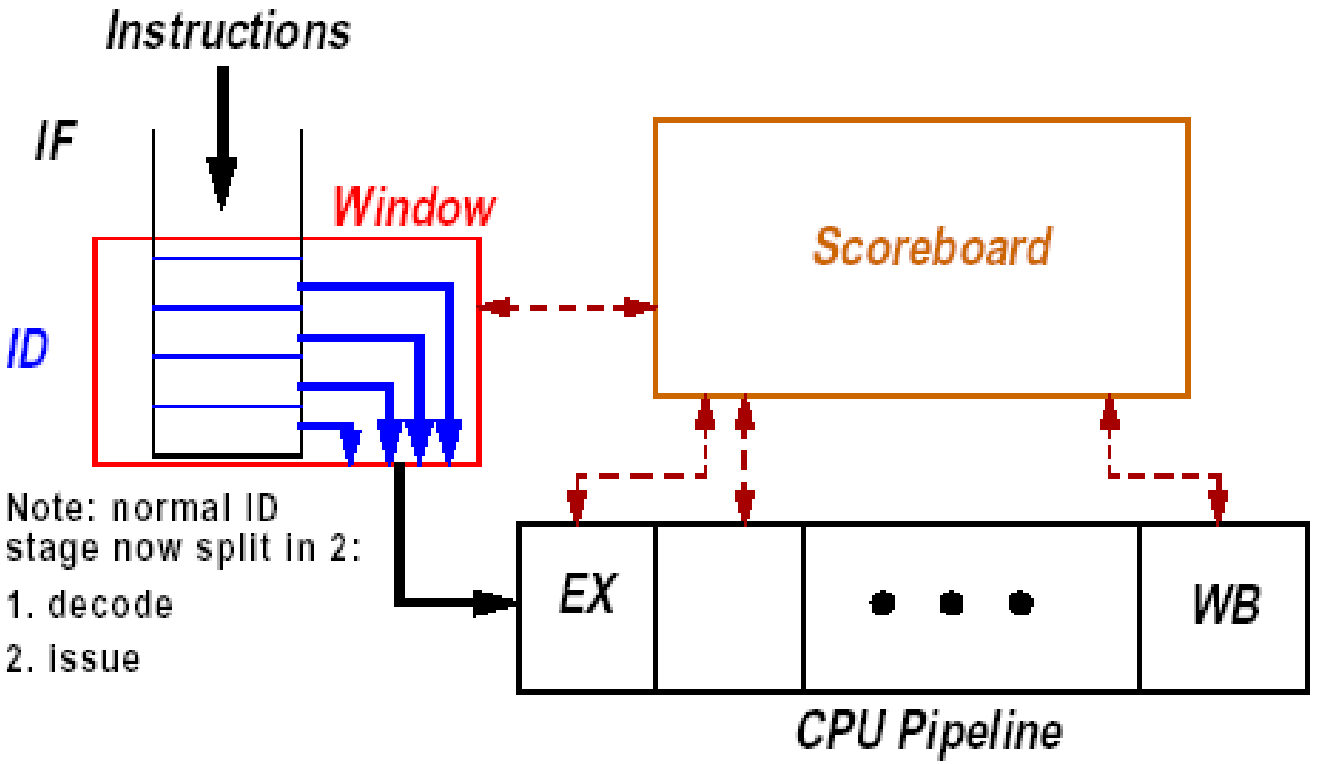
- 为什么要使用硬件调度方案?
  - 在编译时无法确定的相关，可以通过硬件调度来优化
  - 编译器简单
  - 代码在不同组织结构的机器上，同样可以有效的运行
- 基本思想: 允许 stall后的指令继续向前流动

```
DIVD F0,F2,F4
ADDD F10,F0,F8
SUBD F12,F8,F14
```

  - 允许乱序执行（out-of-order execution） => out-of-order completion

硬件方案之一：记分牌

- 记分牌的基本概念示意图



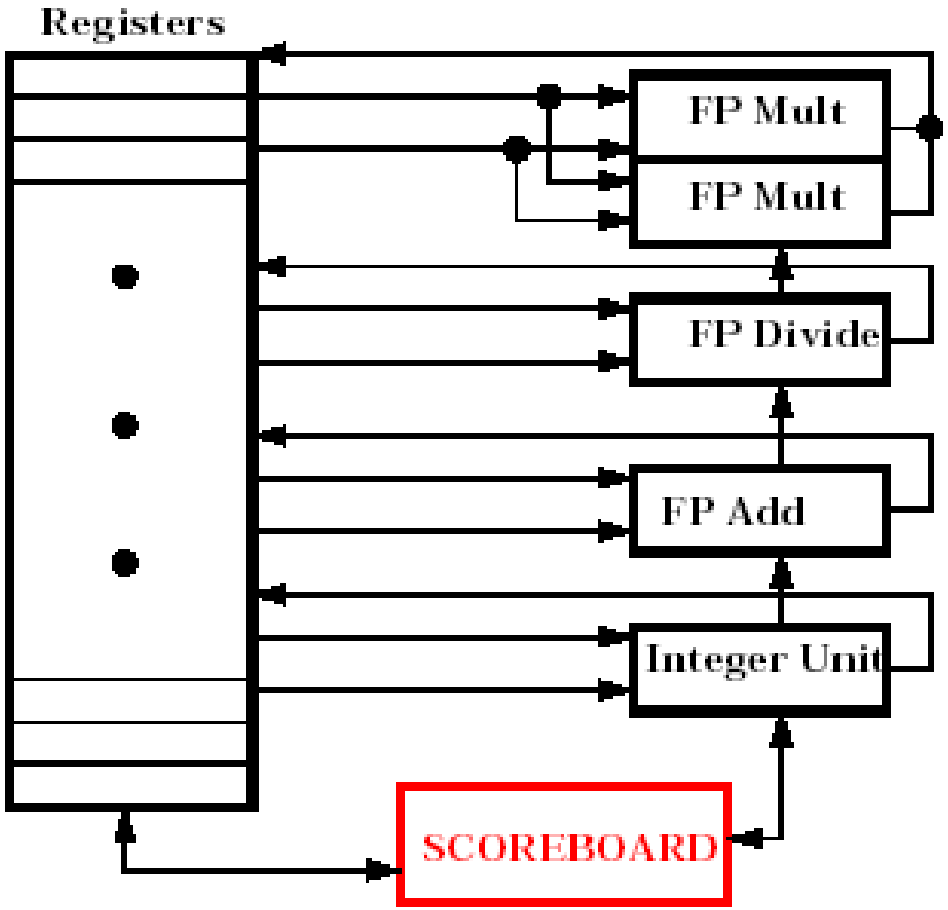
记分牌技术要点（1/2）

- Out-of-order execution 将ID 段分为:
  1. Issue—译码，检测结构相关
  2. Read operands—等待到无数据相关时，读操作数
- 起源于1963年推出的CDC6600
  - 4 FPU
  - 5 Memory Reference
  - 7 IU
- 集中相关检查，互锁机制解决相关
- CDC 6600: 顺序发射，乱序执行，乱序完成，CDC6600流水线没有采用定向技术，只实现非精确中断
- Load /store结构
- 采用这种技术的微处理器企业
  - MIPS，HP， IBM
  - Sun 公司的UltraSparc
  - DEC Alpha

记分牌技术要点(2/2)

- Out-of-order completion => WAR, WAW hazards?
- WAR的一般解决方案
  - 对操作排队
  - 仅在读操作数阶段读寄存器
- 对WAW而言, 检测到相关后，停止发射前一条指令，直到前一条指令完成
- 要提高效率，需要有多条指令进入执行阶段=>必须有多个执行部件或执行部件是流水化的
- 记分牌保存相关操作和状态
- 记分牌用四段代替ID, EX, WB 三段

带有记分牌控制的DLX



Note: this model could support both single or multi-issue

Exception is that one multiply will be issued per cycle

All depends on bus/trunk structure

记分牌控制的四阶段（1/2）

1. Issue—指令译码，检测结构相关

   如果当前指令所使用的功能部件空闲，并且没有其他活动的指令使用相同的目的寄存器（WAW), 记分牌发射该指令到功能部件，并更新记分牌内部数据，如果有结构相关或WAW相关，则该指令的发射暂停，并且也不发射后继指令，直到相关解除.

2. Read operands—没有数据相关时，读操作数

   如果先前已发射的正在运行的指令不对当前指令的源操作数寄存器进行写操作，或者一个正在工作的功能部件已经完成了对该寄存器的写操作，则该操作数有效。操作数有效时，记分牌控制功能部件读操作数，准备执行。

   记分牌在这一步动态地解决了RAW相关，指令可能会乱序执行。

记分牌控制的四阶段（2/2）

3. Execution—取到操作数后执行 (EX)

   接收到操作数后，功能部件开始执行. 当计算出结果后，它通知记分牌，可以结束该条指令的执行.

4. Write result—finish execution (WB)

   一旦记分牌得到功能部件执行完毕的信息后，记分牌检测WAR相关，如果没有WAR相关，就写结果，如果有WAR 相关，则暂停该条指令。

   Example:

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

   CDC 6600 scoreboard 将暂停 SUBD 直到ADDD 读取操作数后，才进入WR段处理。

记分牌的结构

1. Instruction status—记录正在执行的各条指令处于四步中的哪一步
2. Functional unit status—记录功能部件(FU)的状态。用9个域记录每个功能部件的9个参量：

   Busy—指示该部件是否空闲

   Op—该部件所完成的操作

   Fi—其目的寄存器编号

   Fj, Fk—源寄存器编号

   Qj, Qk—产生源操作数Fj, Fk的功能部件

   Rj, Rk—标识源操作数Fj, Fk是否就绪的标志

3. Register result status—如果存在功能部件对某一寄存器进行写操作，指示具体是哪个功能部件对该寄存器进行写操作。如果没有指令对该寄存器进行写操作，则该域 为Blank

记分牌流水线控制

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rk(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

*

Scoreboard Example

Instruction status

Instruction		j	k	Issue	Read operanc	Executi complet	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTI	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 for j Qj	FU for k 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									

Scoreboard Example: Cycle 1

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					

Scoreboard Example: Cycle 2

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					

- Issue 2nd LD?

Scoreboard Example: Cycle 3

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								

- Issue MULT?

Scoreboard Example: Cycle 4

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								

Scoreboard Example: Cycle 5

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							

Scoreboard Example: Cycle 6

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							

Scoreboard Example: Cycle 7

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				

- Read multiply operands?

Scoreboard Example: Cycle 8a

(First half of clock cycle)

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				

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			

Scoreboard Example: Cycle 8b

(Second half of clock cycle)

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			

Scoreboard Example: Cycle 9

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			

- Read operands for MULT & SUB? Issue ADDD?

Scoreboard Example: Cycle 10

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			

Scoreboard Example: Cycle 11

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			

Scoreboard Example: Cycle 12

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			

- **Read operands for DIVD?**

Scoreboard Example: Cycle 13

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			

Scoreboard Example: Cycle 14

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			

Scoreboard Example: Cycle 15

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			

Scoreboard Example: Cycle 16

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			

Scoreboard Example: Cycle 17

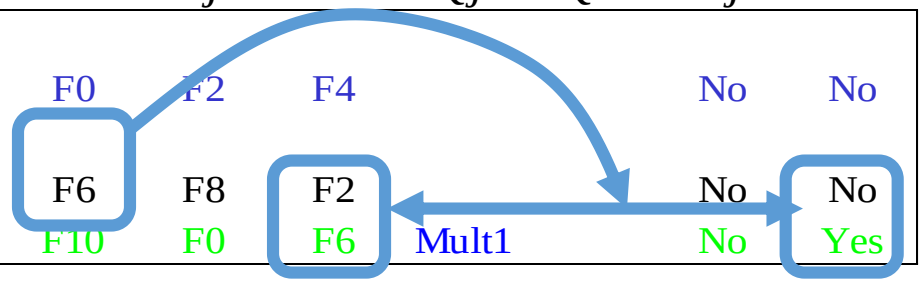
Instruction status:

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	

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			

- Why not write result of ADD???

Scoreboard Example: Cycle 18

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			

Scoreboard Example: Cycle 19

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			

Scoreboard Example: Cycle 20

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			

Scoreboard Example: Cycle 21

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			

- **WAR Hazard is now gone...**

Scoreboard Example: Cycle 22

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			

Continue…….

Scoreboard Example: Cycle 61

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			

Scoreboard Example: Cycle 62

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*									

Review: Scoreboard Example: Cycle 62

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*									

- In-order issue; out-of-order execute & commit

CDC 6600 Scoreboard

- 编译器优化，加速比可达到1.7，手工优化加速比可达到2.5，其存储系统较慢 (no cache)限制了性能的发挥
- 6600 scoreboard的缺陷:
  - 没有定向数据通路
  - 指令窗口较小，仅局限于基本块内的调度
  - 功能部件数较少，容易产生结构相关，特别是其Load store操作也是用IU部件完成的
  - 结构冲突时不能发射
  - WAR相关是通过等待解决的
  - WAW相关时，不会进入IS阶段

ILP小结

- 可通过软件或硬件来挖掘指令级并行潜力
- 循环级并行是最容易判断的
- 程序内在的相关性限制了用软件方法挖掘程序的并行性
- 编译器的数据相关性分析结果，确定了是否可以进行循环展开
  - 当对存储器单元引用时，数据相关分析很困难
- 硬件方法挖掘ILP
  - 在编译阶段无法确定的相关性，可以在程序执行时，用硬件方法判定
  - 这种方法还可以使得程序代码在其他机器上有效地执行
- 记分牌的主要思想是：允许stall后的指令继续进行处理
  - 可以out-of-order execution => out-of-order completion
  - 发射前检测结构相关和WAW相关
  - 写结果前处理WAR相关

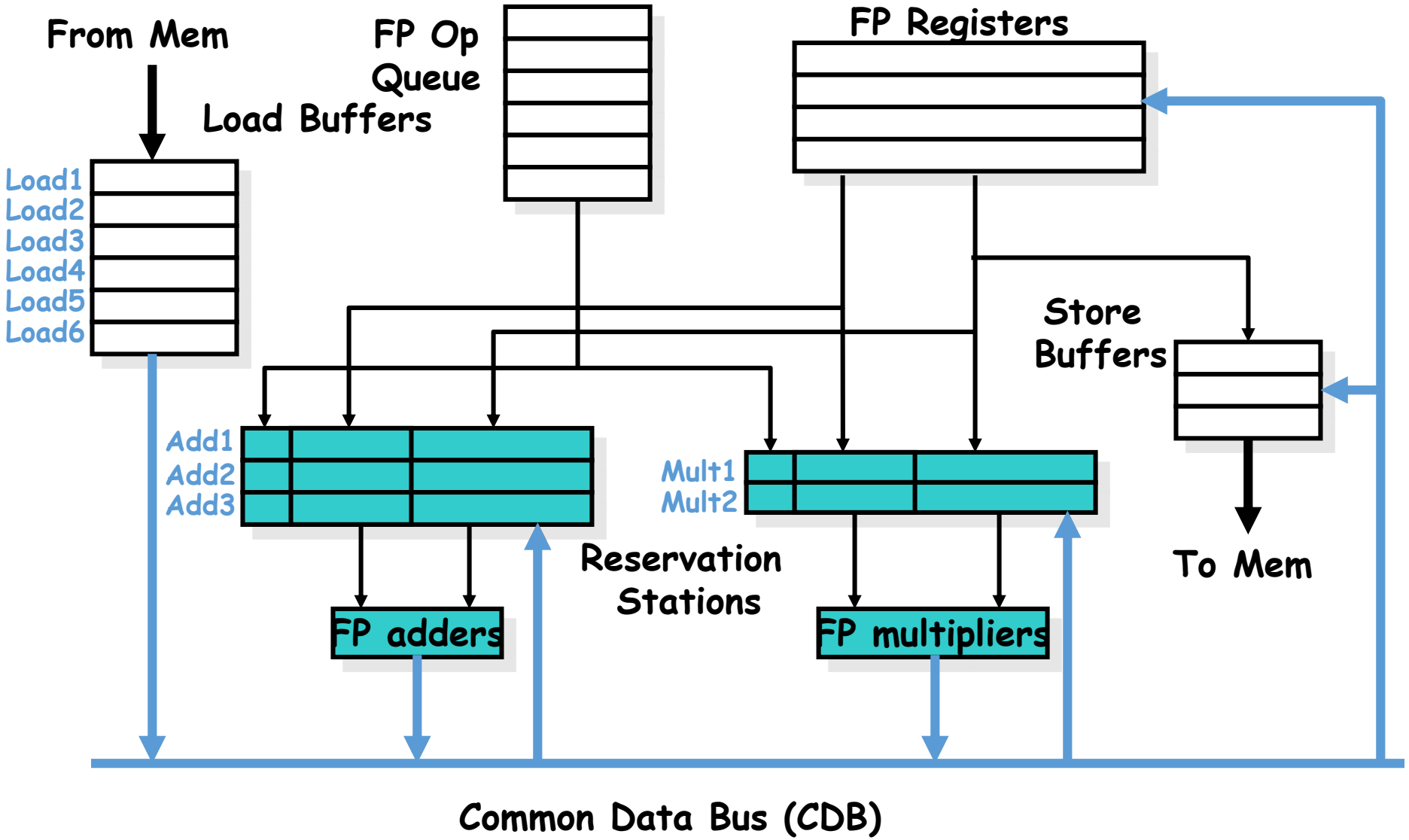
动态调度方案之二：Tomasulo Algorithm

- 该算法首次在 IBM 360/91上使用（ CDC6600推出三年后）
- 目标: 在没有专用编译器的情况下，提高系统性能
- IBM 360 & CDC 6600 ISA的差别
  - IBM360只有 2位寄存器描述符 vs. CDC 6600寄存器描述符3位
  - IBM360　4个FP 寄存器 vs. CDC 6600 8个
  - IBM 360 有memory-register 操作
- 为什么研究? Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604，…

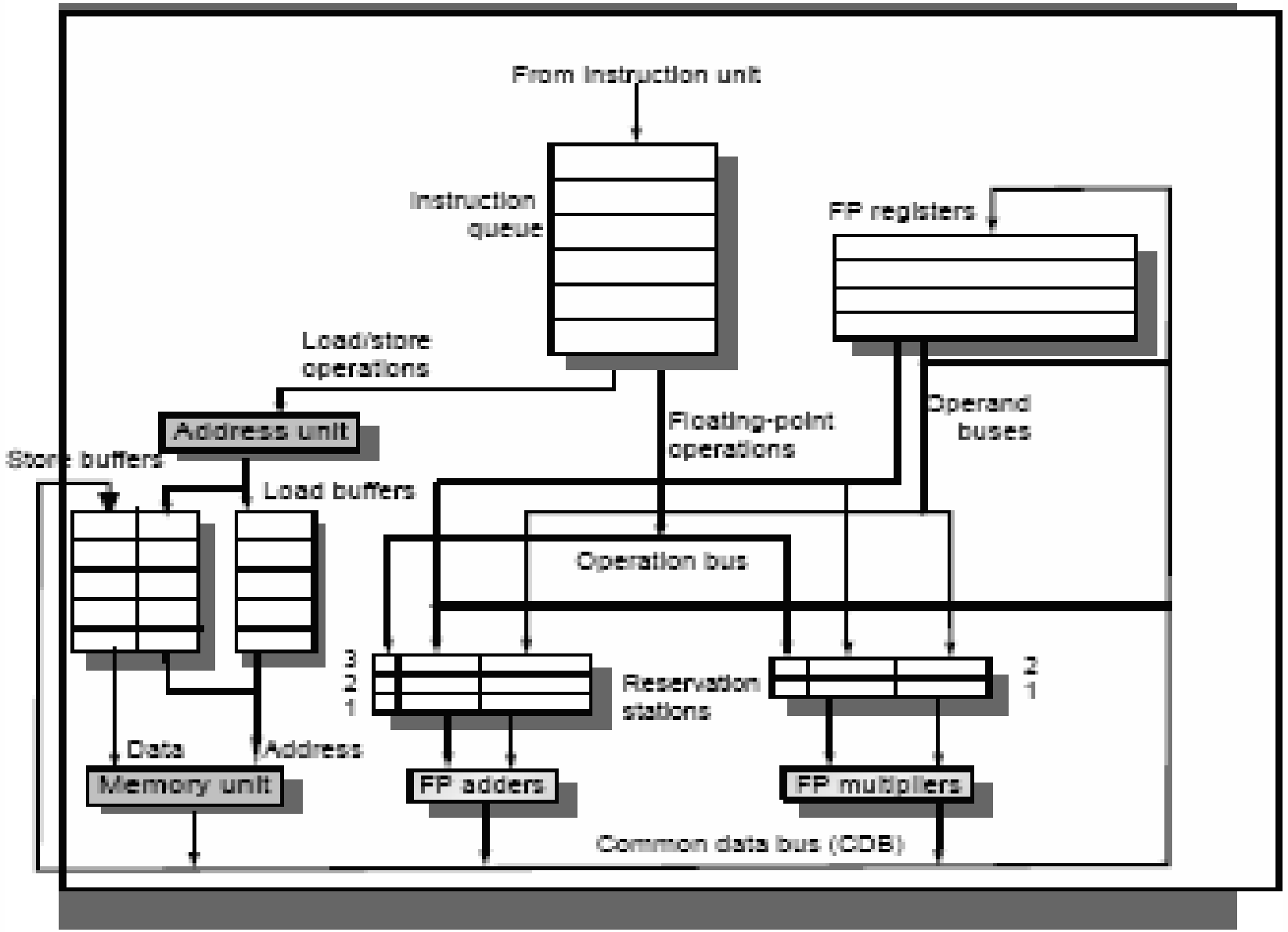
Tomasulo Algorithm vs. Scoreboard

- 控制和缓存分布在各部件中 vs. 控制和缓存集中在记分牌
  - FU 缓存称为 “reservation stations”; 保存待用操作数
- 指令中的寄存器在RS中用寄存器值或指向RS的指针代替。称为 register renaming ;
  - 避免 WAR, WAW hazards
  - RS多于寄存器，因此可以做更多编译器无法做的优化
- 传给FU的结果，从RS来而不是从寄存器来，FU的计算结果通过 Common Data Bus 以广播方式发向所有的功能部件。
- Load和Store部件也看作带有RS的功能部件
- 可以跨越分支，允许FP操作队列中的FP操作不仅仅局限于基本块

Tomasulo Organization



Tomasulo Organization (cont.)



Reservation Station 结构

Op: 部件所进行的操作

Vj, Vk: 源操作数的值

Store 缓冲区有Vk域，用于存放要写入存储器的值

A: 用于存放存储器地址。开始存立即数，计算出有效地址后，存放有效地址

Qj, Qk: 产生源操作数的RS

注：没有记分牌中的准备就绪标志， Qj, Qk=0 => ready

Store 缓存区只有Qk表示产生结果的RS

Busy: 标识RS或FU是否空闲

Register result status—如果存在对寄存器的写操作，指示对该寄存器进行写操作的部件.

Qi: 保留站的编号

Tomasulo 算法的三阶段

1. Issue—从FP操作队列中取指令

   如果RS空闲(no structural hazard), 则控制发射指令和操作数 (renames registers).
   消除WAR，WAW相关

2. Execution—operate on operands (EX)

   当两操作数就绪后，就可以执行
   如果没有准备好，则监测Common Data Bus 以获取结果。通过推迟指令执行避免RAW相关

3. Write result—finish execution (WB)

   将结果通过Common Data Bus传给所有等待该结果的部件;
   表示RS可用

- 通常的数据总线: data + destination (“go to” bus)
- Common data bus: data + source (“come from” bus)
  - 64 bits 数据线 + 4位功能部件源地址（ FU source address）
  - 产生结果的部件如果与RS中等待的部件匹配，就进行写操作
  - 广播方式传送

Tomasulo 算法流水线控制

1. **Issue**

FP Operation:

Wait until : Station r empty

Action or bookkeeping:

if(RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi}

else {RS[r].Vj ←Reg[rs]; RS[r].Qj ←0 }

if(RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi}

else {RS[r].Vk ←Reg[rt]; RS[r].Qk←0 }

RS[r].Busy ← yes; RegisterStat[rd].Qi = r;

Load or Store:

Wait until: Buffer r empty

Action or bookkeeping:

if(RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi}

else {RS[r].Vj ←Reg[rs]; RS[r].Qj ←0 }

RS[r].A ←imm; RS[r].Busy ← yes;

Load only: RegisterStat[rt].Qi = r;

Store only:

if(RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi}

else {RS[r].Vk ←Reg[rt]; RS[r].Qk←0 }

rs, rt : 源寄存器名；**rd:**目的寄存器名
RS: 保留站数据结构；**r:**保留站编号
RegisterStat: 寄存器状态表
Reg: 寄存器堆

注意：Load操作在EXE阶段分两步

2、Execute

FP Operation

wait until: (RS[r].Qj=0) and (RS[r].Qk=0)

Action or bookkeeping:

computer result: Operands are in Vj and Vk

Load-store step1

wait until: RS[r].Qj =0 & r is head of load-store queue

Action or bookkeeping:

RS[r].A ← RS[r].Vj + RS[r].A;

Load step2

wait until: Load Step1 complete

Action or bookkeeping: Read from Mem[RS[r].A]

3、Write result

FP Operation or Load

Wait until: Execution complete at r & CDB available

Action or bookkeeping

∀x (if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0})

∀ x (if(RS[x].Qj =r) {RS[x].Vj ←result; RS[x].Qj ←0});

∀ x (if(RS[x].Qk =r) {RS[x].Vk ←result; RS[x].Qk ←0});

RS[r].Busy ← no;

Store

wait until: Execution complete at r & RS[r].Qk = 0

Action or bookkeeping

Mem[RS[r].A] ← RS[r].Vk;

RS[r].Busy ← no;

Tomasulo Example

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	

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*
0	*FU*									

Tomasulo Example 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					

Tomasulo Example 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					

Note: Unlike 6600, can have multiple loads outstanding

Tomasulo Example 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		R(F4)	Load2	
	Mult2	No					

Register result status:

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

- Note: registers names are removed ("renamed") in Reservation Stations; MULT issued vs. scoreboard
- Load1 completing; what is waiting for Load1?

Tomasulo Example 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		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 completing; what is waiting for Load1?

Tomasulo Example 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)	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			

Tomasulo Example 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)	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			

- Issue ADDD here vs. scoreboard?

Tomasulo Example 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)	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 completing; what is waiting for it?

Tomasulo Example 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)	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			

Tomasulo Example 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)	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			

Tomasulo Example 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)	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 completing; what is waiting for it?

Tomasulo Example 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)	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 vs. scoreboard?
- All quick instructions complete in this cycle!

Tomasulo Example 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)	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+M	(M-M)	Mult2			

Tomasulo Example 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)	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+M	(M-M)	Mult2			

Tomasulo Example 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)	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+M	(M-M)	Mult2			

Tomasulo Example 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)	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+M	(M-M)	Mult2			

Tomasulo Example 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			

Faster than light computation
(skip a couple of cycles)

Tomasulo Example 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			

Tomasulo Example 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			

- Mult2 is completing; what is waiting for it?

Tomasulo Example 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	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			

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

Compare to Scoreboard Cycle 62

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

Issue	*Exec Comp*	*Write Result*
1	3	4
2	4	5
3	15	16
4	7	8
5	56	57
6	10	11

- 为什么scoreboard/6600所需时间较长?
  - 结构冲突
  - 没有定向技术

Tomasulo v. Scoreboard (IBM 360/91 v. CDC 6600)

	流水化的功能部件	多个功能部件
	(6 load, 3 store, 3 +, 2 x/÷)	(1 load/store, 1 + , 2 x, 1 ÷)
指令窗口大小:	~ 14 instructions	~ 5 instructions
	有结构冲突时不发射	相同
WAR:	用寄存器重命名避免	stall 来避免
WAW:	用寄存器重命名避免	停止发射
	从FU广播结果	写寄存器方式
	Control: RS	集中式scoreboard

review

- Tomasulo Algorithm 三阶段

1. Issue—从FP操作队列中取指令

   如果RS空闲(no structural hazard), 则控制发射指令和操作数 (renames registers).

2. Execution—operate on operands (EX)

   当两操作数就绪后，就可以执行
   如果没有准备好，则监测Common Data Bus 以获取结果

3. Write result—finish execution (WB)

   将结果通过Common Data Bus传给所有等待该结果的部件;
   表示RS可用

- 通常的数据总线: data + destination (“go to” bus)
- Common data bus: data + source (“come from” bus)
  - 64 bits 数据线 + 4位功能部件源地址（ FU source address）
  - 产生结果的部件如果与RS中等待的部件匹配，就进行写操作
  - 广播方式传送

Tomasulo 缺陷

- 复杂
  - delays of 360/91, MIPS 10000, IBM 620?
- 要求高速CDB
- 性能受限于Common Data Bus

Tomasulo Loop Example

```
Loop:   LD      F0   0(R1)
        MULTD   F4   F0   F2
        SD      F4   0(R1)
        SUBI    R1   R1   #8
        BNEZ    R1   Loop
```

- 设Multiply执行阶段4 clocks
- 第一次load 需8 clocks (cache miss), 第2次以后假设命中(hit)
- 为清楚起见，下面我们也列出SUBI, BNEZ的时钟周期

Loop Example

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1			
1	MULTD	F4	F0	F2			
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

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

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	**80**	*Fu*									

Loop Example Cycle 1

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2			
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	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					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
1	**80**	*Fu*	Load1								

Loop Example Cycle 2

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	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(F2)	Load1	
	Mult2	No					

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	**80**	*Fu*	Load1		Mult1						

Loop Example Cycle 3

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

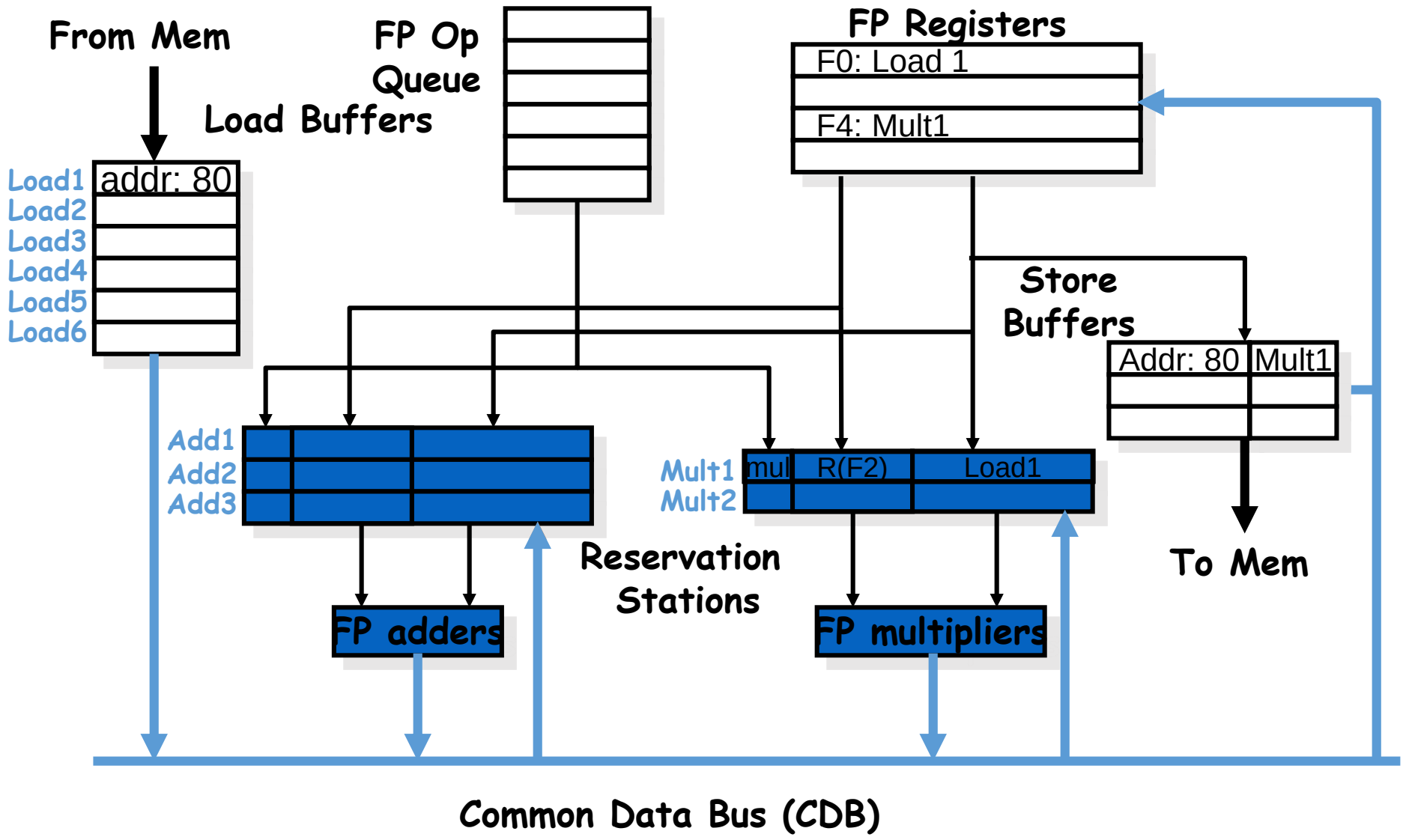
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	**80**	*Fu*	Load1		Mult1						

- Implicit renaming sets up “DataFlow” graph

What does this mean physically?



Loop Example Cycle 4

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	**80**	*Fu*	Load1		Mult1						

- Dispatching SUBI Instruction

Loop Example Cycle 5

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	**72**	*Fu*	Load1		Mult1						

- And, BNEZ instruction

Loop Example Cycle 6

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	**72**	*Fu*	Load2		Mult1						

- Notice that F0 never sees Load from location 80

Loop Example Cycle 7

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1			

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	**72**	*Fu*	Load2		Mult2						

- Register file completely detached from iteration 1

Loop Example Cycle 8

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

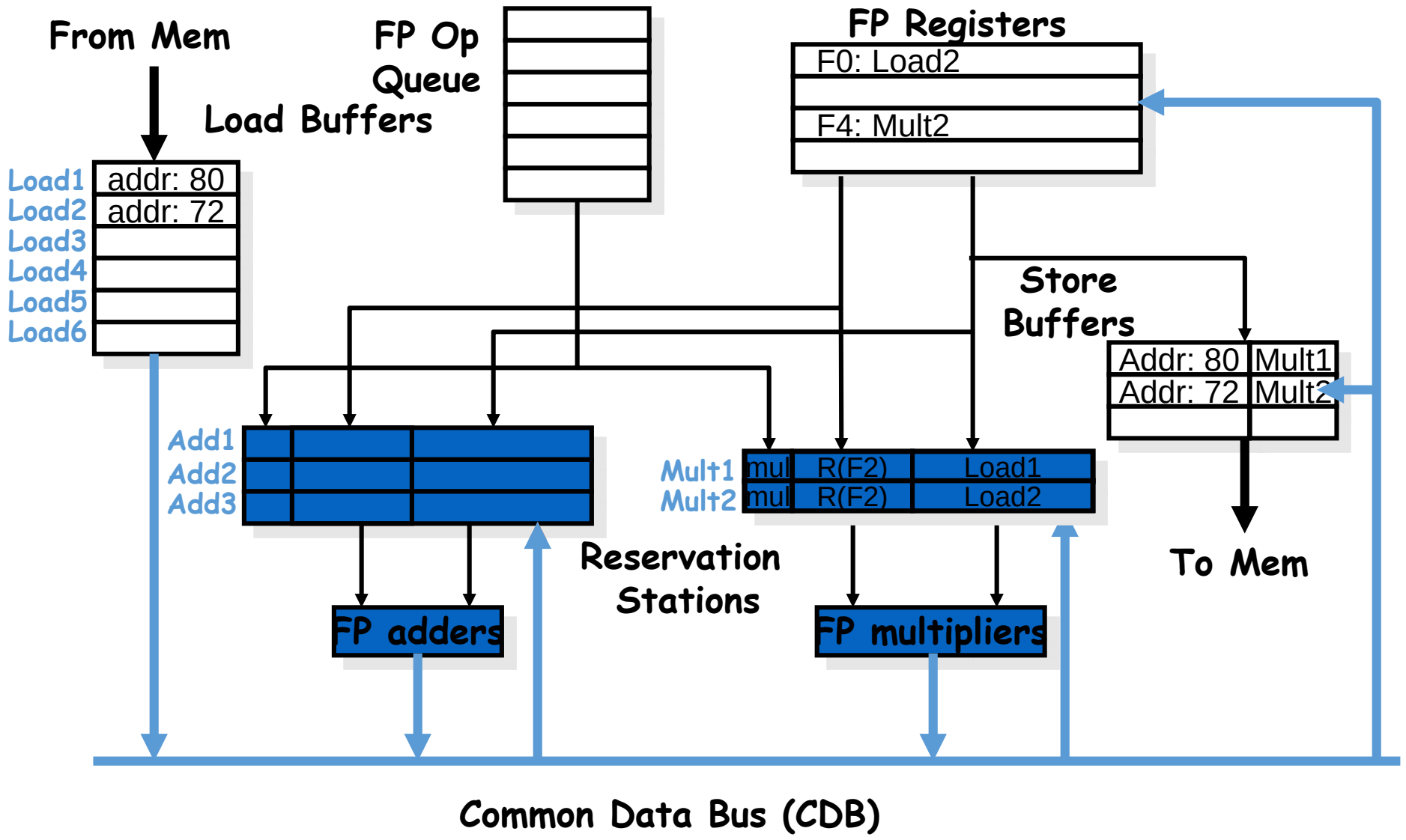
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	**72**	*Fu*	Load2		Mult2						

- First and Second iteration completely overlapped

What does this mean physically?



Loop Example Cycle 9

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
9	**72**	*Fu*	Load2		Mult2						

- Load1 completing: who is waiting?
- Note: Dispatching SUBI

Loop Example Cycle 10

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd	M[80]	R(F2)		
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	**64**	*Fu*	Load2		Mult2						

- Load2 completing: who is waiting?
- Note: Dispatching BNEZ

Loop Example Cycle 11

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	Multd	M[80]	R(F2)		
4	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	**64**	*Fu*	Load3		Mult2						

- Next load in sequence

Loop Example Cycle 12

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	Multd	M[80]	R(F2)		
3	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	**64**	*Fu*	Load3		Mult2						

- Why not issue third multiply?

Loop Example Cycle 13

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	Multd	M[80]	R(F2)		
2	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
13	**64**	*Fu*	Load3		Mult2						

Loop Example Cycle 14

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	Multd	M[80]	R(F2)		
1	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	**64**	*Fu*	Load3		Mult2						

- Mult1 completing. Who is waiting?

Loop Example Cycle 15

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Fu	Load3		Mult2						

- Mult2 completing. Who is waiting?

Loop Example Cycle 16

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1* *Vk*	*S2* *Qj*	*RS* *Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
16	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 17

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
17	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 18

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
18	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 19

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
19	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 20

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	20

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
20	**64**	*Fu*	Load3		Mult1						

为何Tomasulo可以循环覆盖执行？

- 寄存器重命名技术
  - 不同的循环使用不同的物理寄存器 (dynamic loop unrolling).
  - 将代码中的静态寄存器名修改为动态寄存器指针 “pointers”
  - 有效地增加了寄存器文件的大小
- 关键: 整数部件必须先行，以便能发射多个循环中的操作

Summary#1/2

- Reservations stations: 寄存器重命名，缓冲源操作数
  - 避免寄存器成为瓶颈
  - 避免了Scoreboard中无法解决的 WAR, WAW hazards
  - 允许硬件做循环展开
- 不限于基本块(IU先行，解决控制相关)
- 贡献
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 后 Pentium II; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264使用这种技术

Summary #2/2

- 动态硬件方案可以用硬件进行循环展开
- 但如何处理精确中断?
  - Out-of-order execution ⇒ out-of-order completion!
- 如何处理分支?
  - 我们可以用硬件做循环展开必须可以解决分支指令问题

为什么顺序发射?

- 顺序发射使我们可以进行程序的数据流分析
  - 我们可以知道某条指令的结果会流向哪些指令
  - 如果我们乱序发射，可能会混淆RAW和WAR相关
- 每一周期发射多条指令也使用该原则将会正确地工作:
  - 需要多端口的 “rename table” ,以便同时对一组指令所用的寄存器重命名
  - 需要在单周期内发射到多个RS中.
  - 寄存器文件需要有2x 个读端口和x个写端口.

关于异常处理???

- 乱序完成加大了实现精确中断的难度
  - 在前面指令还没有完成时，寄存器文件中可能会有后面指令的运行结果.
  - 如果这些前面的指令执行时有中断产生，怎么办？
  - 例如：DIVD F10, F0, F2

    SUBD F4, F6, F8

    ADDD F12, F14, F16
- 需要“rollback” 寄存器文件到原来的状态:
  - 精确中断的含义是其返回地址为：
    - 该地址之前的所有指令都已完成
    - 其后的指令还都没有完成
- 实现精确中断的技术：顺序完成（或提交）
  - 即提交指令完成的顺序必须与指令发射的顺序相同

进行循环重叠执行需要尽快解决分支问题!

- 在循环展开的例子中，我们假设整数部件可以快速解决分支问题，以便进行循环重叠执行!

```
Loop:  LD      F0   0    R1
       MULTD   F4   F0   F2
       SD      F4   0    R1
       SUBI    R1   R1   #8
       BNEZ    R1   Loop
```

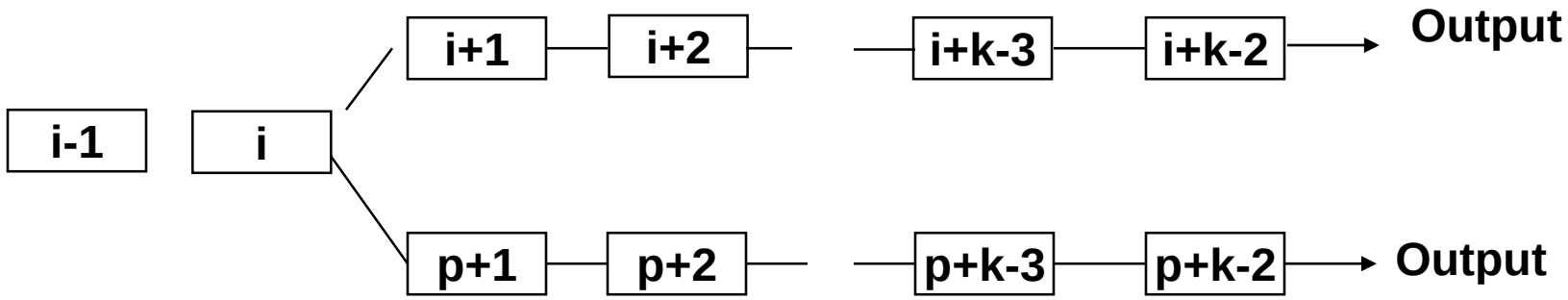
- 如果分支依赖于multd,怎么办??
  - 需要能预测分支方向
  - 如果分支成功，我们就可以重叠执行循环
- 对于superscalar机器这一问题更加突出

控制相关的动态解决技术

- 控制相关：由条件转移或程序中断引起的相关，也称全局相关。控制相关对流水线的吞吐率和效率影响相对于数据相关要大得多
  - 条件指令在一般程序中所占的比例相当大
  - 中断虽然在程序中所占的比例不大，但中断发生在程序中的哪一条指令，发生在一条指令执行过程中的哪一个功能段都是不确定的
- 处理好条件转移和中断引起的控制相关是很重要的。
- 关键问题：
  - 要确保流水线能够正常工作
  - 减少因断流引起的吞吐率和效率的下降

分支对性能的影响

- 假设在一条有K段的流水线中，在最后一段才能确定目标地址



- 当分支方向预测错误时，不仅流水线中有多个功能段要浪费，更严重的是可能造成程序执行结果发生错误，因此当程序沿着错误方向运行后，作废这些程序时，一定不能破坏通用寄存器和主存储器的内容。

条件转移指令对流水线性能的影响

- 假设对于一条有K段的流水线，由于条件分支的影响，在最坏情况下，每次条件转移将造成k-1个时钟周期的断流。假设条件分支在一般程序中所占的比例为p, 条件成功的概率为q。试分析分支对流水线的影响。
- 结论：条件转移指令对流水线的影响很大，必须采取相关措施来减少这种影响。
- 预测可以是静态预测“Static” (at compile time) 或动态预测“Dynamic” (at runtime)
  - 例：一个循环供循环10次，它将分支成功9次，1次不成功。
  - 动态分支预测 vs. 静态分支预测，哪个好？

Dynamic Branch Prediction

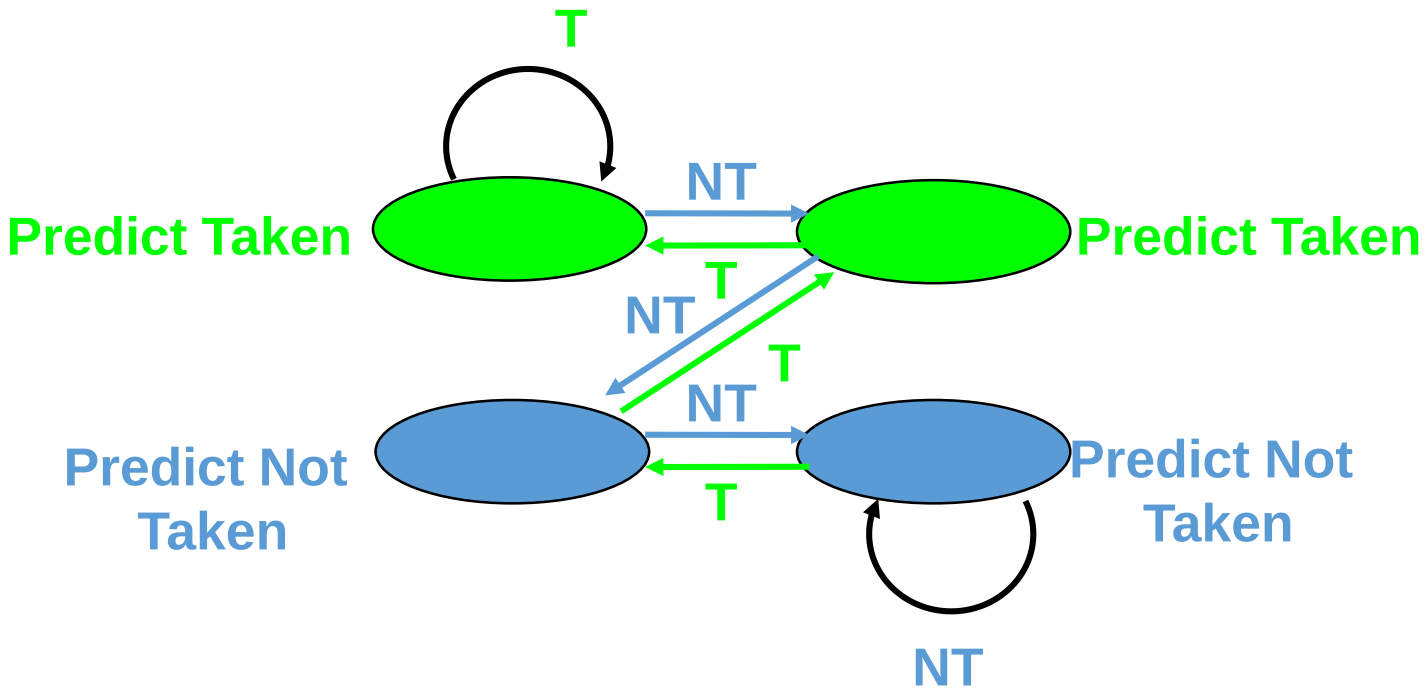
- 动态分支预测：预测分支的方向在程序运行时刻动态确定
- 需解决的关键问题是：
  - 如何记录转移历史信息
  - 如何根据所记录的转移历史信息，预测转移的方向
- 主要方法
  - 基于BPB(Branch Prediction Buffer)或BHT(Branch History Table)的方法
    - 1-bit BHT和2-bit BHT
    - Correlating Branch Predictors
    - Tournament Predictors: Adaptively Combining Local and Global Predictors
  - High Performance Instruction Delivery
    - BTB
    - Integrated Instruction Fetch Units
    - Return Address Predictors
- Performance = ƒ(accuracy, cost of misprediction)
  - Misprediction ⇒ Flush Reorder Buffer

1-bit BHT

- Branch History Table: 分支指令的PC的低位索引1-bit BHT
  - 该表记录上一次转移是否成功
  - 不做地址检查
- 问题: 在一个循环中, 1-bit BHT 将导致2次分支预测错误(avg is 9 iteratios before exit):
  - 最后一次循环, 前面都是预测成功，而这次需要退出循环
  - 首次循环，由于前面预测为失败，而这次实际上为成功

2-bit BHT

- 解决办法: 2位记录分支历史



- Blue: stop, not taken
- Green: go, taken

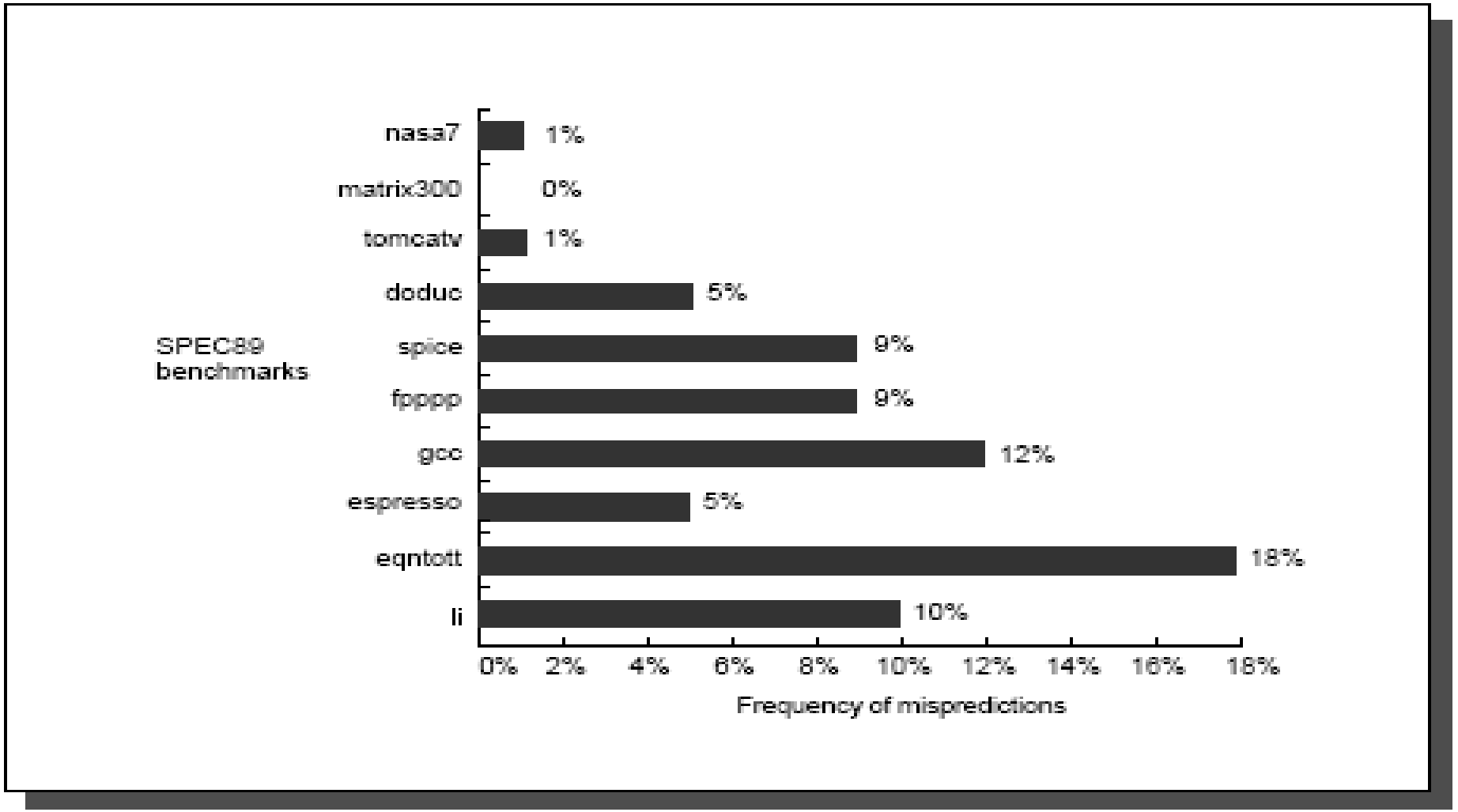


FIGURE 3.8 Prediction accuracy of a 4096-entry two-bit prediction buffer for the SPEC89 benchmarks. The misprediction rate for the integer benchmarks (gcc, espresso, eqntott, and li) is substantially higher (average of 11%) than that for the FP programs (average of 4%). Even omitting the FP kernels (nasa7, matrix300, and tomcatv) still yields a higher accuracy for the FP benchmarks than for the integer benchmarks. These data, as well as the rest of the data in this section, are taken from a branch prediction study done using the IBM Power architecture and optimized code for that system. See Pan et al. [1992].

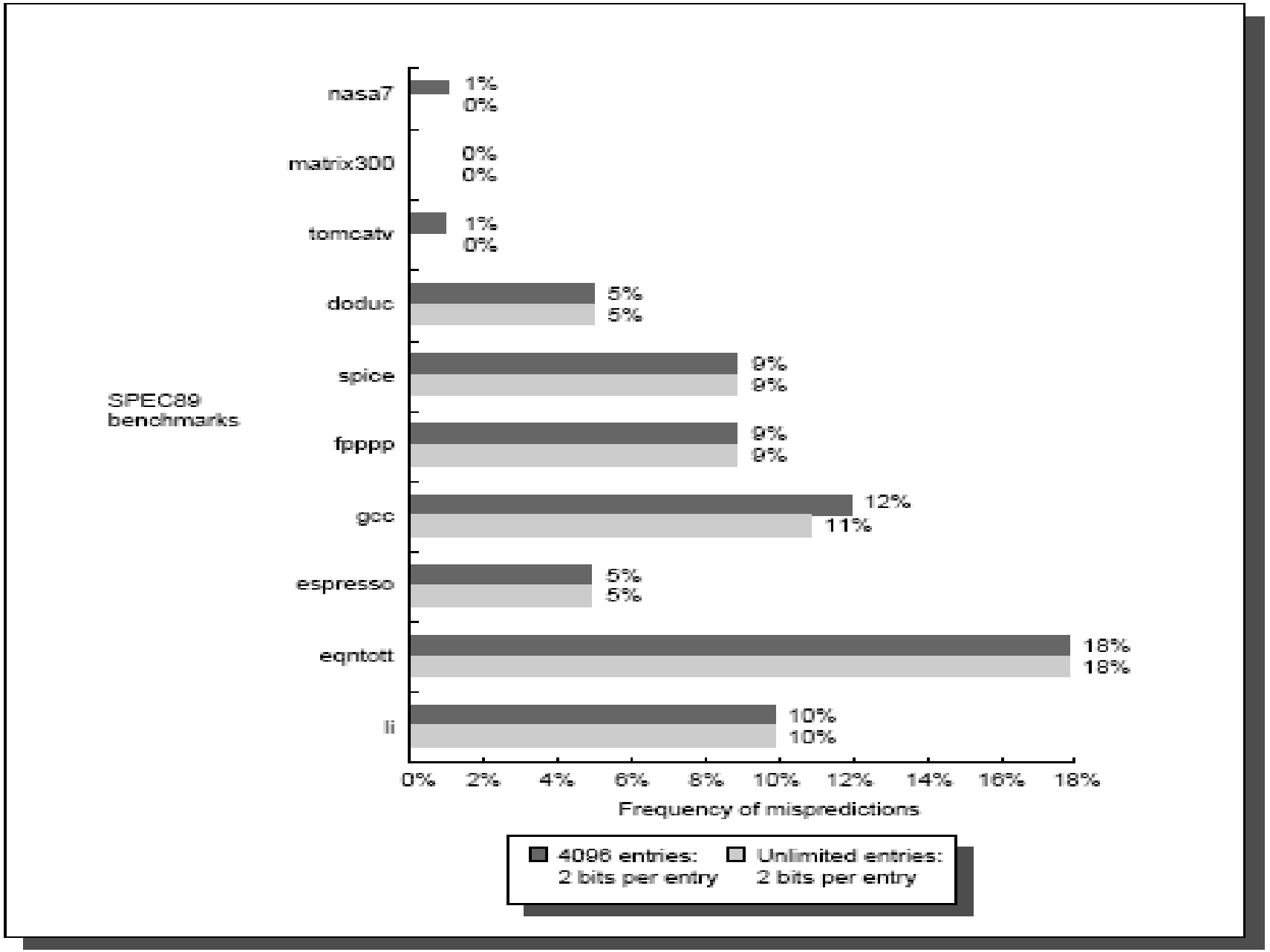


FIGURE 3.9 Prediction accuracy of a 4096-entry two-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.

BHT Accuracy

- 分支预测错误的原因:
  - 预测错误
  - 由于使用PC的低位查找BHT表，可能得到错误的分支历史记录
- BHT表的大小问题
  - 4096 项的表分支预测错误的比例为1% (nasa7, tomcatv) to 18% (eqntott), spice at 9% and gcc at 12%
  - 再增加项数，对提高预测准确率几乎没有效果 (in Alpha 21164)

Correlating Branch Predicator

- 例如：

```
if (aa==2)   aa=0;
if (bb==2)  bb=0;
if (aa!=bb) {
```

- 翻译为DLX

```
     SUBI R3,R1,#2
     BNEZ R3,L1          ; branch b1 (aa!=2)
     ADDI  R1,R0,R0      ;aa=0
L1:  SUBI R3,R2,#2
     BNEZ R3,L2          ;branch b2(bb!=2)
     ADDI R2,R0,R0       ; bb=0
L2:  SUBI R3,R1,R2       ;R3=aa-bb
       BEQZ R3,L3        ;branch b3 (aa==bb)
```

Correlating Branches

- 观察结果：b3 与分支b2 和b1相关。如果b1和b2都分支失败，则b3一定成功。
- Correlating predictors 或 两级预测器：分支预测器根据其他分支的行为来进行预测。
- 工作原理：根据一个简单的例子我们来看其基本原理

```
if (d==0)d=1;
if (d==1) d=0;
```

翻译为DLX

```
    BNEZ R1,L1              ;branch b1(d!=0)
    ADDI R1,R0,#1           ;d==0, so d=1
L1: ADDI R3,R1,#-1
    BNEZ R3,L2              ;branch b2(d!=1)
    ...
```

两级预测器基本工作原理

- 假设d的初始值序列为0，1，2
- b1 如果分支失败，b2一定也分支失败。
- 前面的两位标准的预测方案就没法利用这一点，而两级预测方案就可以。

翻译为DLX

```
        BNEZ R1,L1          ;branch b1(d!=0)
        ADDI R1,R0,#1       ;d==0, so d=1
L1:     ADDI R3,R1,#-1
        BNEZ R3,L2          ;branch b2(d!=1)
```

Initial value of d	d==0?	b1	Value of d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

FIGURE 3.10 Possible execution sequences for a code fragment.

- 假设d的初始值在2和0之间切换。
- 用1-bit预测器，初始设置为预测失败，T表示预测成功，NT表示预测失败。
- 结论：这样的序列每次预测都错，预测错误率100％

```
      BNEZ R1,L1          ;branch b1(d!=0)
      ADDI R1,R0,#1       ;d==0, so d=1
L1:   ADDI R3,R1,#-1
      BNEZ R3,L2          ;branch b2(d!=1)
```

d=?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

FIGURE 3.11 Behavior of a one-bit predictor initialized to not taken. T stands for taken, NT for not taken.

Correlating Branches

- 基本思想：用1位作为correlation位。即每个分支都有两个相互独立的预测位：一个预测位假设最近一次执行的分支失败时的预测位，另一个预测位是假设最近一次执行的分支成功时的预测位。
- 最近一次执行的分支通常与要预测的分支不是同一条指令
- 记为（1，1）前一位表示最近一次分支失败时的预测位，后一位表示最近一次分支成功时的预测位

Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	not taken	not taken
NT/T	not taken	taken
T/NT	taken	not taken
T/T	taken	taken

FIGURE 3.12 Combinations and meaning of the taken/not taken prediction bits. T stands for taken, NT for not taken.

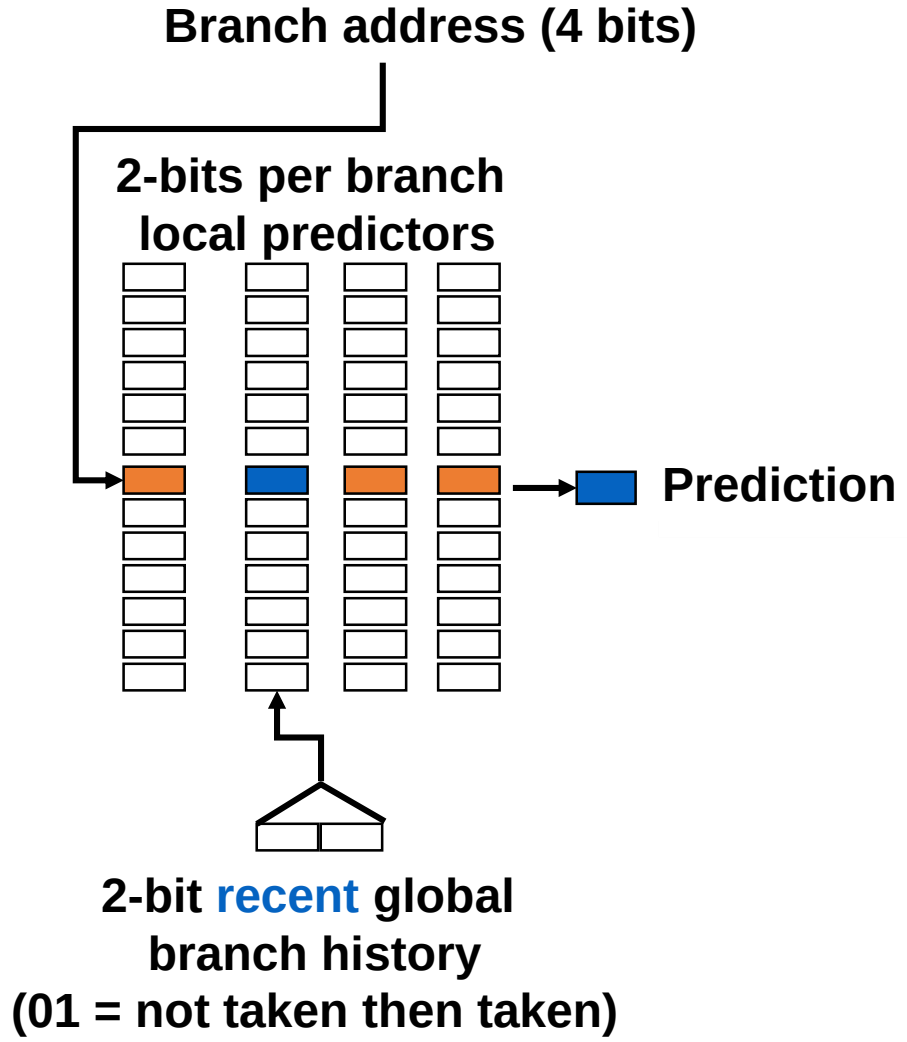
- Correlating 预测器的预测和执行情况，
- 显然只有在第一次d=2时，预测错误，其他都预测正确
- 记为（1，1）预测器，即根据最近一次分支的行为来选择一对1-bit预测器中的一个。
- 更一般的表示为（m, n)，即根据最近的m个分支，从2^m个分支预测器中选择预测器，每个预测器的位数为n

```
        BNEZ R1,L1          ;branch b1(d!=0)
        ADDI R1,R0,#1       ;d==0, so d=1
L1:     ADDI R3,R1,#-1
        BNEZ R3,L2          ;branch b2(d!=1)
```

d=?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	**NT**/NT	T	T/NT	**NT**/NT	T	NT/T
0	**T**/NT	NT	T/NT	**NT**/T	NT	NT/T
2	**T**/NT	T	T/NT	**NT**/T	T	NT/T
0	**T**/NT	NT	T/NT	**NT**/T	NT	NT/T

FIGURE 3.13 The action of the one-bit predictor with one bit of correlation, initialized to not taken/not taken. T stands for taken, NT for not taken. The prediction used is shown in bold.

Correlating Branches



- (2,2) predictor: 2-bit global, 2-bit local

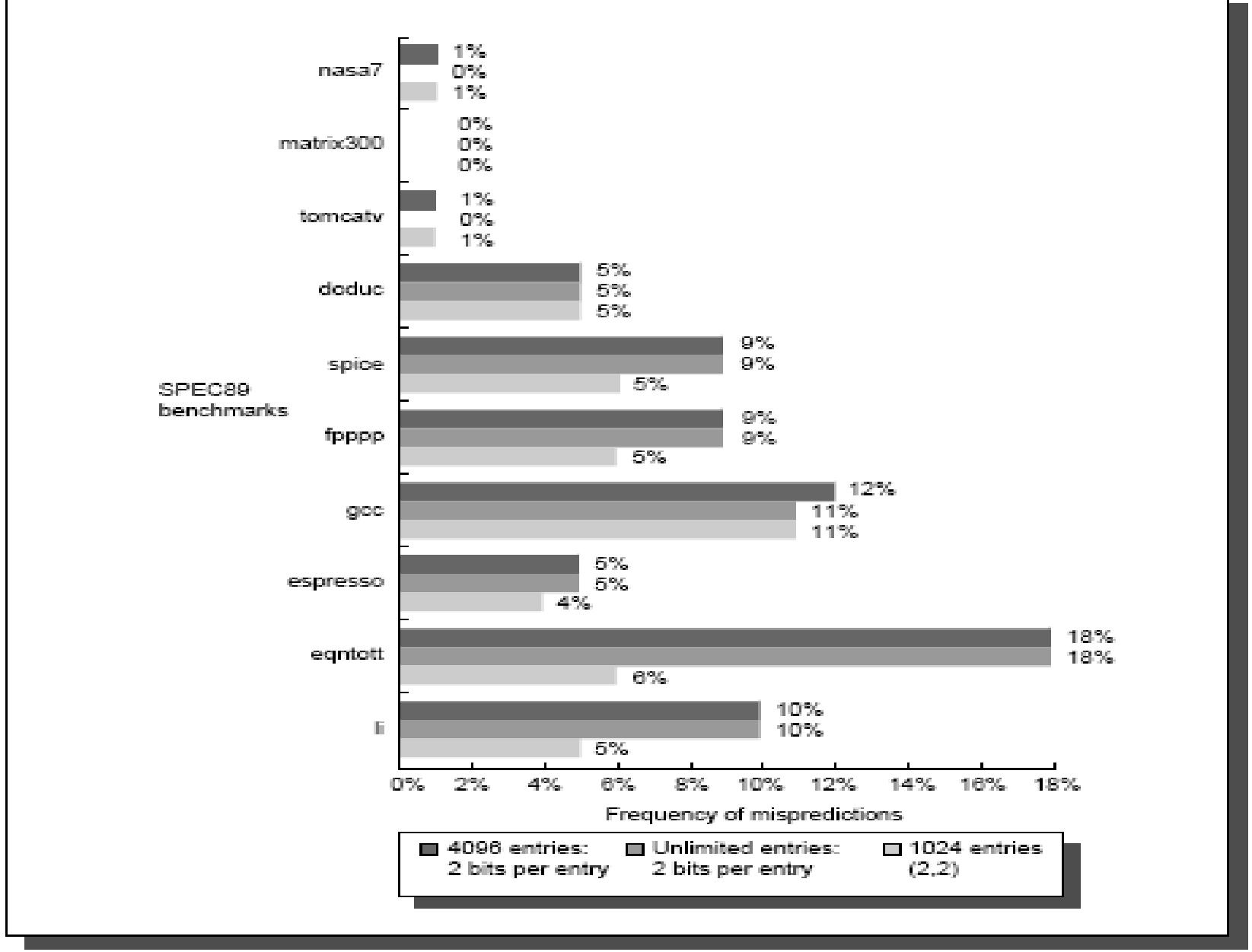
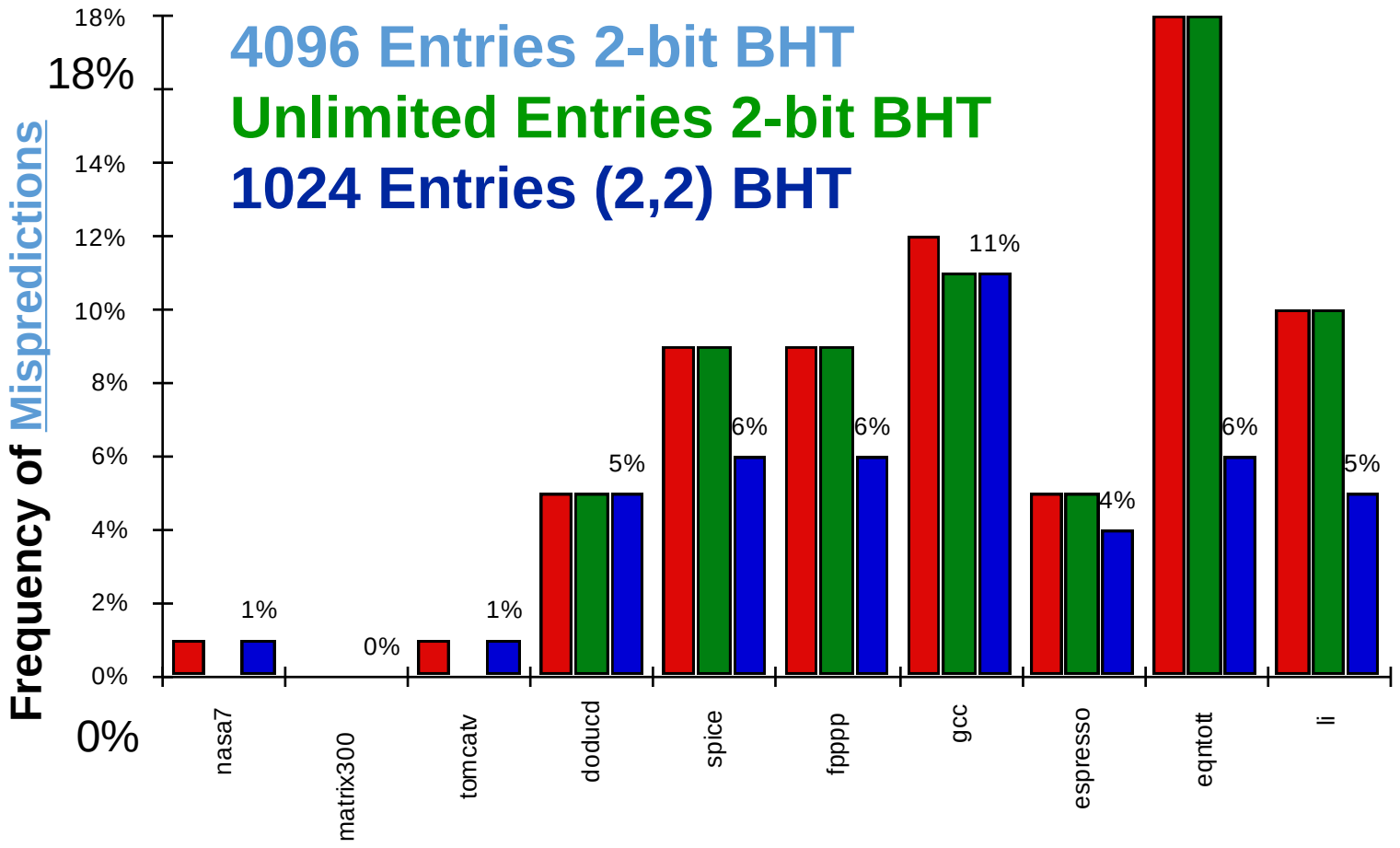


FIGURE 3.15 Comparison of two-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating two-bit predictor with unlimited entries and a two-bit predictor with two bits of global history and a total of 1024 entries.

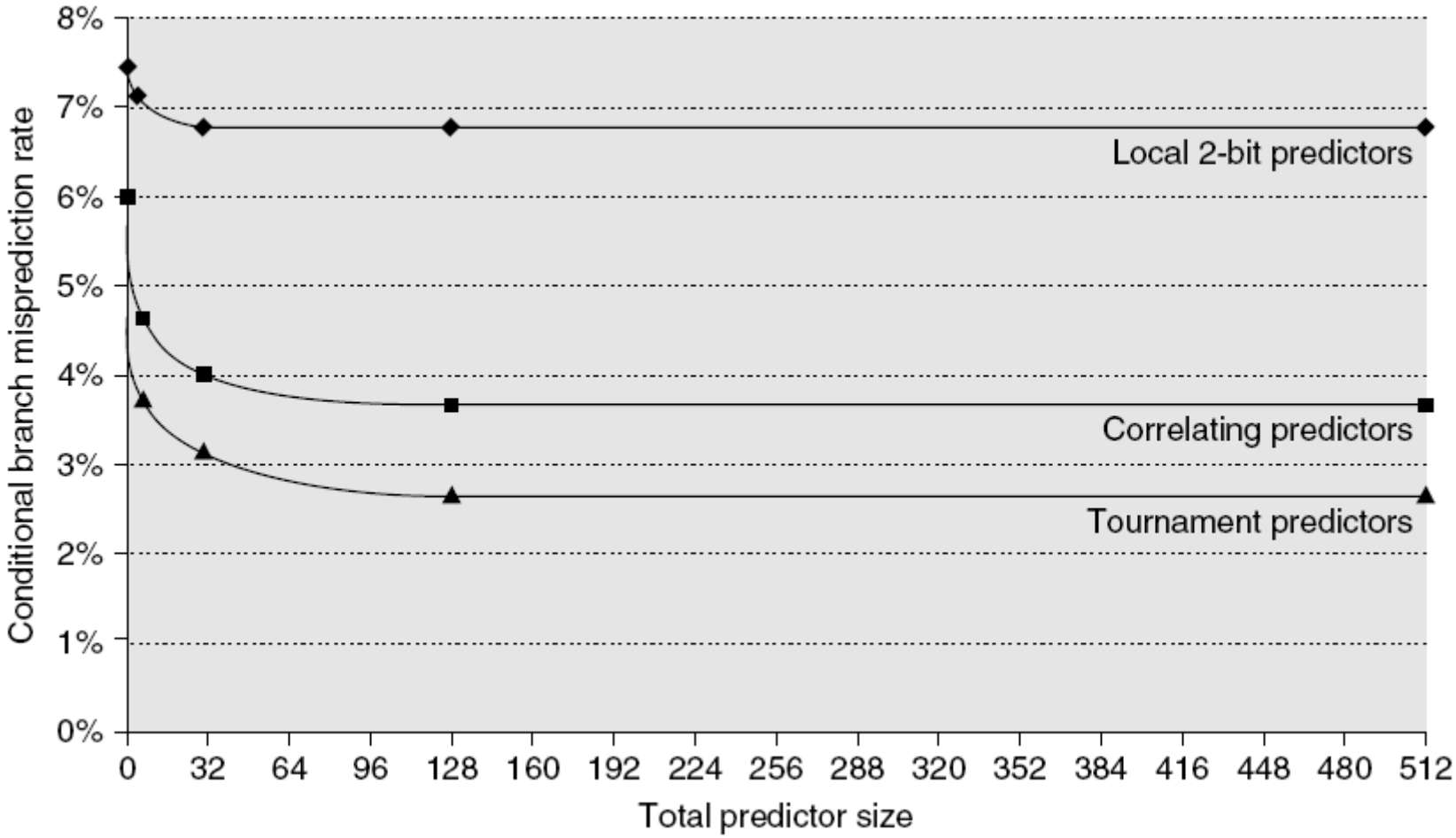
Accuracy of Different Schemes



Branch Prediction

- Basic 2-bit predictor:
- Correlating predictor:
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes of preceding *n* branches
- Local predictor:
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes for the last *n* occurrences of this branch
- Tournament predictor:
  - Combine correlating predictor with local predictor

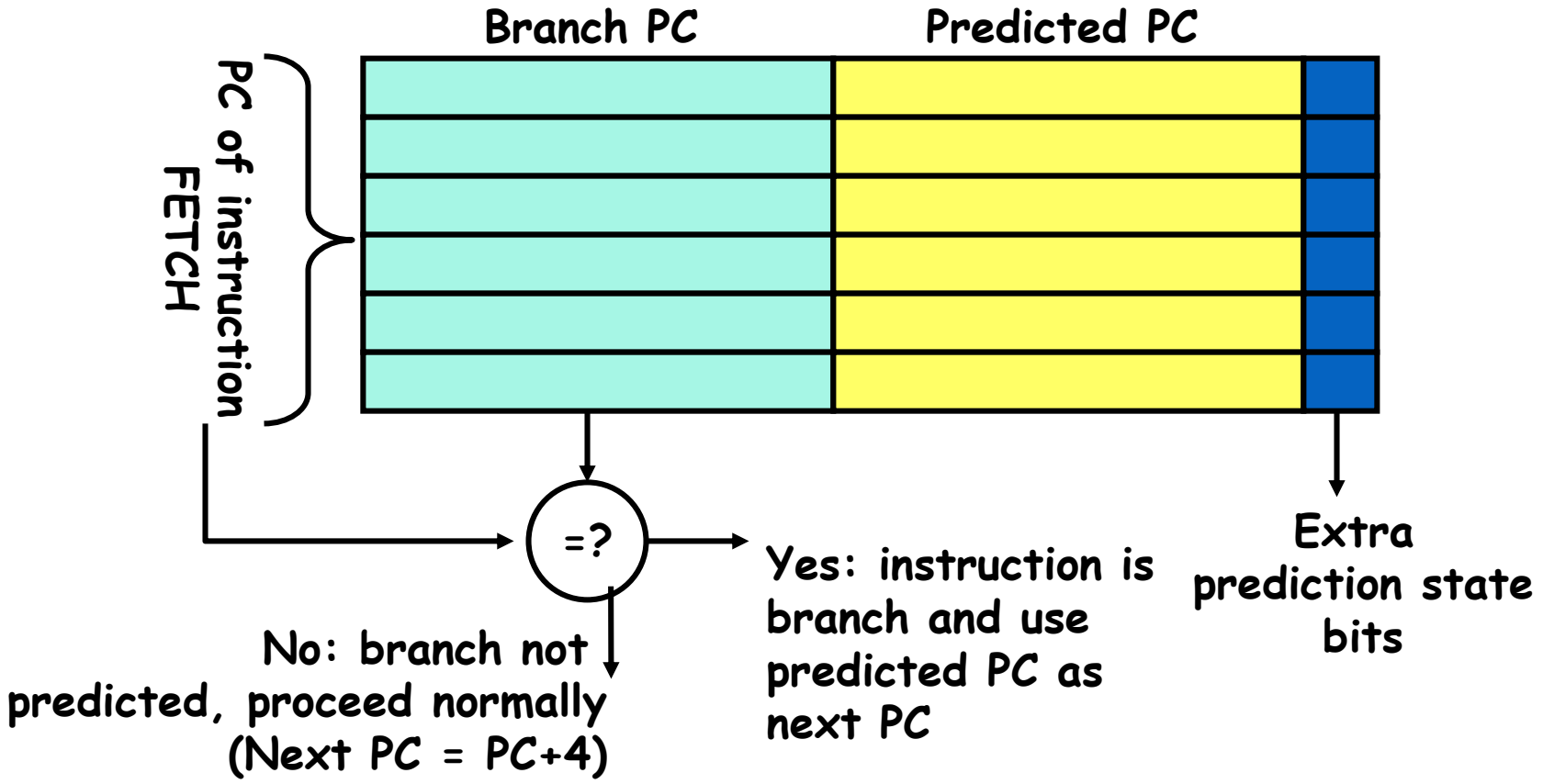
Branch Prediction Performance



Branch predictor performance

Simple dynamic prediction: Branch Target Buffer (BTB)

- 分支指令的地址作为BTB的索引，以得到分支预测地址
  - 必须检测分支指令的地址是否匹配，以免用错误的分支地址
  - 从表中得到预测地址
  - 分支方向确定后，更新预测的PC



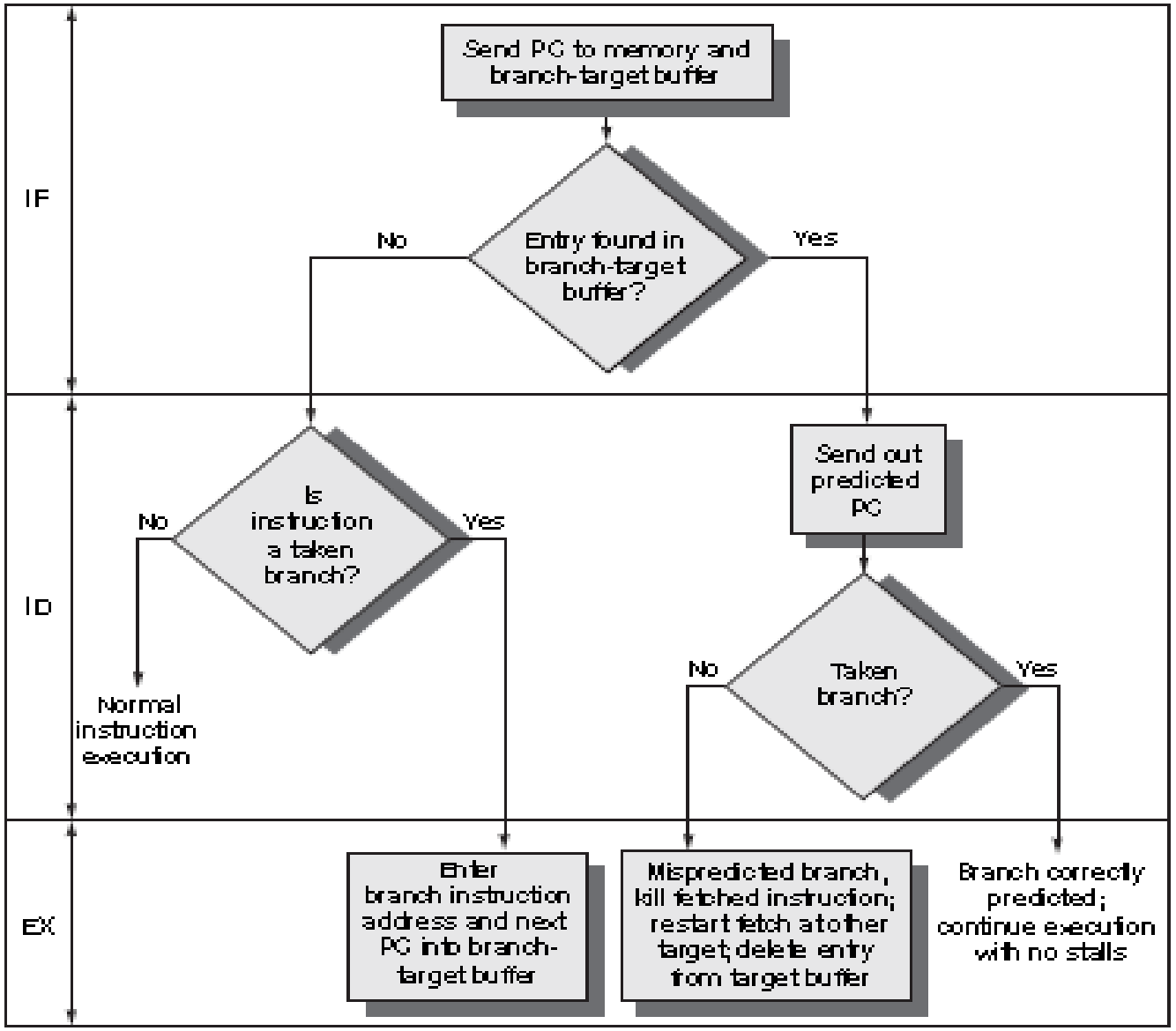


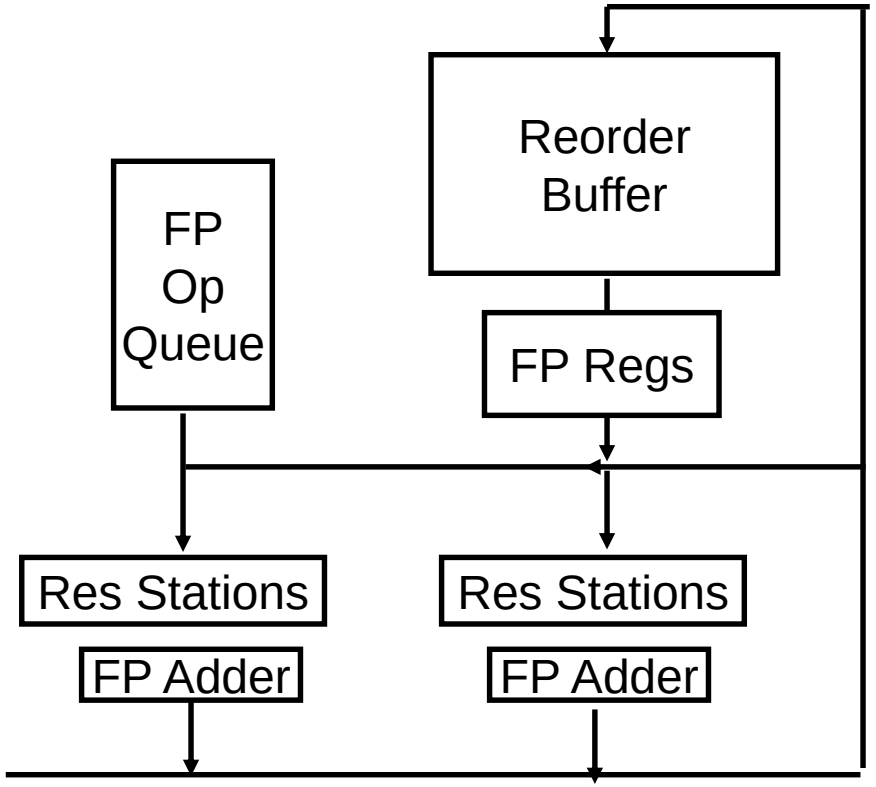
Figure 2.23 The steps involved in handling an instruction with a branch-target buffer.

Instruction in buffer	Prediction	Actual branch	Penalty cycles
yes	taken	taken	0
yes	taken	not taken	2
no		taken	2
no		not taken	0

Figure 2.24 Penalties for all possible combinations of whether the branch is in the buffer and what it actually does, assuming we store only taken branches in the buffer. There is no branch penalty if everything is correctly predicted and the branch is found in the target buffer. If the branch is not correctly predicted, the penalty is equal to 1 clock cycle to update the buffer with the correct information (during which an instruction cannot be fetched) and 1 clock cycle, if needed, to restart fetching the next correct instruction for the branch. If the branch is not found and taken, a 2-cycle penalty is encountered, during which time the buffer is updated.

硬件支持精确中断

- 需要硬件缓存没有提交的指令结果: *reorder buffer （ROB）*
  - 3 个域: 指令类型,目的地址, 值
  - Reorder buffer 可以作为操作数源 => 就像有更多的寄存器（与RS类似）
  - 当指令执行阶段完成后，用ROB的编号代替RS中的值
  - 增加指令提交阶段
  - ROB提供执行完成阶段和提交阶段的操作数
  - 一旦操作数提交，结果就写入寄存器
  - 这样，在预测失败时，容易恢复推断执行的指令，或发生异常时，容易恢复状态



支持推断执行的 Tomasulo 算法的四阶段

1. Issue—get instruction from FP Op Queue
   - 如果RS和ROB有空闲单元就发射指令。如果寄存器或ROB中源操作数可用，就将其发送到RS，目的地址的ROB编号也发送给RS (this stage sometimes called “dispatch”)
2. Execution—operate on operands (EX)
   - 当操作数就绪后，开始执行。如果没有就绪，监测CDB，检查RAW相关
3. Write result—finish execution (WB)
   - 将运算结果通过CDB传送给所有等待结果的FU以及ROB单元，标识RS可用
4. Commit—update register with reorder result
   - 按ROB表中顺序，如果结果已有，就更新寄存器（或存储器），并将该指令从ROB表中删除
   - 预测失败或有中断时，刷新ROB

P191 Figure 3.14 (英文版)
P141 Figure 3-9 (中文版)

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

Tomasulo With Reorder Buffer - Cycle 0

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1		
Load2		
Load3		

Entry	Busy	Instruction	State	Destination	Value
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #									
Busy	no	no	no	no	no	no	no		no

Tomasulo With Reorder Buffer - Cycle 1

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

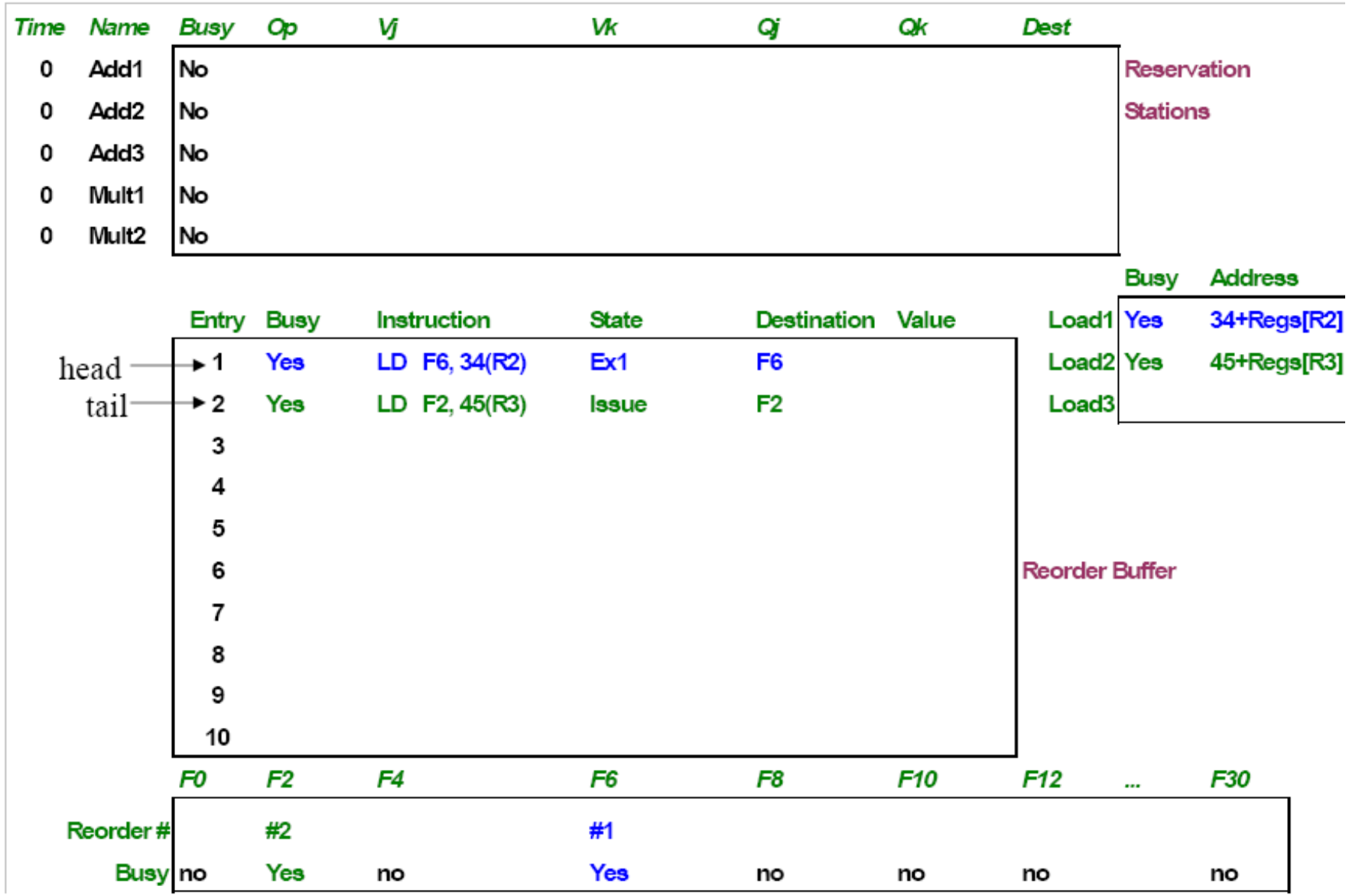
	Busy	Address
Load1	Yes	34+Regs[R2]
Load2		
Load3		

Entry	Busy	Instruction	State	Destination	Value
1	Yes	LD F6, 34(R2)	Issue	F6	
2					
3					
4					
5					
6					
7					
8					
9					
10					

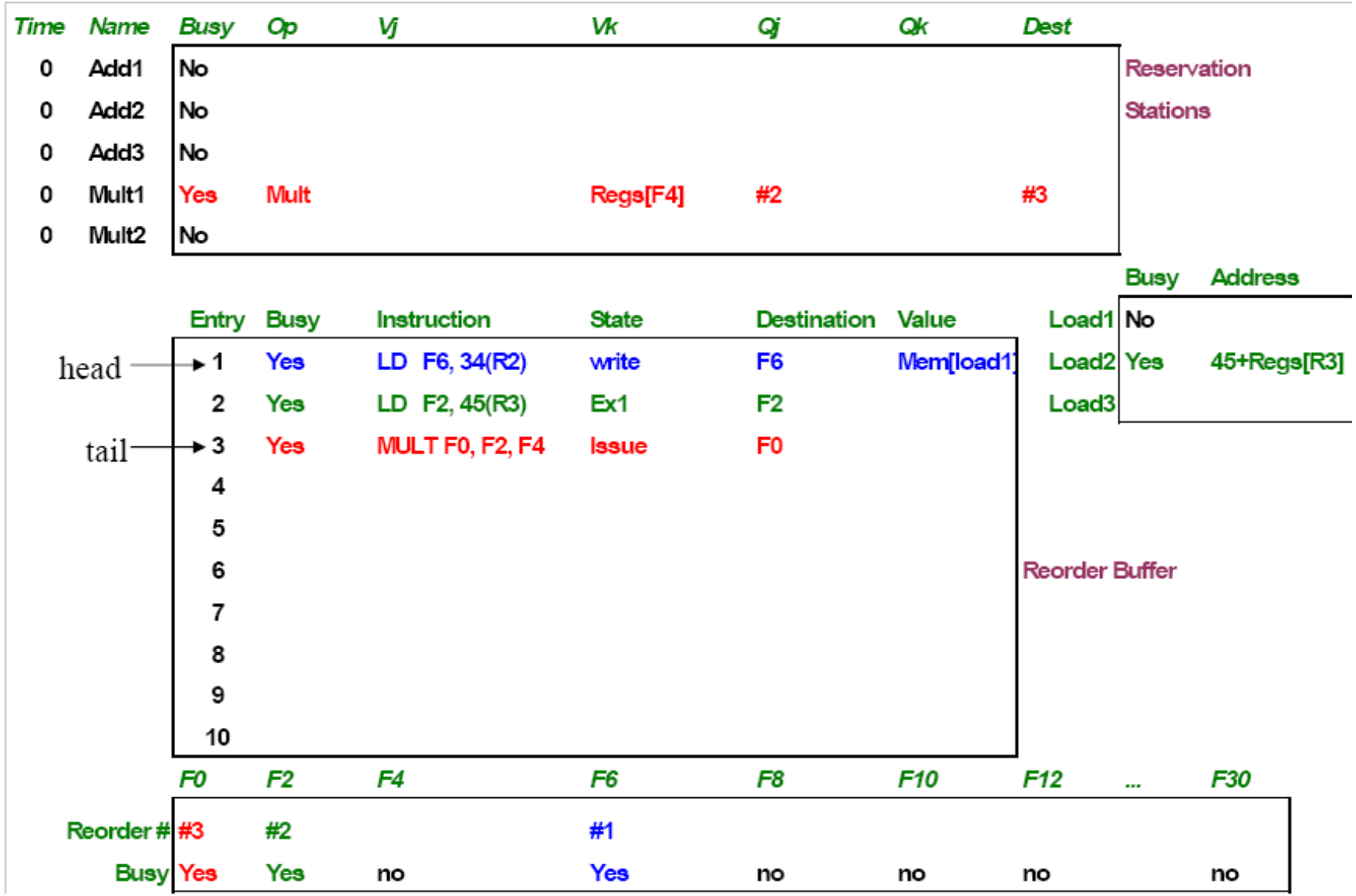
Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#1					
Busy	no	no	no	Yes	no	no	no		no

Tomasulo With Reorder Buffer - Cycle 2



Tomasulo With Reorder Buffer - Cycle 3



Tomasulo With Reorder Buffer - Cycle 4

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
head →	2	Yes	LD F2, 45(R3)	write	F2	Mem[load2]
	3	Yes	MULT F0, F2, F4	EX1	F0	
tail →	4	Yes	SUBD F8, F6, F2	Issue	F8	
	5					
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3	#2			#4				
Busy	Yes	Yes	no	no	Yes	no	no		no

Tomasulo With Reorder Buffer - Cycle 5

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

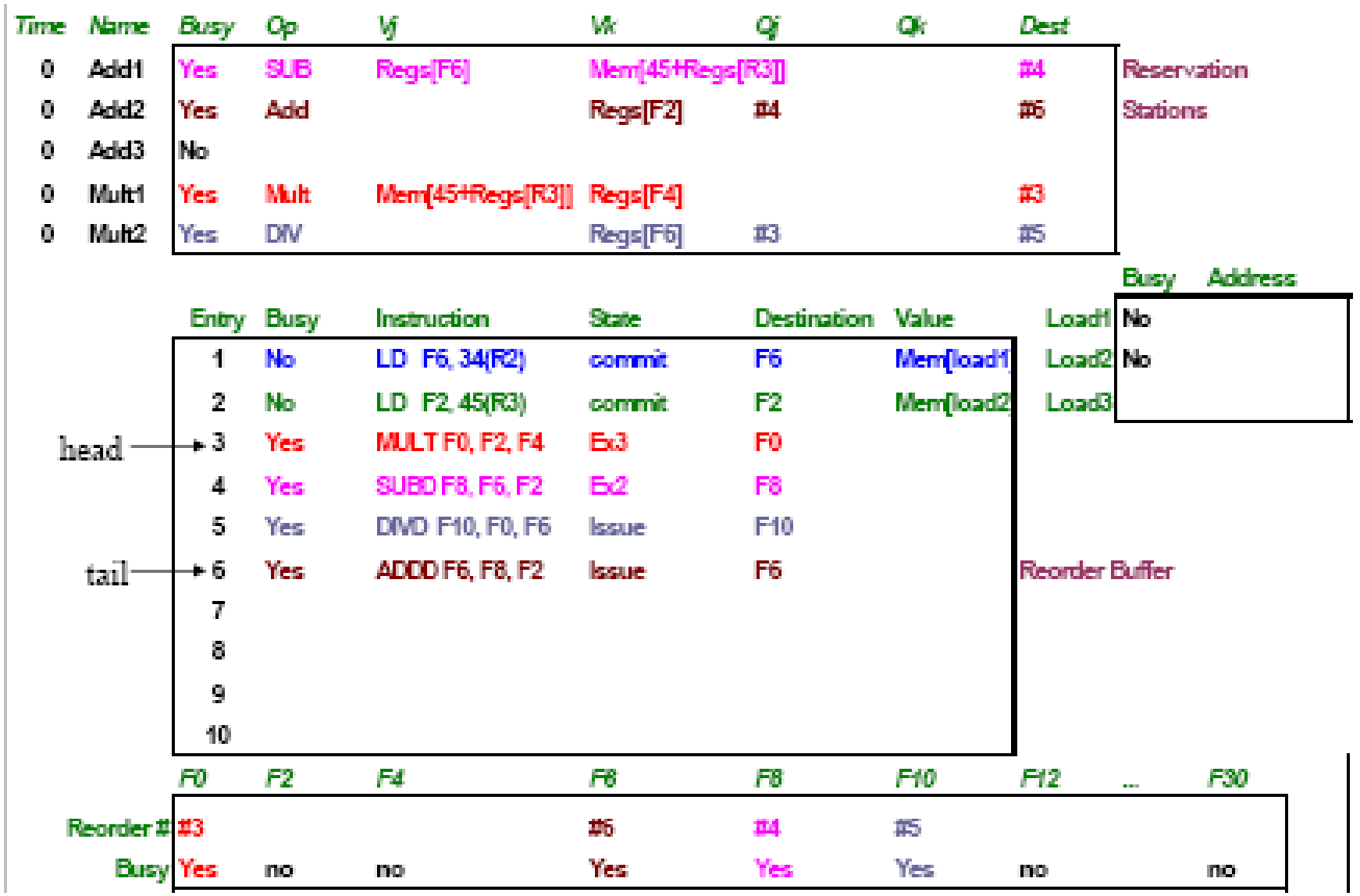
	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex2	F0	
	4	Yes	SUBD F8, F6, F2	Ex1	F8	
tail →	5	Yes	DIVD F10, F0, F6	Issue	F10	
	6					
	7					
	8					
	9					
	10					

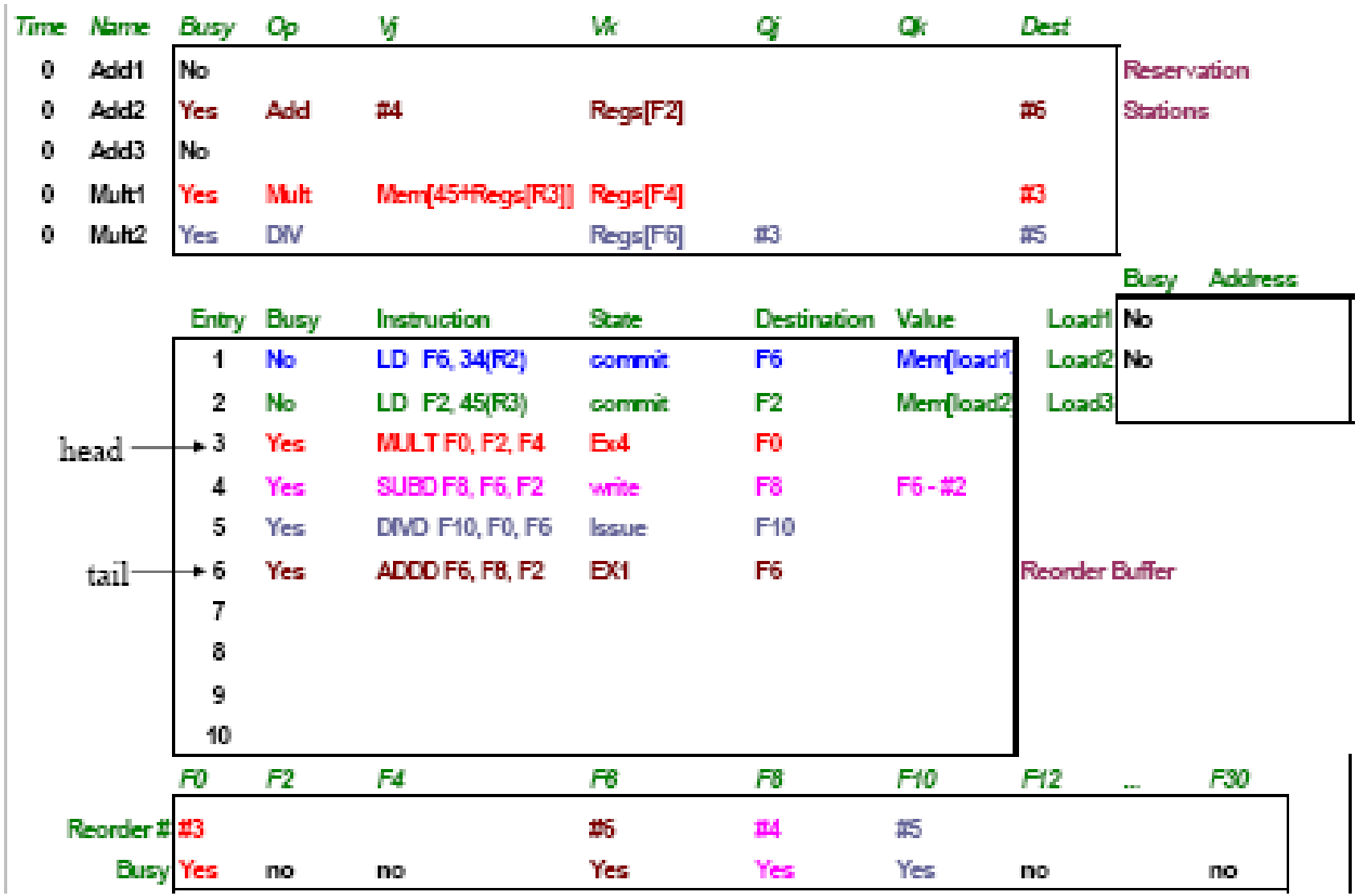
Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3				#4	#5			
Busy	Yes	no	no	no	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 6



Tomasulo With Reorder Buffer - Cycle 7



Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

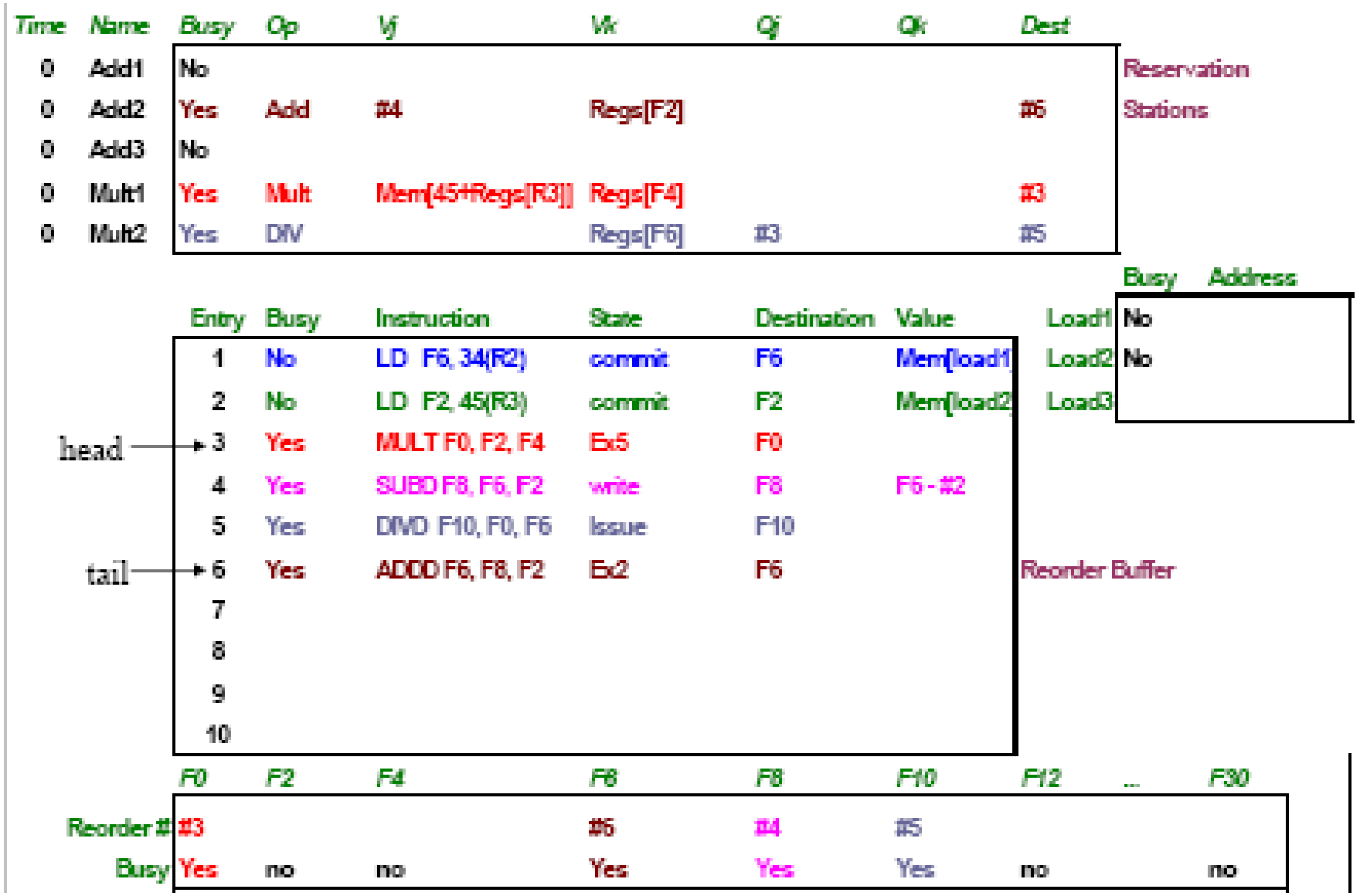
	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex4	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	EX1	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#5	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 8



Tomasulo With Reorder Buffer - Cycle 9

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex6	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 10

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

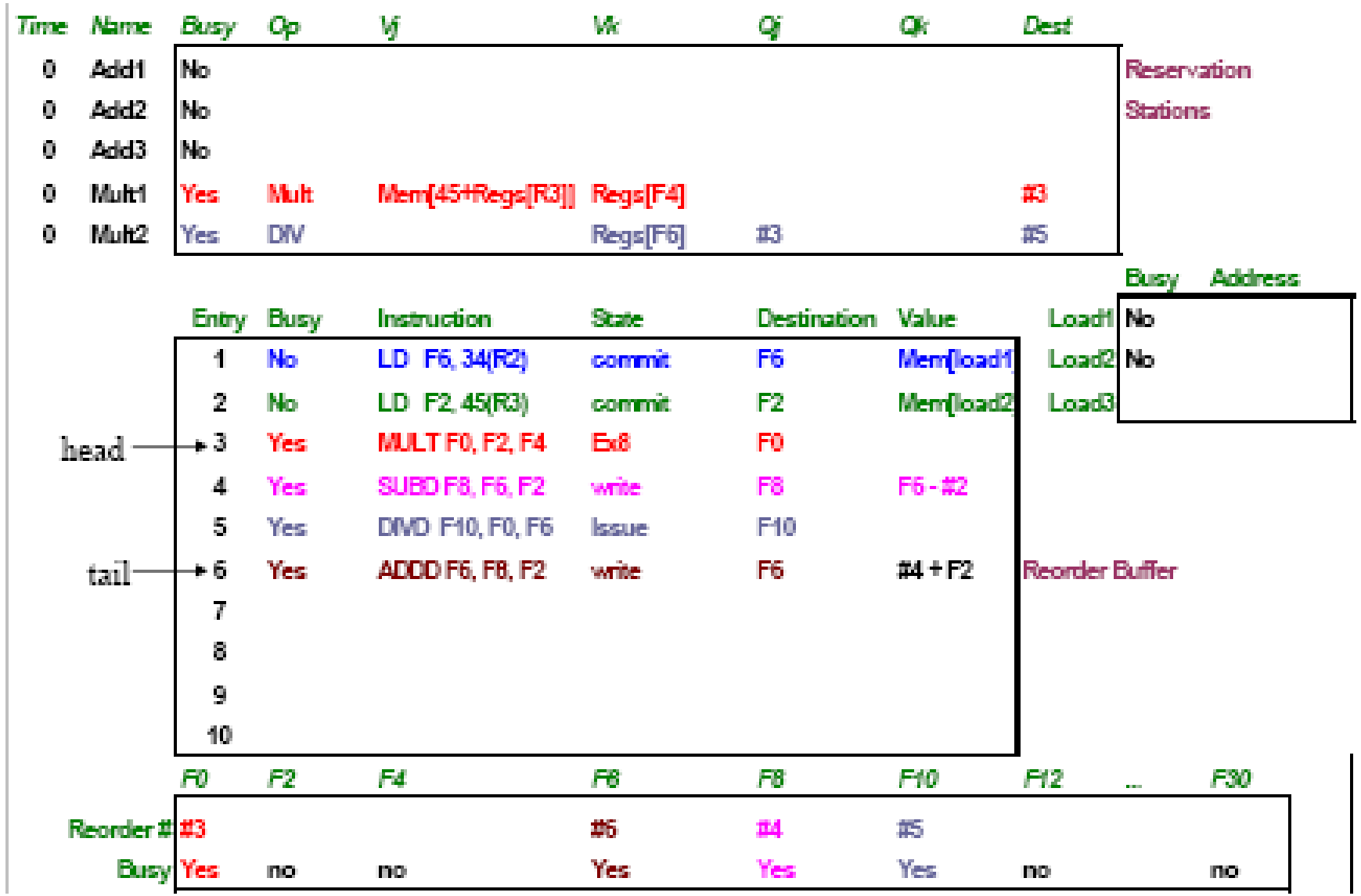
	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex7	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 11



Tomasulo With Reorder Buffer - Cycle 12

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2
head →	3	Yes	MULT F0, F2, F4	Ex9	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#5	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 13

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	write	F0	#2 x Regs[F4]
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex1	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Figure 3.30

P 230

Tomasulo With Reorder Buffer - Cycle 14

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

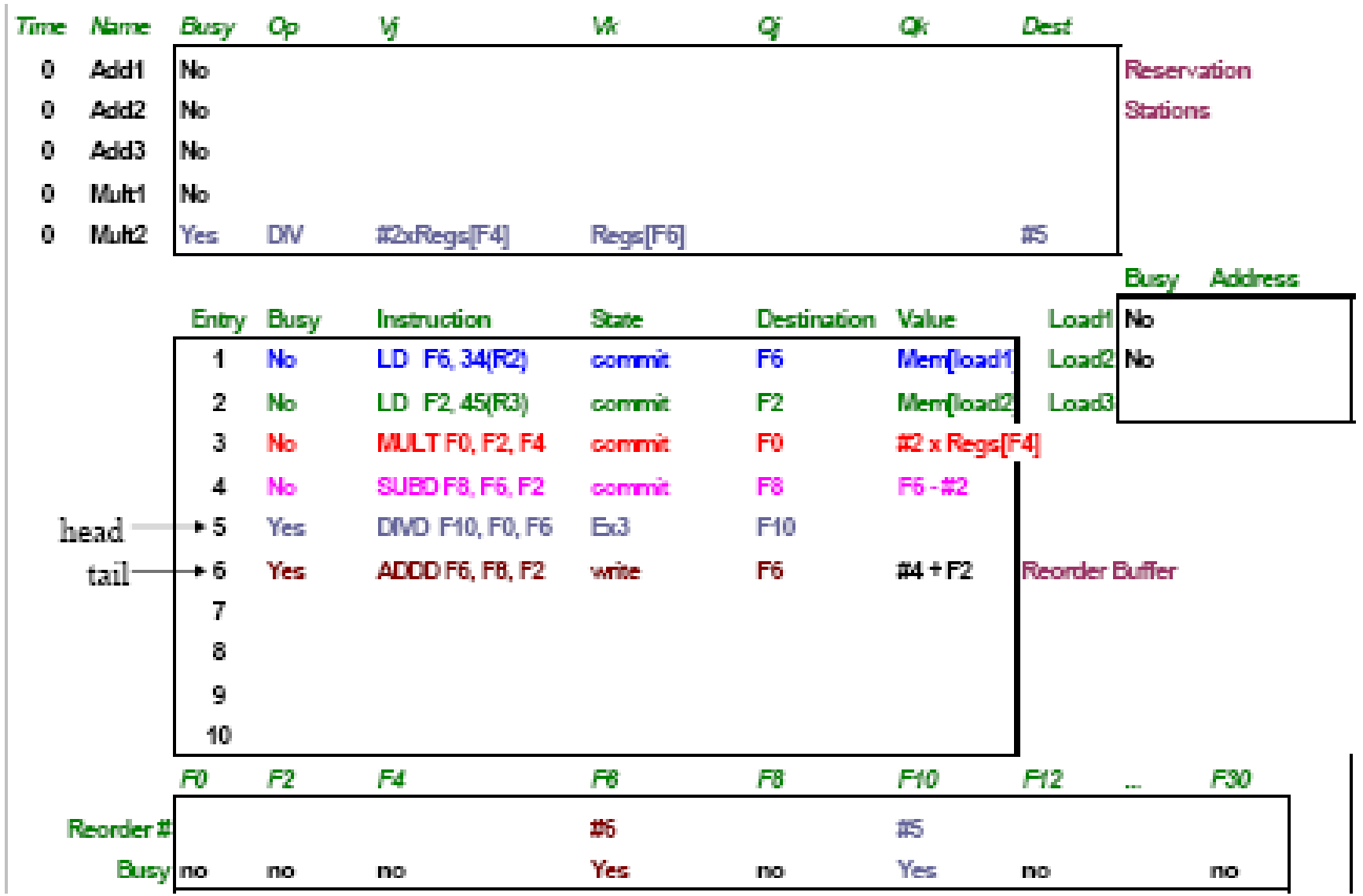
	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
head →	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex2	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#5	#4	#5			
Busy	No	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 15



Tomasulo With Reorder Buffer - Cycle 16

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
	4	No	SUBD F8, F6, F2	commit	F8	F6 - #2
head →	5	Yes	DIVD F10, F0, F6	Ex4	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

Need 36 more EX cycles for DIV to finish…

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6		#5			
Busy	no	no	no	Yes	no	Yes	no		no

Tomasulo With Reorder Buffer: Summary

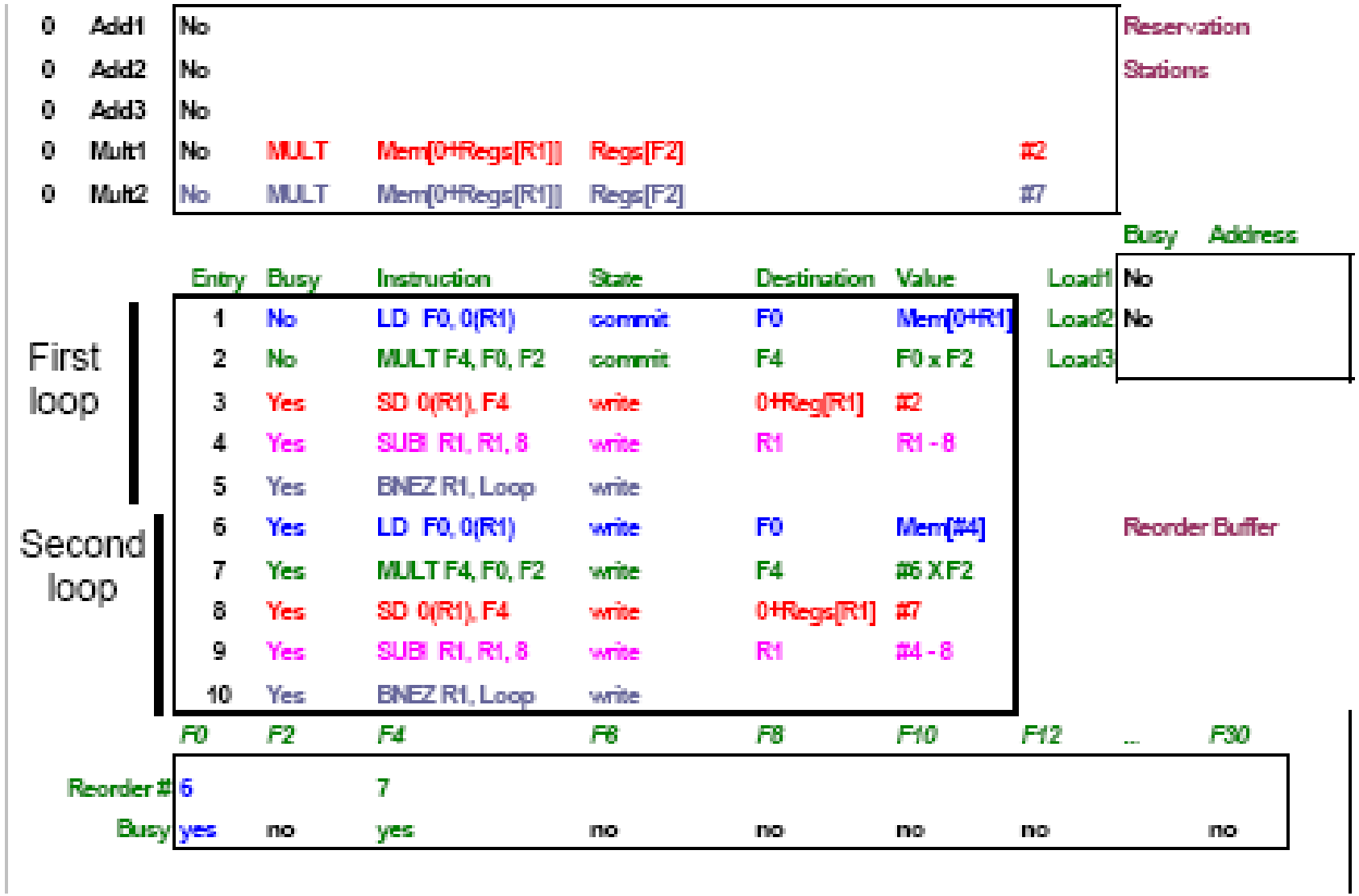
Instruction	Issue	Exec Comp	Writeback	Commit
LD F6, 34(R2)	1	2	3	4
LD F2, 45(R3)	2	3	4	5
MULT F0, F2, F4	3	12	13	14
SUBD F8, F6, F2	4	6	7	15
DIVD F10, F0, F6	5	52	53	54
ADDD F6, F8, F2	6	8	9	55

In-order Issue/Commit, Out-of-Order Execution/Writeback

Precise State with ROB

- ROB maintains precise state and allows speculation
  - Waits until precise condition reaches retire/commit stage
  - (Or until branch is noted mis-predicted)
  - Clear ROB, RS, and register status table (Flush)
  - Service exception/Restart from True Branch target
- Need to do similar things with memory ops
  - Called Memory Ordering Buffer (MOB)
    - Completed stores write to MOB then complete (write to memory) in-order (when they read head of buffer)

Example of Speculative State of Reorder Buffer



Multiply has just reached commit, so other instructions can start committing

消除存储器的二义性 ：
处理对存储器引用的RAW相关

- Question: 给定一个指令序列，store，load 这两个操作是否有关?
  - 即下列代码是否有相关问题?

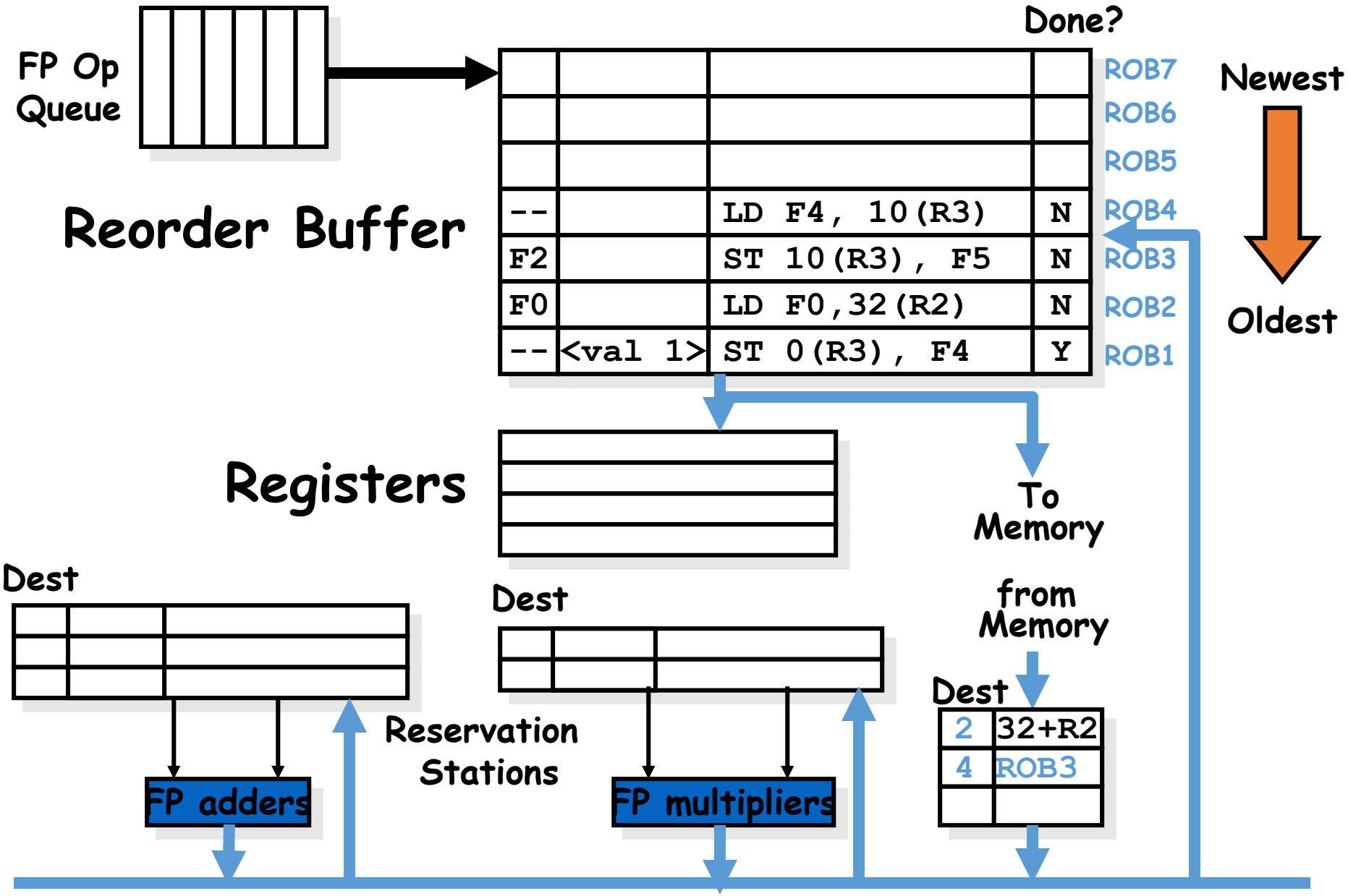
```
Eg:    st    0(R2),R5
       ld    R6,0(R3)
```

- 我们是否可以较早启动ld?
  - Store的地址可能会延迟很长时间才能得到.
  - 我们也许想在同一个周期开始这两个操作的执行.
- 两种方法:
  - No Speculation: 不进行load操作，直到我们确信地址 0(R2) ≠ 0(R3)
  - Speculation: 我们可以假设他们相关还是不相关 (called “dependence speculation”)，如果推测错误通过ROB来修正

Hardware Support for Memory Disambiguation

- 需要缓冲区以程序序保存所有对存储器的写操作
  - 保存地址（地址可用时）和值（值可用时）
  - FIFO ordering: 以程序序确认和删除store
- 当发射一个load操作时，记录当前store队列的头指针.
- 当load的地址可用时，检查store队列:
  - 如果store队列存在正等待该地址的store操作，则stall该load操作
  - 如果load地址与前面的store地址匹配，则有 *memory-induced RAW hazard*:
    - 存储的值可用 ⇒ 返回值
    - 存储的值还没有准备好 ⇒ 返回源指令的ROB编号
  - 否则发出存储器请求
- 推测执行时，由于实际的store操作顺序提交，所以不会有WAR/WAW 相关.

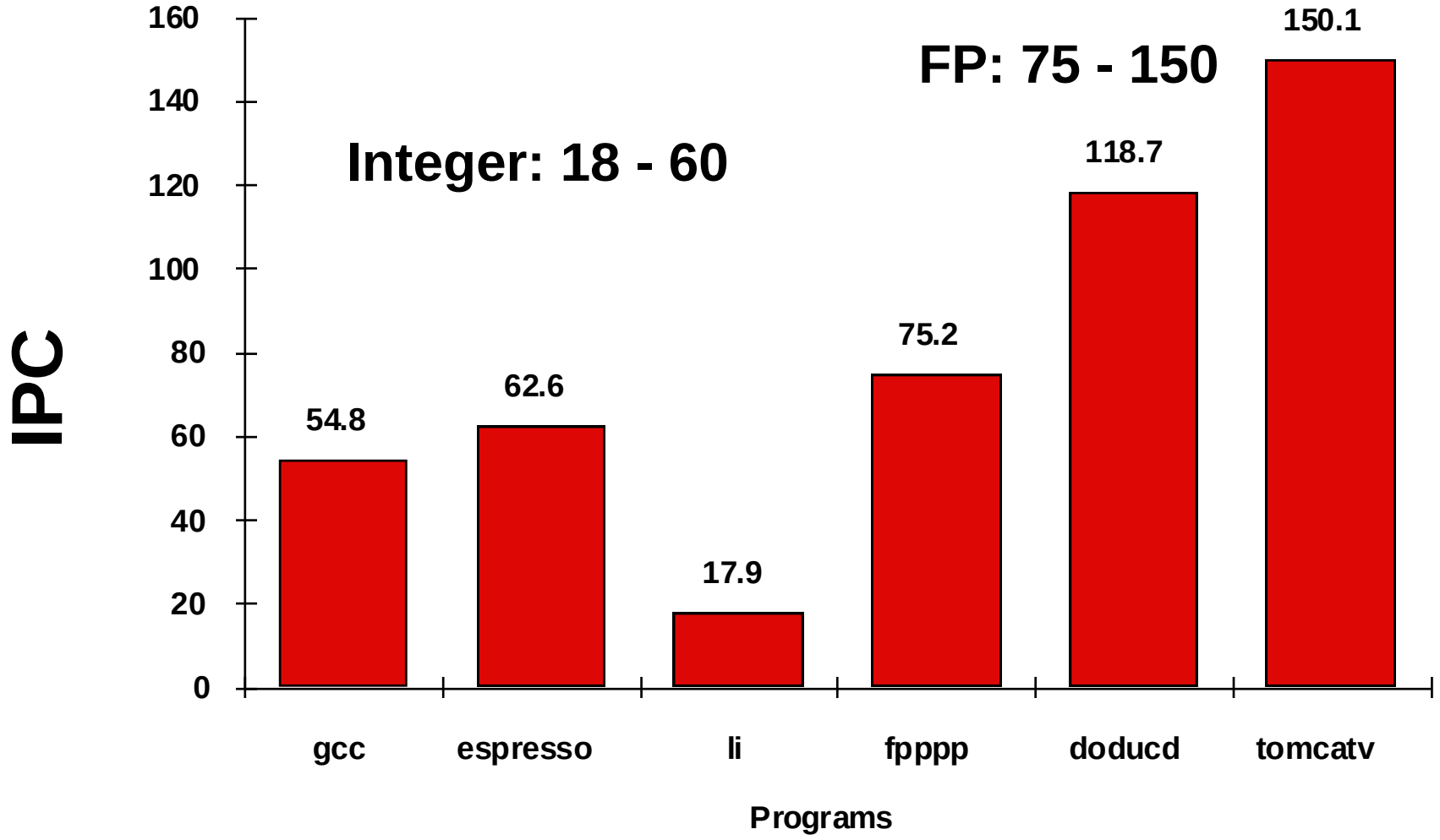
Memory Disambiguation:



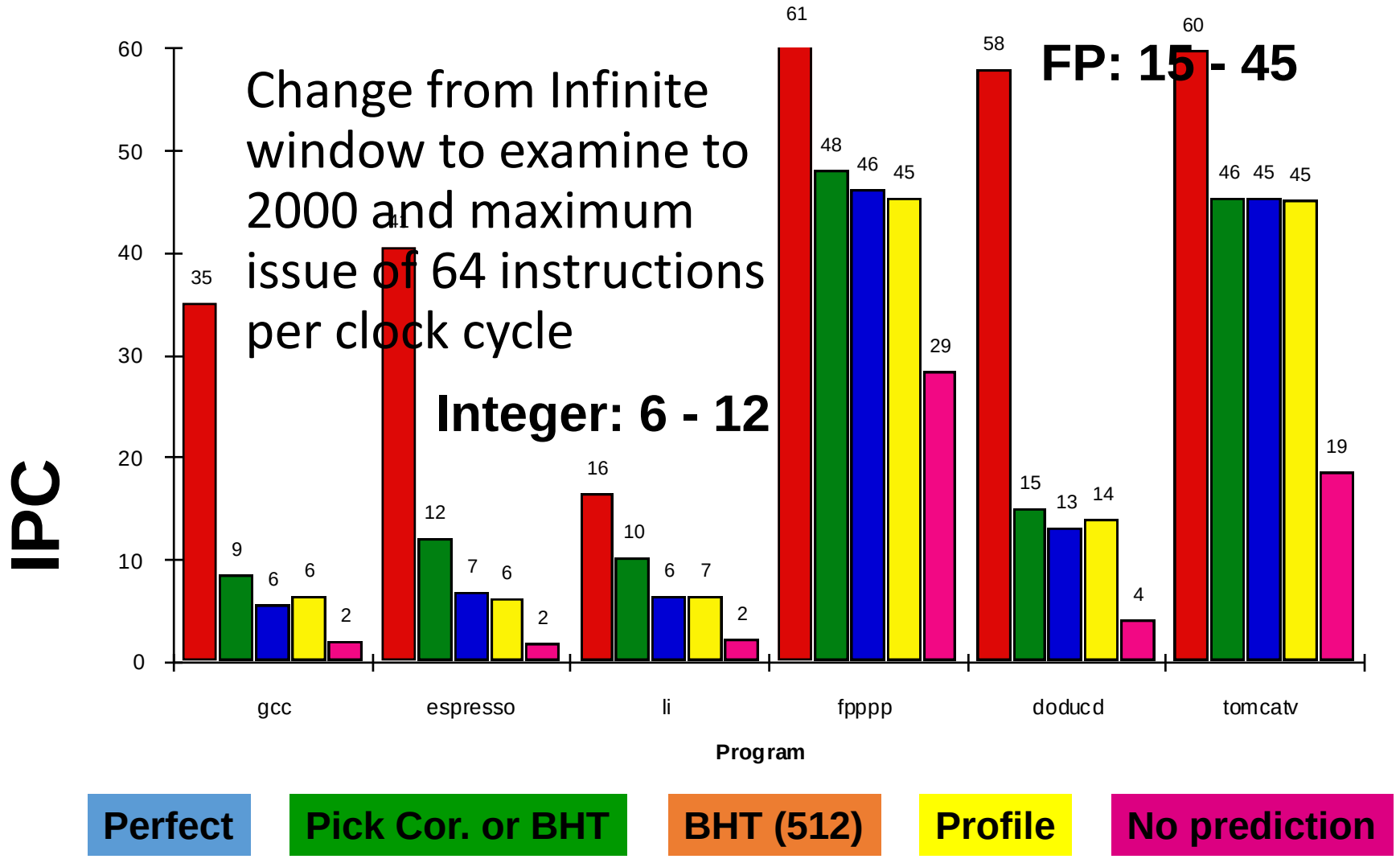
Tomasulo + ROB Summary

- Many implementations are very similar
  - Pentium III, PowerPC, etc
- Some limitations
  - Too many value copy operations
    - Register file => RS => ROB => Register File
  - Too many muxes/busses (CDB)
    - Values are coming from everywhere to everywhere else!
  - Reservation Stations mix values(data) and tags(control)
    - Slows down the max clock frequency

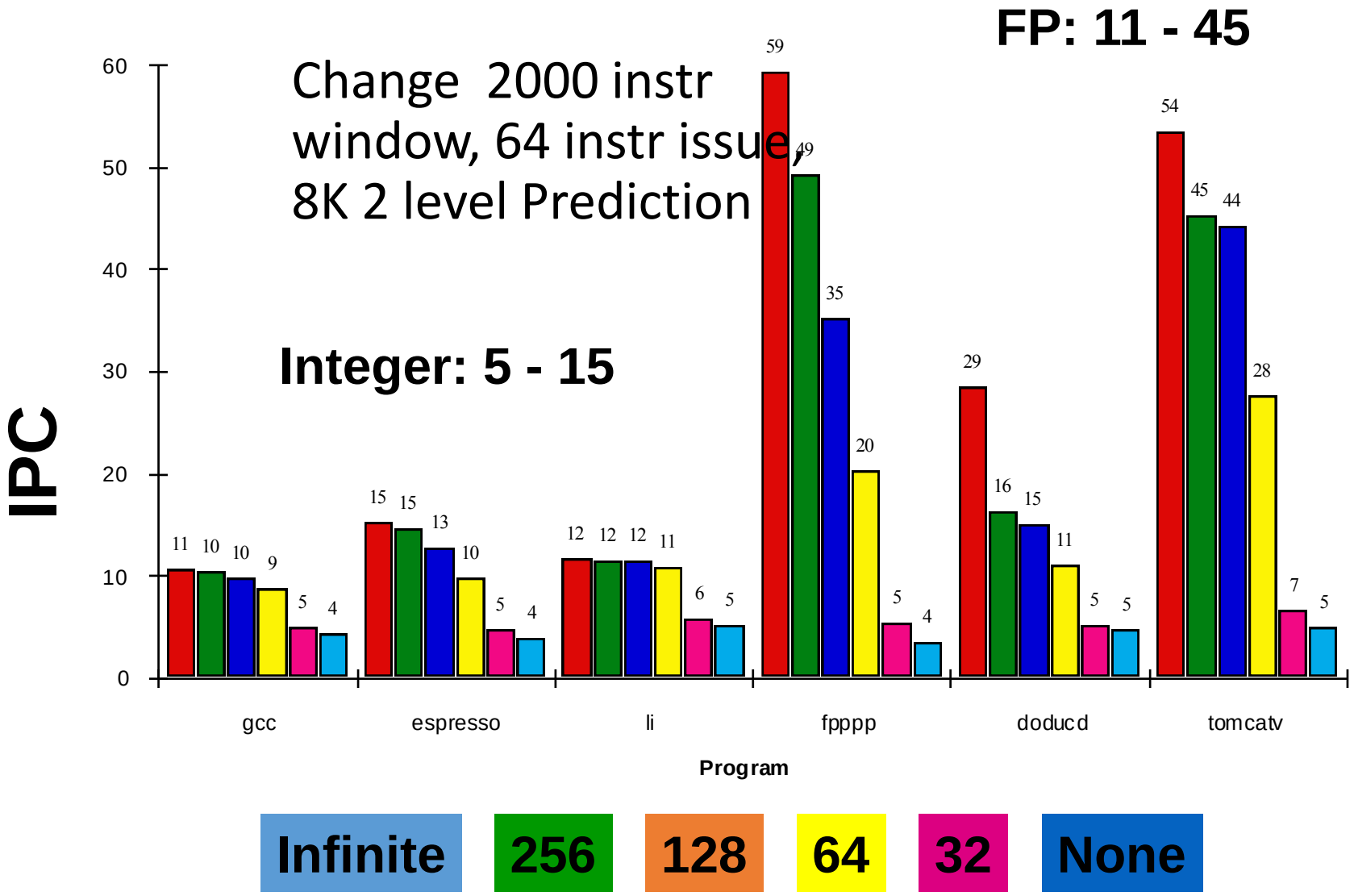
Upper Limit to ILP: Ideal Machine



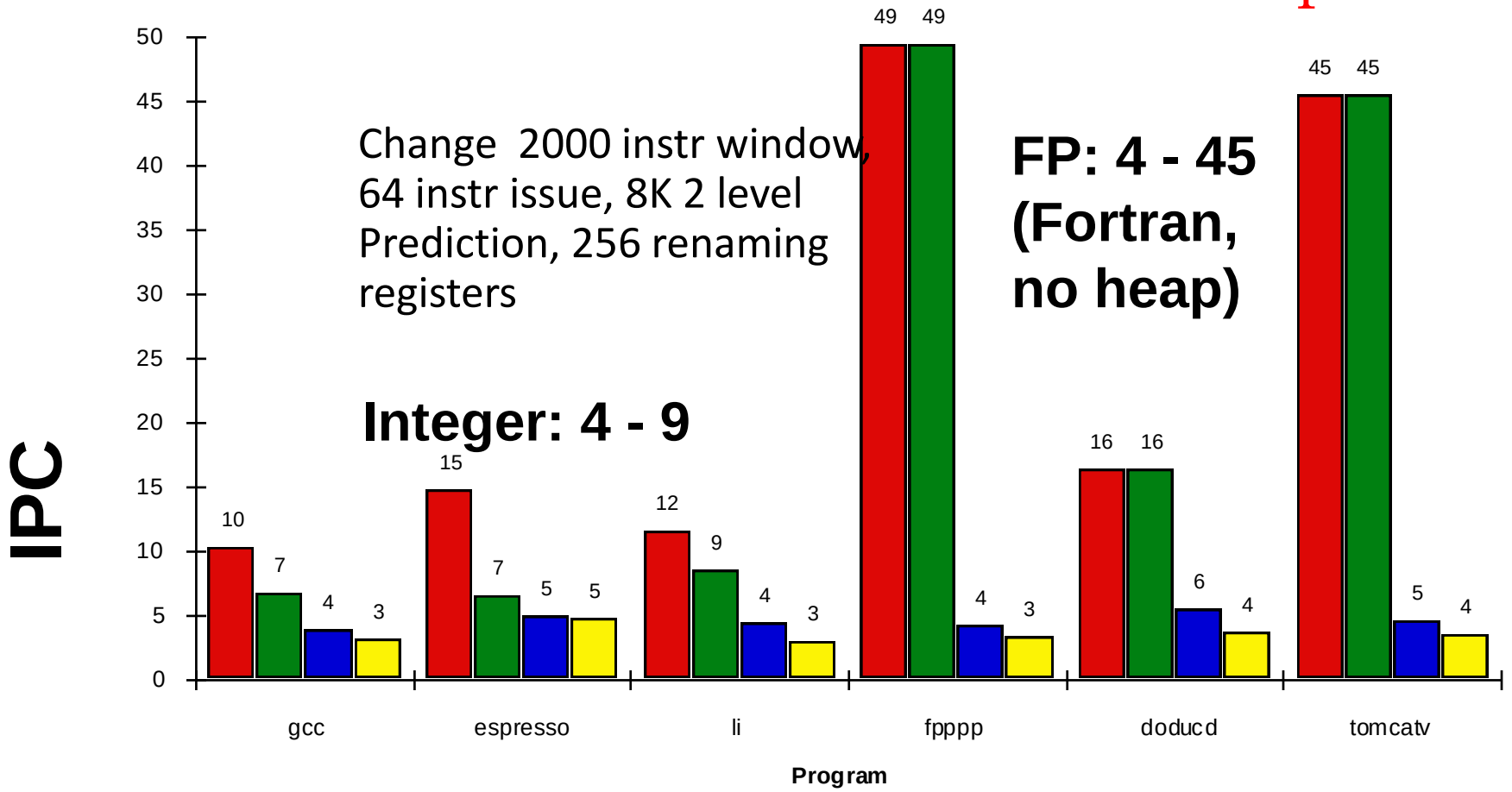
More Realistic HW: Branch Impact



More Realistic HW: Register Impact (rename regs)

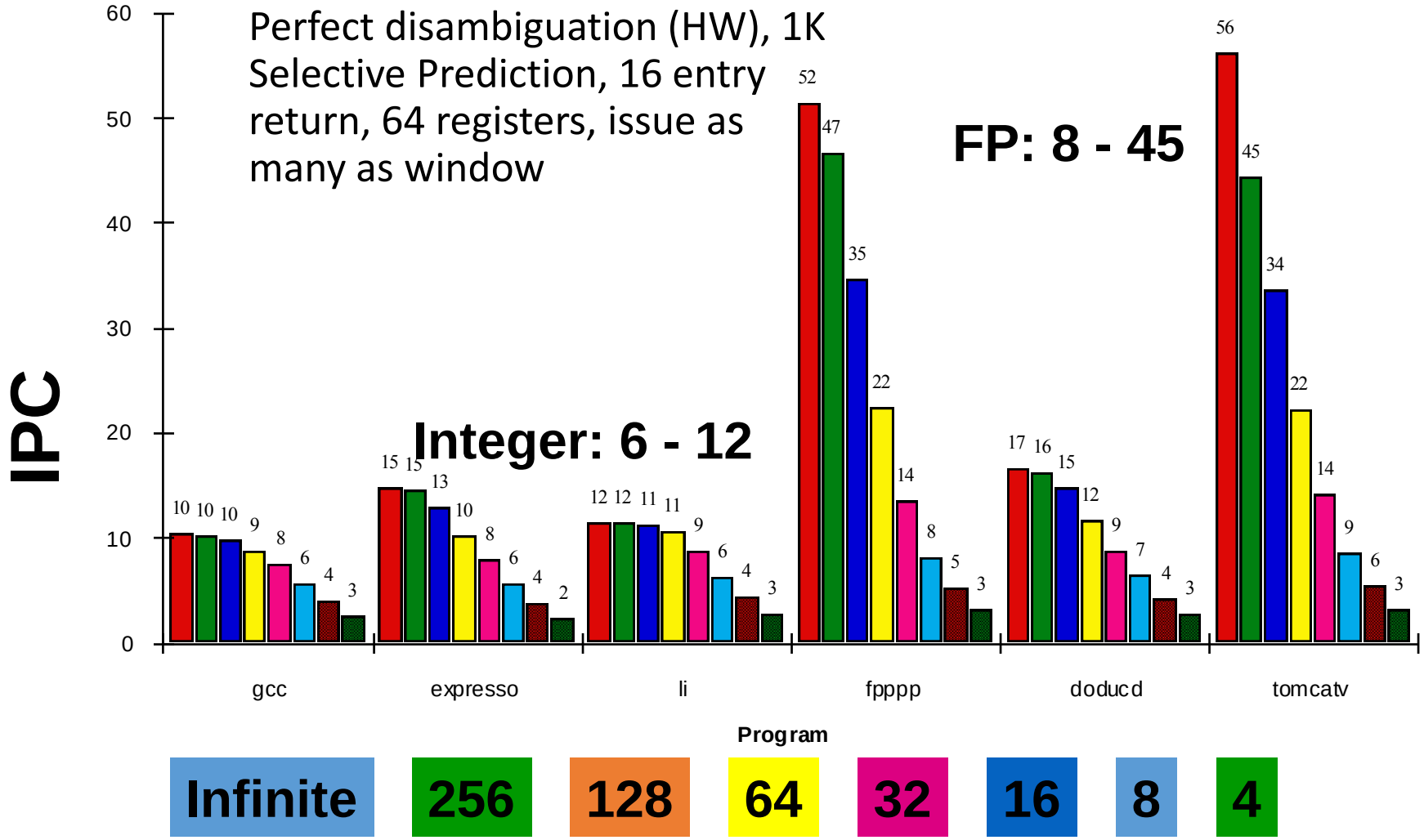


More Realistic HW: Alias Impact



Perfect	Global/Stack perf; heap conflicts	Inspec. Assem.	None

Realistic HW for '9X: Window Impact



小结#1/2

- Reservations stations: 寄存器重命名，缓冲源操作数
  - 避免寄存器成为瓶颈
  - 避免了Scoreboard中无法解决的 WAR, WAW hazards
  - 允许硬件做循环展开
  - 不限于基本块(IU先行，解决控制相关)
- 贡献
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 后 Pentium II; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264使用这种技术

小结 #2/2

- 动态调度方案可以用硬件动态完成循环展开
  - 通过重命名机制来消除WAR和 WAW 相关
- Reorder Buffer:
  - 提供了撤销指令运行的机制
  - 指令以发射序存放在ROB中
  - 指令顺序提交
- 分支预测对提高性能是非常重要的
  - 推断执行是利用了ROB撤销指令执行的机制
- Superscalar 和VLIW: CPI < 1 (IPC > 1)

如何使CPI < 1 (1/2)

- 前面所述的各种技术主要通过减少数据相关和控制相关，使得CPI = 1 (CPI接近1）
- 是否能够使CPI < 1?
- 两种基本方法
- Superscalar:
  - 每个时钟周期所发射的指令数不定（1 —8条）
  - 由编译器或硬件完成调度
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000
  - 该方法对目前通用计算是最成功的方法
- 新的概念 Instructions Per Clock (IPC) vs. CPI

如何使 CPI < 1？ (2/2)

- (Very) Long Instruction Words (V)LIW:
  - 每个时钟周期流出的指令数（操作）固定 (4-16)
  - 由编译器调度，实际上由多个单操作指令构成一个超长指令
  - 目前比较成功的应用于DSP，多媒体应用
  - 1999/2000 HP和Intel达成协议共同研究VLIW
  - Intel Architecture-64 (Merced/A-64) 64-bit address
  - Style: “Explicitly Parallel Instruction Computer (EPIC)”

用于多发射处理器的五种主要方法

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	hardware	static	in-order execution	Sun UltraSPARC II/ III
Superscalar (dynamic)	dynamic	hardware	dynamic	some out-of-order execution	HP PA 8500, IBM RS64 III
Superscalar (speculative)	dynamic	hardware	dynamic with speculation	out-of-order execution with speculation	Pentium III/4, MIPS R10K, Alpha 21264
VLIW/LIW	static	software	static	no hazards between issue packets	Trimedia, i860
EPIC	mostly static	mostly software	mostly static	explicit dependences marked by compiler	Itanium

FIGURE 3.23 **There are five primary approaches in use for multiple-issue processors, and this table shows the primary characteristics that distinguish them.** This chapter has focused on the hardware-intensive techniques, which are all some form of superscalar. The next chapter focuses on compiler-based approaches, which are either VLIW or EPIC. Figure 3.61 on page 341 near the end of this chapter provides more details on a variety of recent superscalar processors.

Superscalar DLX

- Superscalar DLX: 每个时钟周期发射2条指令，1 条FP指令和一条其他指令
  - 每个时钟周期取64位; 左边为Int , 右边为FP
  - 只有第一条指令发射了，才能发射第二条
  - 需要更多的寄存器端口，因为如果两条指令中第一条指令是对FP的load操作（通过整数部件完成），另一条指令为浮点操作指令，则都会有对浮点寄存器文件的操作

Type	*Pipe Stages*					
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction		IF	ID	EX	MEM	WB
FP instruction		IF	ID	EX	MEM	WB
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 原来1 cycle load 延时在Superscalar中扩展为3条指令

Review: 具有最小stalls数的循环展开优化

```
1 Loop: 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
```

LD to ADDD: 1 Cycle
ADDD to SD: 2 Cycles

14 clock cycles, or 3.5 per iteration

采用Superscalar技术的循环展开

	Integer instruction	*FP instruction*	*Clock cycle*
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD ＋8(R1),F20		12

- 循环展开5次以消除延时 (+1 due to SS)
- 12 clocks, or 2.4 clocks per iteration (1.5X)

多发射的问题

- 如果Integer和FP操作很容易区分组合，那么对这类程序在下列条件满足的情况下理想CPI= 0.5 :
  - 程序中50% 为FP 操作
  - 没有任何相关
- 如果在同一时刻发射的指令越多，译码和发射就越困难
  - 即使是同一时刻发射2条 =>需检查2个操作码，6个寄存器描述符 ，检查是发射1条还是2条指令。

- VLIW
  - 指令字较长可以容纳较多的操作
  - 根据定义,VLIW中的所有操作是由编译时刻组合的，并且是相互无关的，也就是说：可以并行执行
  - 例如 2 个整数操作，2个浮点操作，2个存储器引用，1个分支指令
    - 每一个操作用16 到 24 位 表示 => 共7*16 ＝ 112 bits 到 7*24 ＝ 168 bits wide
  - 需要用编译技术调度来解决分支问题

基于VLIW的循环展开

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16			SUBI R1,R1,#48	7
SD 16(R1),F20	SD 8(R1),F24				8
SD -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays

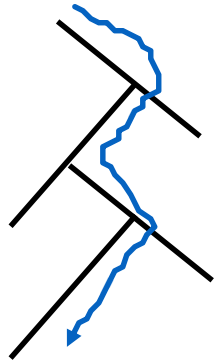
7 results in 9 clocks, or 1.3 clocks per iteration (1.8X)

Average: 2.5 ops per clock, 50% efficiency

注: 在VLIW中，一条超长指令有更多的读写寄存器操作(15 vs. 6 in SS)

Trace Scheduling

- 消除分支的一种策略
- 两步：
  - *Trace Selection*
    - 搜索可能最长的直线型代码（由一组基本块构成）（通过静态预测或profile技术）(*trace*)
  - *Trace Compaction*
    - 将trace中的指令拼装为若干条VLIW 指令
    - 需要一些保存环境的代码，以防预测错误
- 由编译器撤销预测错误造成的后果（恢复寄存器的原值)



HW推断执行(Tomasulo) vs.
SW (VLIW) 推断执行

- HW 确定地址冲突
- HW 分支预测较好，预测准确率较高
- HW 可支持精确中断模型
- HW 不必执行保存环境和恢复环境的指令
- SW 推断执行比HW设计简单的多

Superscalar vs. VLIW

- 代码量较小
- 二进制兼容性好

- 译码、发射指令的硬件设计简单
- 更多的寄存器，一般使用多个寄存器文件而不是多端口寄存器文件

Superscalar 的动态调度（1/2）

- 静态调度的缺陷：
  - 有相关就停止发射
  - 基于原来Superscalar的代码生成器所生成的代码可能在新的Superscalar上运行效率较差，代码与superscalar的结构有关

Superscalar 的动态调度（2/2）

- 用Tomasulo如何发射两条指令并保持指令序
  - 假设有1 浮点操作，1个整数操作
  - Tomasulo控制器一个控制整型操作的发射，一个控制浮点型操作的发射
- 如果每个周期发射两条不同的指令，比较容易保持指令序（整型类操作序，浮点类操作序）
- 现在只有FP的Loads操作可能会引起整型操作发射和浮点操作发射的相关
- 存储器引用问题：
  - 将load的保留站组织成队列方式，操作数必须按指令序读取
  - Load操作时检测Store队列中Store的地址以防止RAW冲突
  - Store操作时检测Load队列的地址，以防止WAR相关
  - Store操作按指令序进行，防止WAW相关

Example

Consider the execution of the following loop, which increments each element of an integer array, on a two-issue processor, once without speculation and once with speculation:

```
Loop:  LD R2,0(R1)          ; R2=array element
       DADDIU R2,R2,#1 ; increment R2
       SD R2,0(R1)          ;store result
       DADDIU R1,R1,#8 ;increment pointer
       BNE R2,R3,LOOP ;branch if not last element
```

Assume that there are separate integer functional units for effective address calculation, for ALU operations, and for branch condition evaluation. Create a table for the first three iterations of this loop for both processors. Assume that up to two instructions of any type can commit per clock.

Iteration number	Instructions		Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD	R2,0(R1)	1	2	3	4	First issue
1	DADDIU	R2,R2,#1	1	5		6	Wait for LW
1	SD	R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	Execute directly
1	BNE	R2,R3,LOOP	3	7			Wait for DADDIU
2	LD	R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU	R2,R2,#1	4	11		12	Wait for LW
2	SD	R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU	R1,R1,#8	5	8		9	Wait for BNE
2	BNE	R2,R3,LOOP	6	13			Wait for DADDIU
3	LD	R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU	R2,R2,#1	7	17		18	Wait for LW
3	SD	R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU	R1,R1,#8	8	14		15	Wait for BNE
3	BNE	R2,R3,LOOP	9	19			Wait for DADDIU

Figure 2.20 The time of issue, execution, and writing result for a dual-issue version of our pipeline *without* speculation. Note that the LD following the BNE cannot start execution earlier because it must wait until the branch outcome is determined. This type of program, with data-dependent branches that cannot be resolved earlier, shows the strength of speculation. Separate functional units for address calculation, ALU operations, and branch-condition evaluation allow multiple instructions to execute in the same cycle. Figure 2.21 shows this example with speculation,

Iteration number	Instructions		Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD	R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LW
1	SD	R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	8	Commit in order
1	BNE	R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU	R2,R2,#1	4	8		9	10	Wait for LW
2	SD	R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU	R1,R1,#8	5	6		7	11	Commit in order
2	BNE	R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD	R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU	R2,R2,#1	7	11		12	13	Wait for LW
3	SD	R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU	R1,R1,#8	8	9		10	14	Executes earlier
3	BNE	R2,R3,LOOP	9	13			14	Wait for DADDIU

Figure 2.21 The time of issue, execution, and writing result for a dual-issue version of our pipeline *with* speculation. Note that the LD following the BNE can start execution early because it is speculative.

多发射处理器受到的限制（1/2）

- 程序内在的ILP的限制
  - 如果每5条指令中有1条相关指令：如何保持5-路VLIW 并行?
  - 部件的操作延时：许多操作需要调度，使部件延时加大
- 多指令流出的处理器需要大量的硬件资源
  - 需要多个功能部件来使得多个操作并行(Easy)
  - 需要更大的指令访问带宽(Easy)
  - 需要增加寄存器文件的端口数（以及通信带宽） (Hard)
  - 增加存储器的端口数（带宽） (Harder)

多发射处理器受到的限制（2/2）

- 一些由Superscalar或VLIW的实现带来的特殊问题
  - Superscalar的译码、发射问题: 到底能发射多少条指令?
  - VLIW 代码量问题: 循环展开 + VLIW中无用的区域
  - VLIW 互锁 => 1 个相关导致所有指令停顿
  - VLIW 的二进制兼容问题

ILP受到的限制

- 大量研究结果的相互矛盾
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - HW方式好
  - 软件（Compiler）方式好
- 通过增加HW成本使用现有各种机制到底能提高多少ILP?
- 我们是否要设计新的HW/SW机制来提高性能?

Summary

- 推断执行:
  - 在控制相关还没有实际解决的情况下，就开始执行
  - 乱序执行，顺序确认(reorder buffer)
- Superscalar and VLIW: CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - 同一时刻发射更多的指令 => 导致更大的冲突开销
- 基于硬件的推断执行
  - 动态分支预测
  - 推断执行
  - 动态调度

04-28-review #1/2

指令集动态调度方案

- 记分牌、Tomasulo
- 顺序发射、乱序执行、乱序发射

Tomasulo

- Reservations stations: 寄存器重命名，缓冲源操作数
  - 避免寄存器成为瓶颈,避免了Scoreboard中无法解决的 WAR, WAW hazards
- Reorder Buffer：+顺序提交
  - 允许硬件做循环展开,不限于基本块(IU先行，解决控制相关)
- 贡献：Dynamic scheduling、register renaming、Load/store disambiguation

04-28-review #2/2

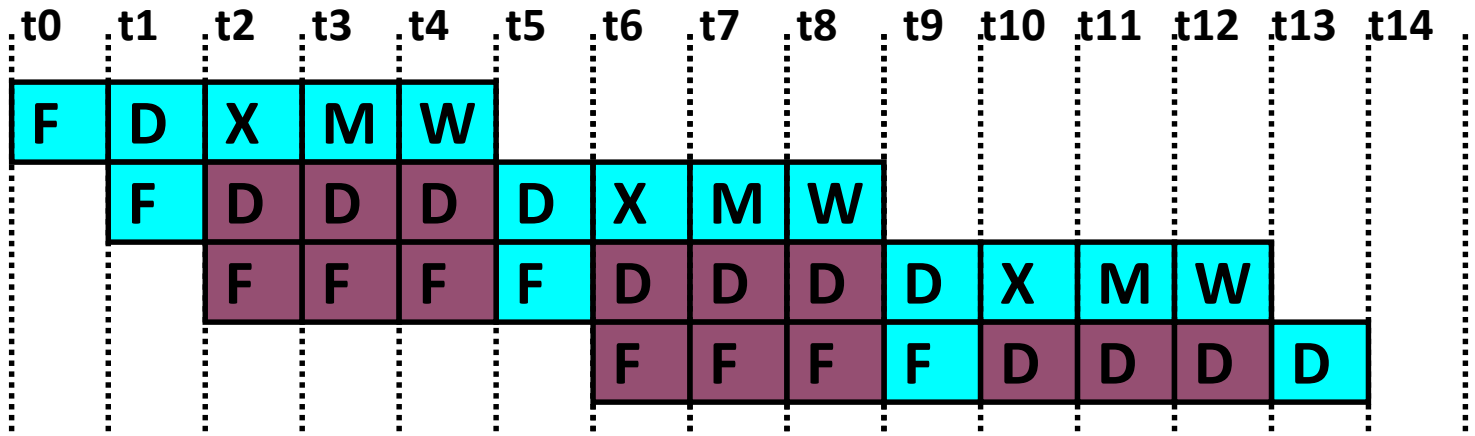
- 分支预测对提高性能是非常重要的
- 推断执行:
  - 推断执行是利用了ROB撤销指令执行的机制
  - 在控制相关还没有实际解决的情况下，就开始执行
  - 乱序执行，顺序确认(reorder buffer)
- 基于硬件的推断执行
  - 动态分支预测
  - 推断执行
  - 动态调度
- Superscalar 和VLIW: CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - 同一时刻发射更多的指令 => 导致更大的冲突开销

Multithreading

- 背景：从单线程程序挖掘指令集并行越来越困难
- 许多工作任务可以使用线程级并行来完成
  - 线程级并行来源于多道程序设计
  - 线程级并行的基础是多线程应用，即一个任务可以用多个线程并行来加速
- 多线程应用可以用线程级并行来提高单个处理器的利用率
  - 针对单个处理器：多个线程以重叠方式共享单个处理器的功能单元

Pipeline Hazards

LW r1, 0(r2)
LW r5, 12(r1)
ADDI r5, r5, #12
SW 12(r1), r5



- Each instruction may depend on the next

如何处理相关?

- –使用interlock机制(slow)
- –或定向路径 (needs hardware, doesn't help all hazards)

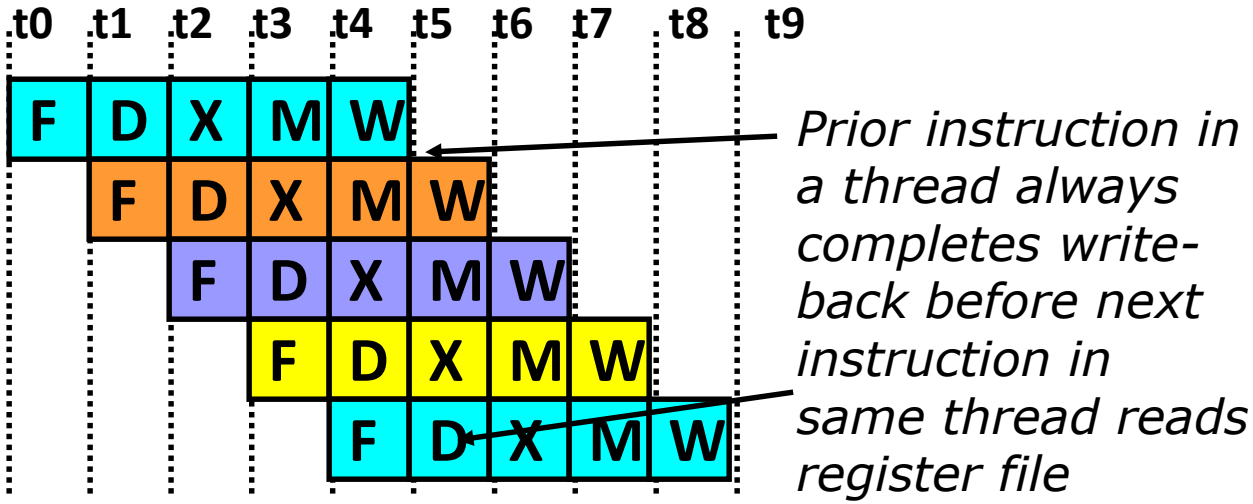
Multithreading

如何保证流水线中指令间无数据依赖关系？

一种办法：在相同的流水线中交叉执行来自不同线程的指令

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

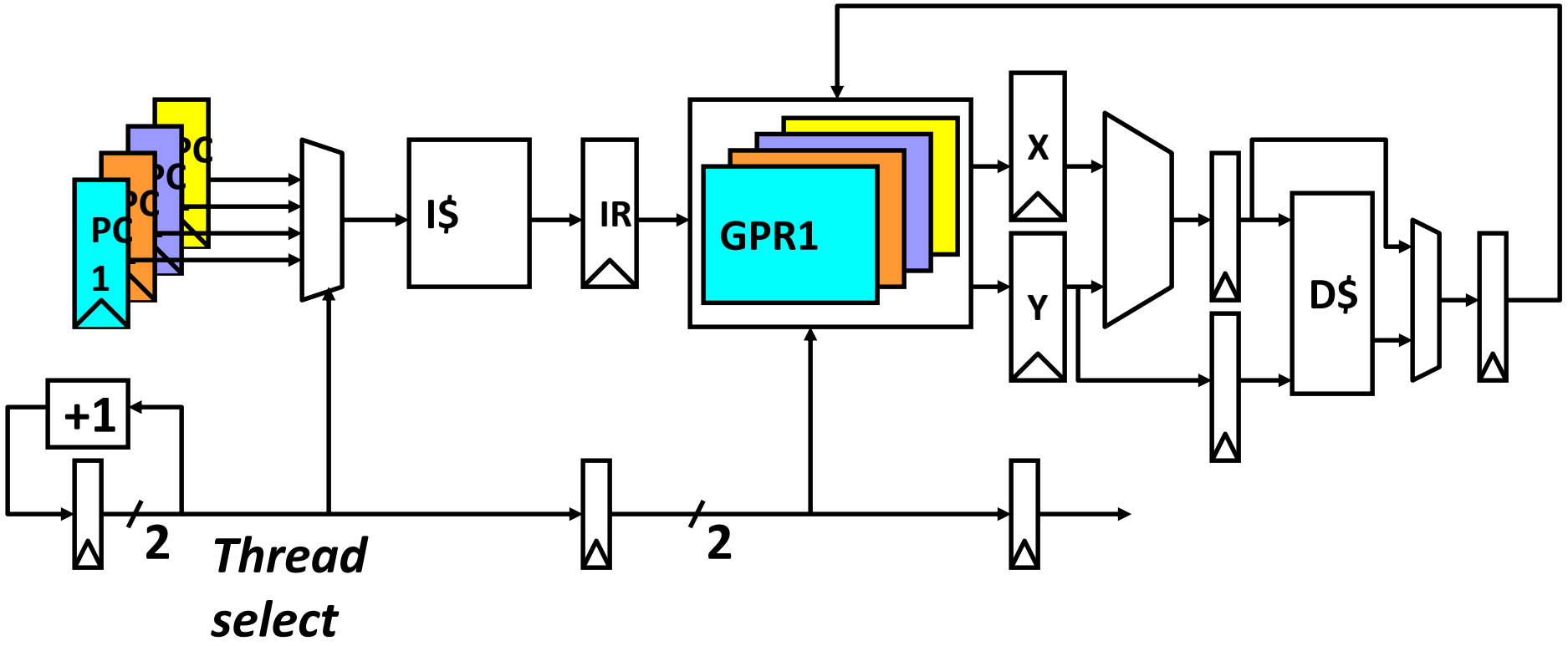
T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)



CDC 6600 Peripheral Processors
(Cray, 1964)

- First multithreaded hardware
- 10 “virtual” I/O processors
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Accumulator-based instruction set to reduce processor state

Simple Multithreaded Pipeline



- 必须传递线程选择信号以保证各流水段读写的正确性
- 从软件（包括OS）的角度看 好像存在多个CPU（虽然针对每个线程，CPU似乎运行的慢一些

Multithreading Costs

- 每个线程需要拥有自己的用户态信息（user state)：包括PC、GPRs

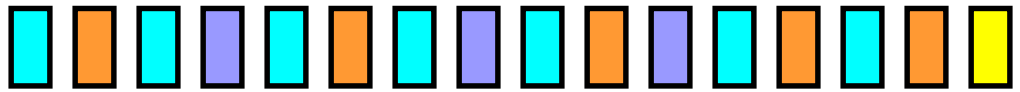
- 需要自己的系统态信息（system state）
  - 虚拟存储的页表基地址寄存器（Virtual-memory page-table-base register）
  - 异常处理寄存器（Exception-handling registers）

- *其他开销:*
  - 需要处理由于线程竞争导致的Cache/TLB冲突 或 需要更大的cache/TLB 容量
  - 更多的OS调度开销

Thread Scheduling Policies

- 固定交叉模式 *(CDC 6600 PPUs, 1964)*
  - 针对N个线程，每个线程每隔N个周期执行一条指令
  - 如果流水线的某一时隙(slot)其对应线程未就绪，插入pipeline bubble

- 软件控制的交叉模式 *(TI ASC PPUs, 1971)*
  - OS 为N个线程分配流水线的S个pipeline slots
  - 硬件针对S个slots采用固定交叉模式执行相应的线程



- 硬件控制的线程调度 *(HEP, 1982)*
  - 硬件跟踪哪些线程处于ready状态
  - 根据优先级方案选择线程执行

Denelcor HEP

(Burton Smith, 1982)

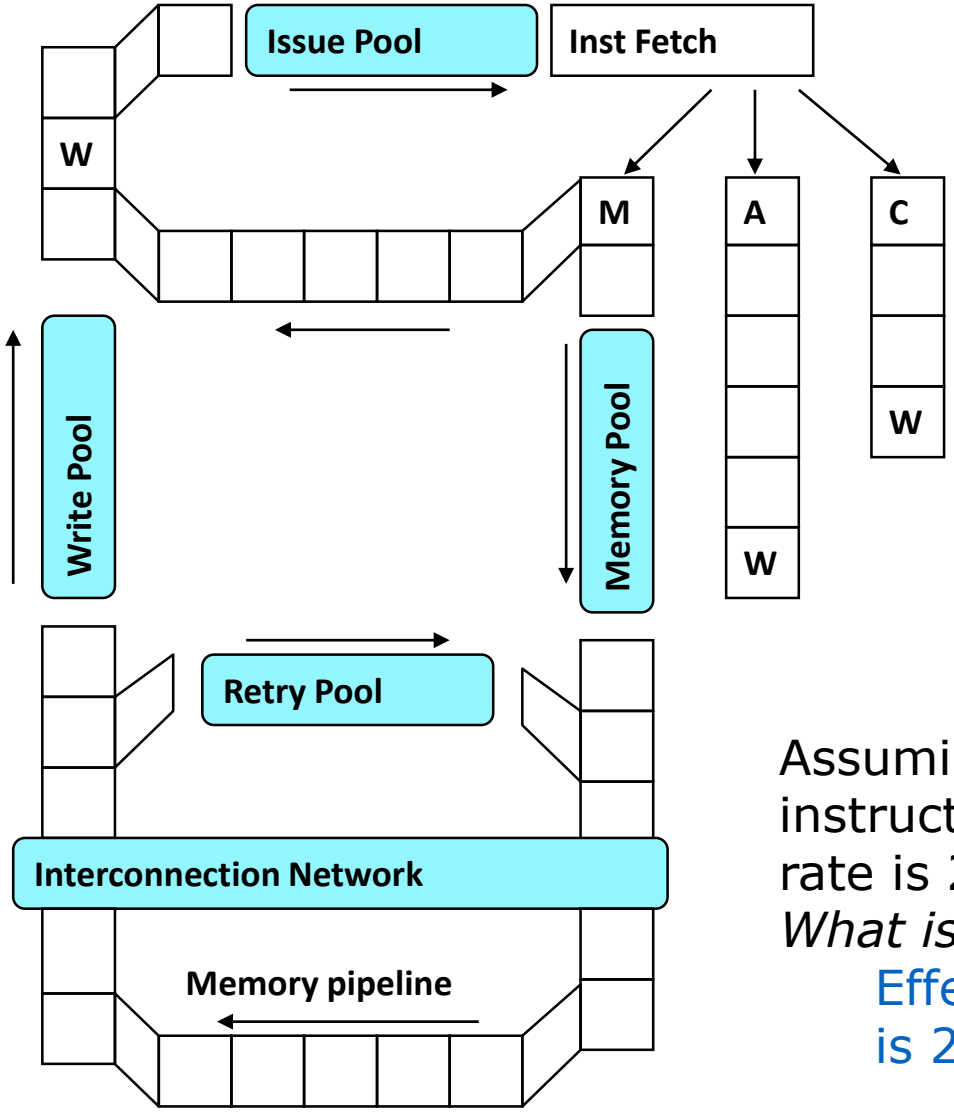
First commercial machine to use hardware threading in main CPU

- 120 threads per processor
- 10 MHz clock rate
- Up to 8 processors
- precursor to Tera MTA (Multithreaded Architecture)

Tera MTA (1990-)

- Up to 256 processors
- Up to 128 active threads per processor
- Processors and memory modules populate a sparse 3D torus interconnection fabric
- Flat, shared main memory
  - No data cache
  - Sustains one main memory access per cycle per processor
- GaAs logic in prototype, 1KW/processor @ 260MHz
  - Second version CMOS, MTA-2, 50W/processor
  - New version, XMT, fits into AMD Opteron socket, runs at 500MHz

MTA Pipeline



- Every cycle, one VLIW instruction from one active thread is launched into pipeline
- Instruction pipeline is 21 cycles long
- Memory operations incur ~150 cycles of latency

Assuming a single thread issues one instruction every 21 cycles, and clock rate is 260 MHz…

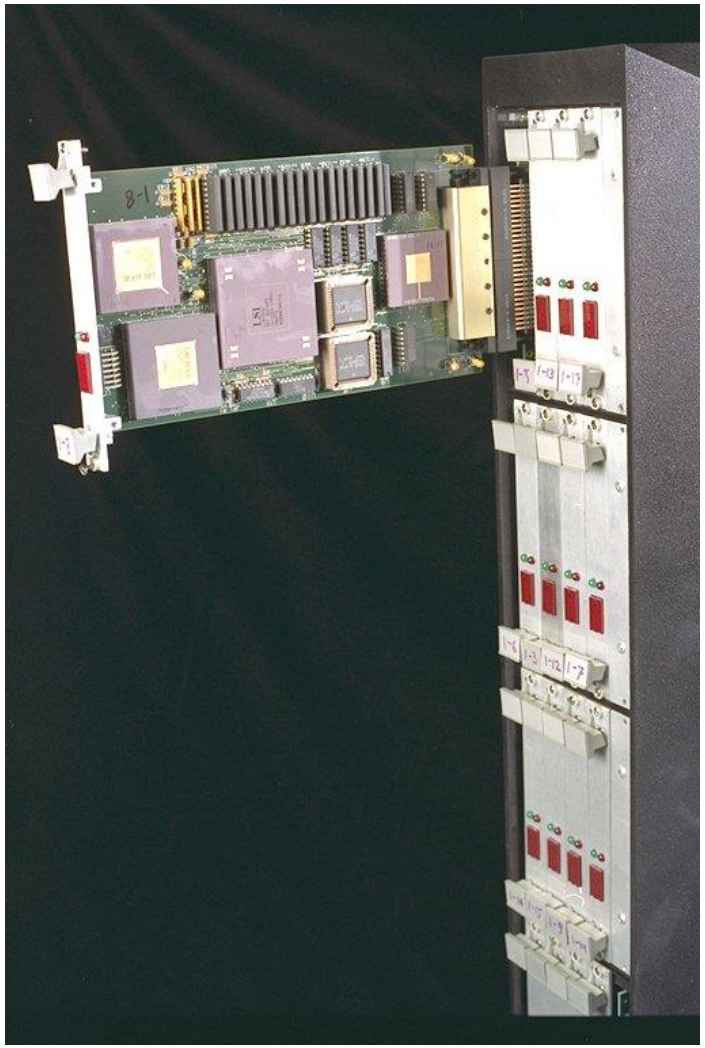
What is single-thread performance?

Effective single-thread issue rate is 260/21 = 12.4 MIPS

Coarse-Grain Multithreading

- Tera MTA 适用于大规模数据集并且数据本地性较低的应用
  - 没有数据Cache
  - 需要大量的并行线程以隐藏存储器访问延迟
- 其他数据本地性较好的应用
  - 如果Cache命中，流水线上的停顿较少
  - 仅需要添加较少的线程来隐藏偶尔的Cache失效导致的延迟
  - Cache失效时，将该线程换出
- 当某一线程的执行存在长延迟操作时，选择另一线程执行，并切换线程上下文

MIT Alewife (1990)



- Modified SPARC chips
  - register windows hold different thread contexts
- Up to four threads per node
- Thread switch on local cache miss

IBM PowerPC RS64-IV (2000)

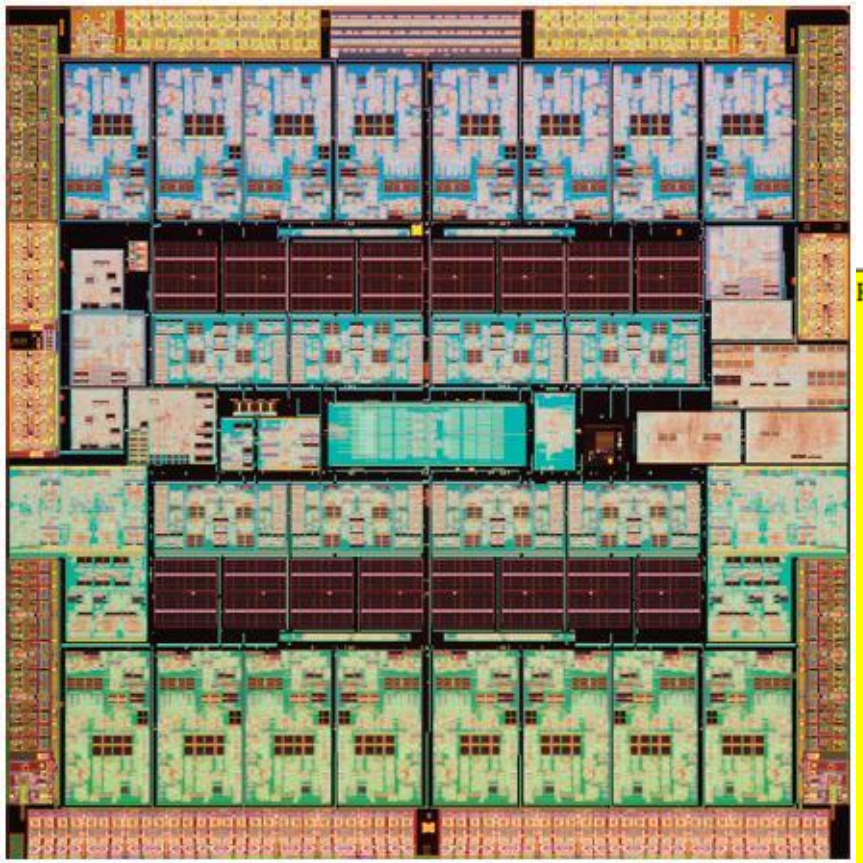
- Commercial coarse-grain multithreading CPU
- Based on PowerPC with quad-issue in-order five-stage pipeline
- Each physical CPU supports two virtual CPUs
- On L2 cache miss, pipeline is flushed and execution switches to second thread
  - short pipeline minimizes flush penalty (4 cycles), small compared to memory access latency
  - flush pipeline to simplify exception handling

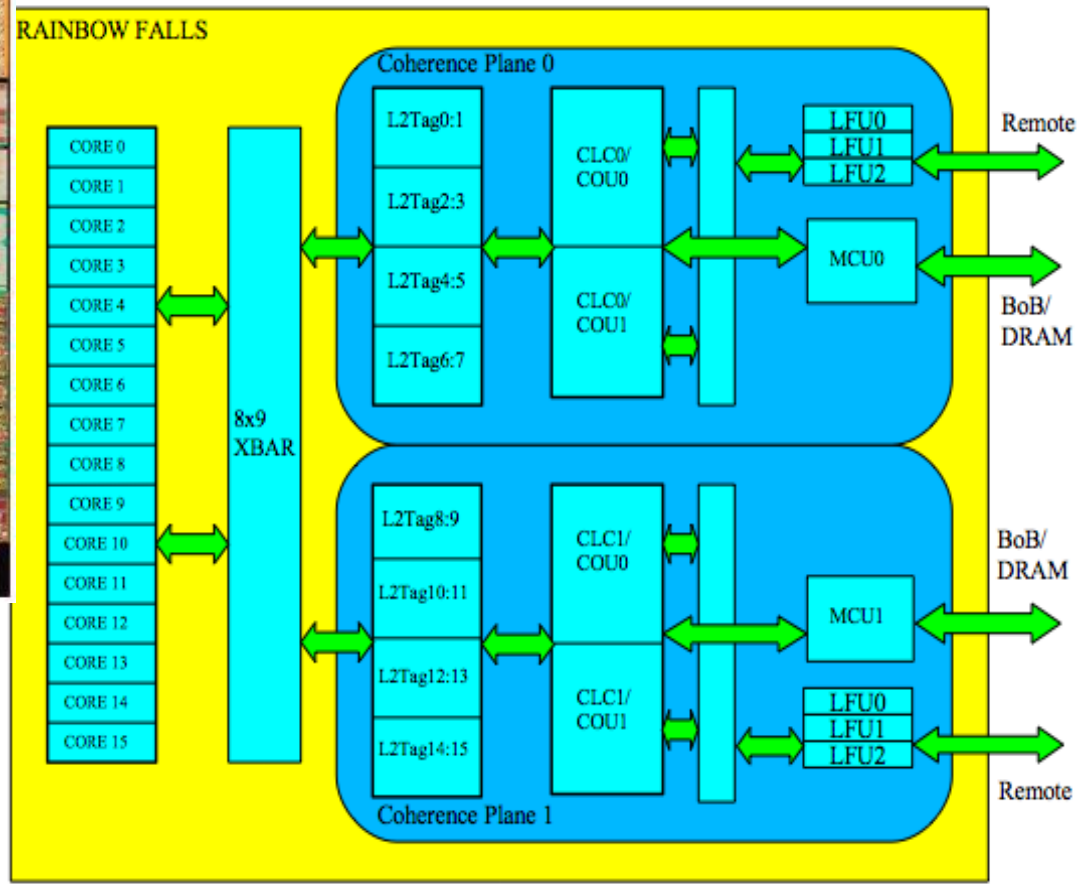
Oracle/Sun Niagara processors

- Target is datacenters running web servers and databases, with many concurrent requests
- Provide multiple simple cores each with multiple hardware threads, reduced energy/operation though much lower single thread performance

- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core
- T4 [2011], 8 cores, 8 threads/core
- T5 [2012], 16 cores, 8 threads/core

Oracle/Sun Niagara-3, "Rainbow Falls" 2009

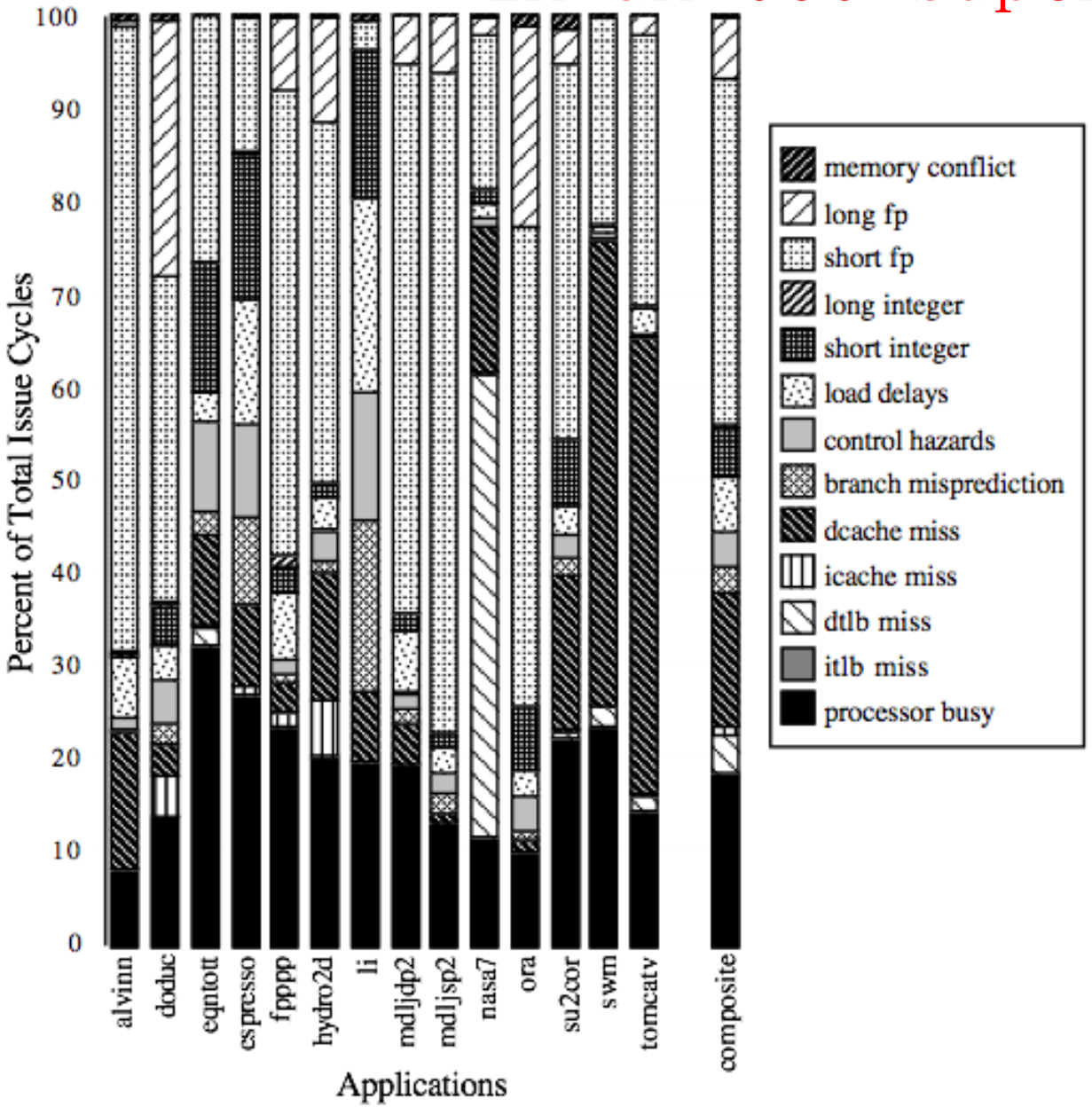




Simultaneous Multithreading (SMT) for OoO Superscalars

- “vertical”多线程：即某一时段每条流水线上运行一个线程
- SMT 使用OoOSuperscalar细粒度控制技术在相同时钟周期运行多个线程的指令，以更好的利用系统资源

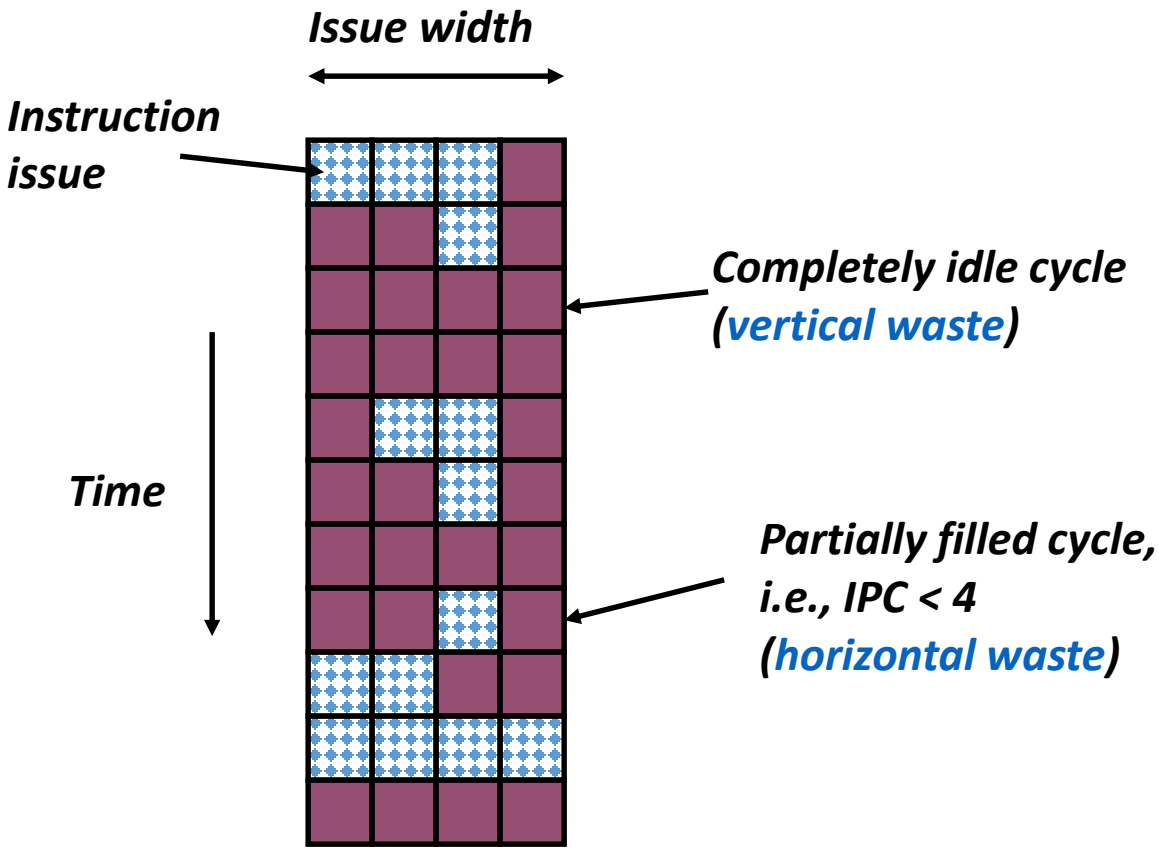
For most apps, most execution units lie idle in an OoO superscalar



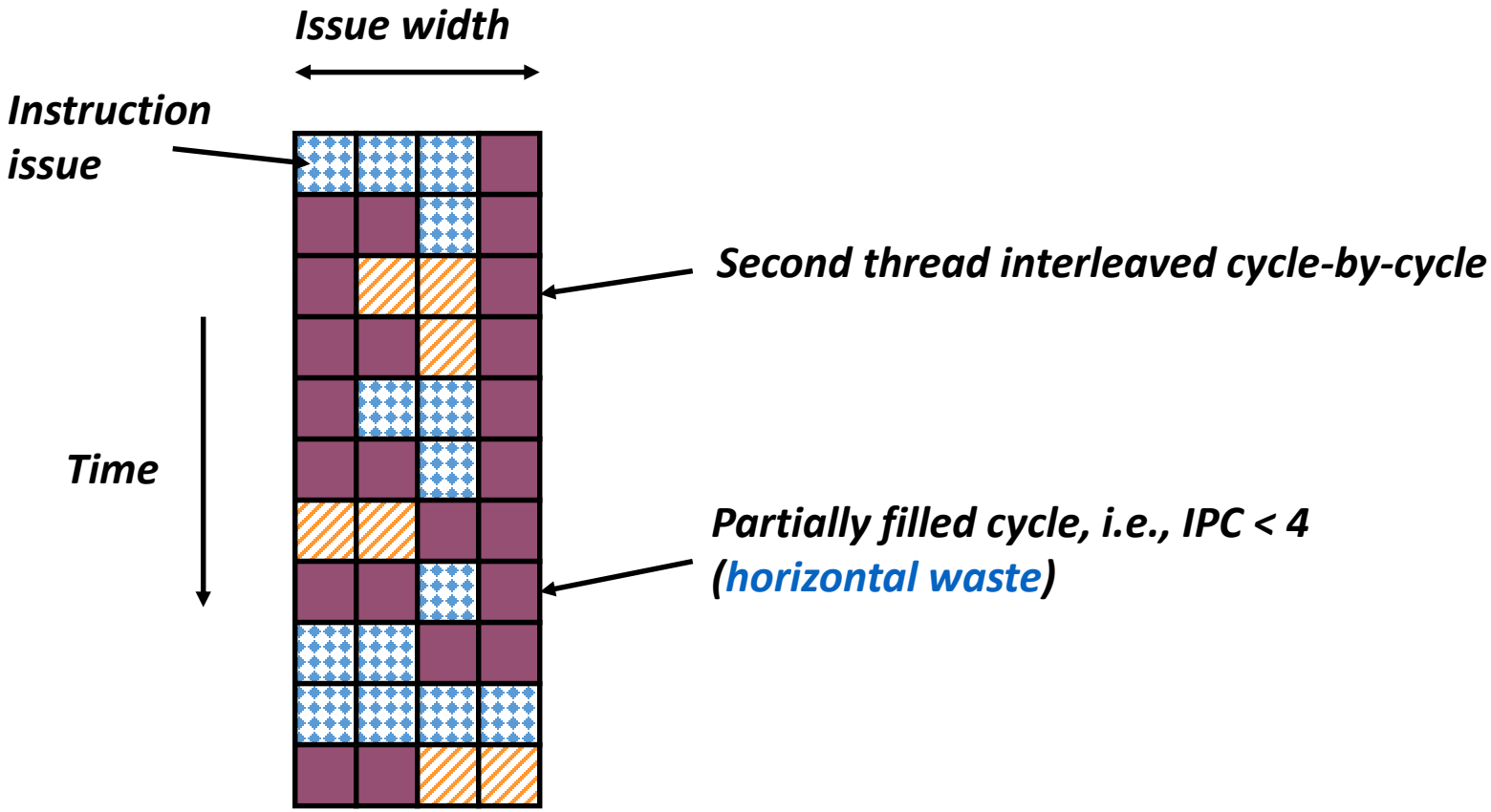
For an 8-way superscalar.

From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism", ISCA 1995.

Superscalar Machine Efficiency

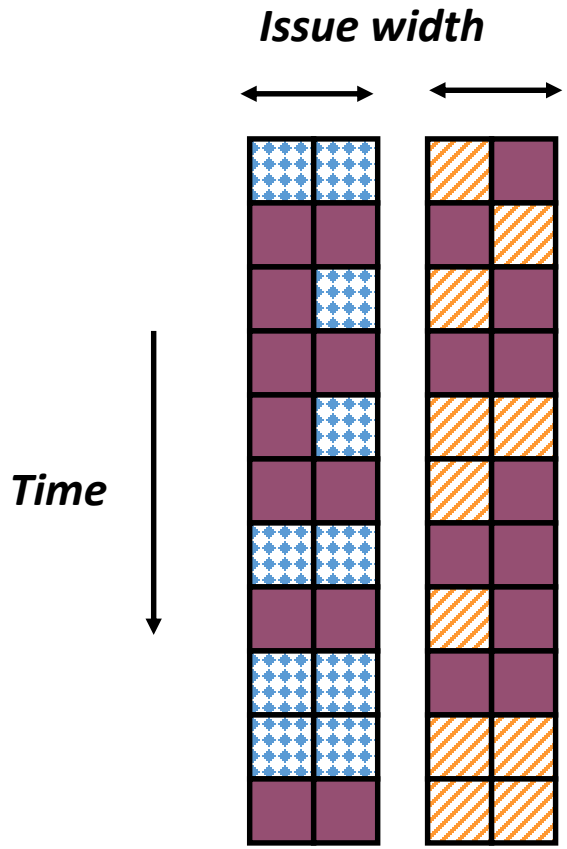


Vertical Multithreading



- 如果基于细粒度的时钟周期交叉运行模式，结果怎样？
  - removes vertical waste, but leaves some horizontal waste

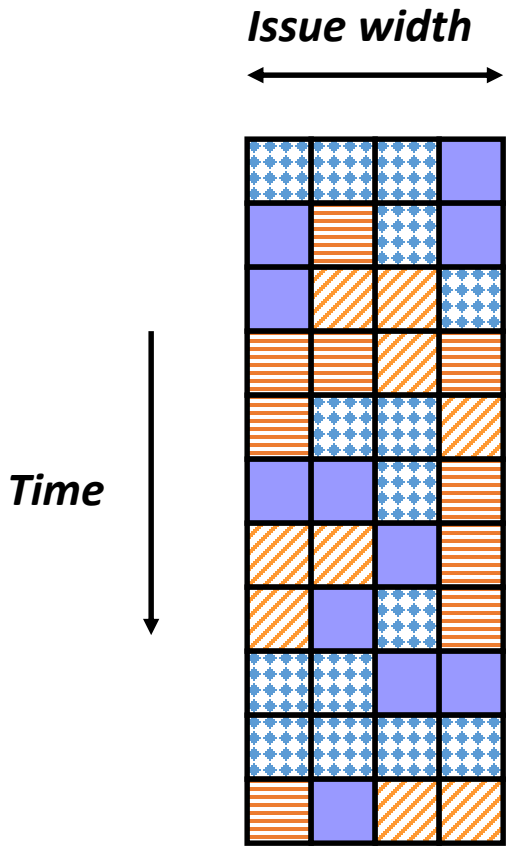
Chip Multiprocessing (CMP)



- 分成两个处理器后的效果？
  - reduces horizontal waste,
  - leaves some vertical waste, and
  - puts upper limit on peak throughput of each thread.

Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- 采用多线程交叉模式使用多个issue slots

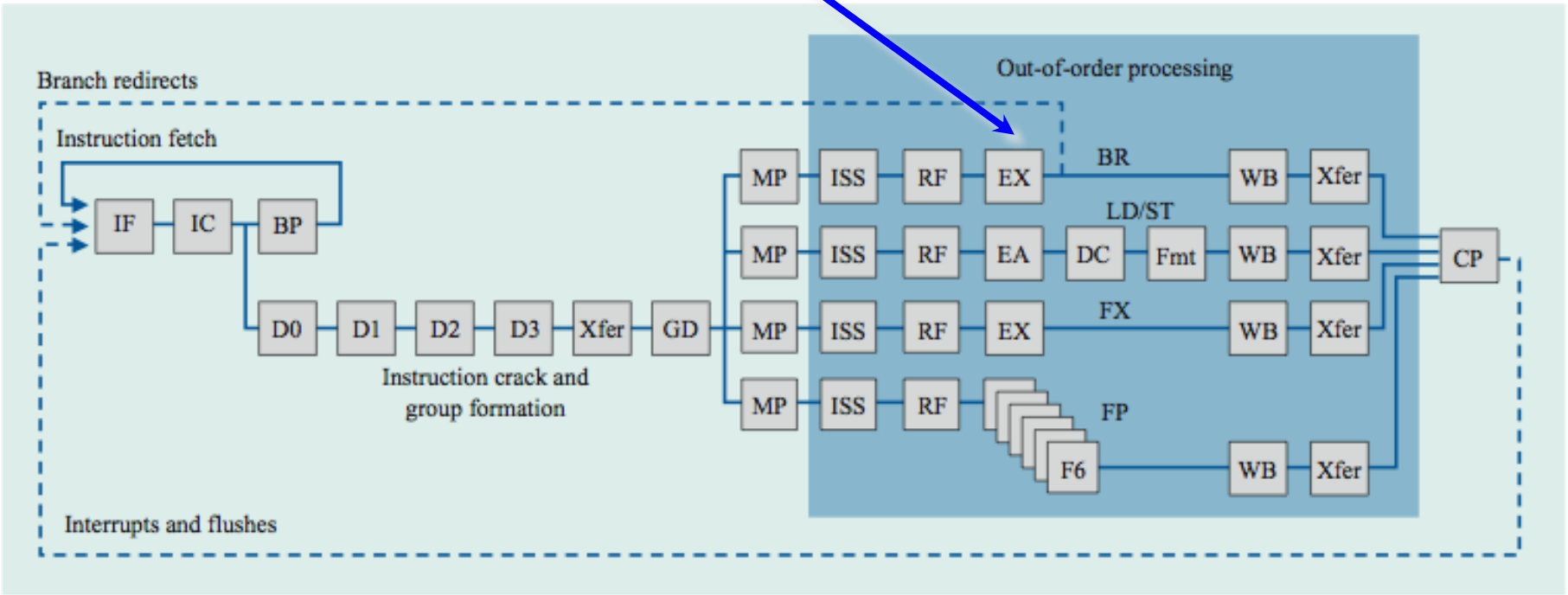
O-o-O Simultaneous Multithreading

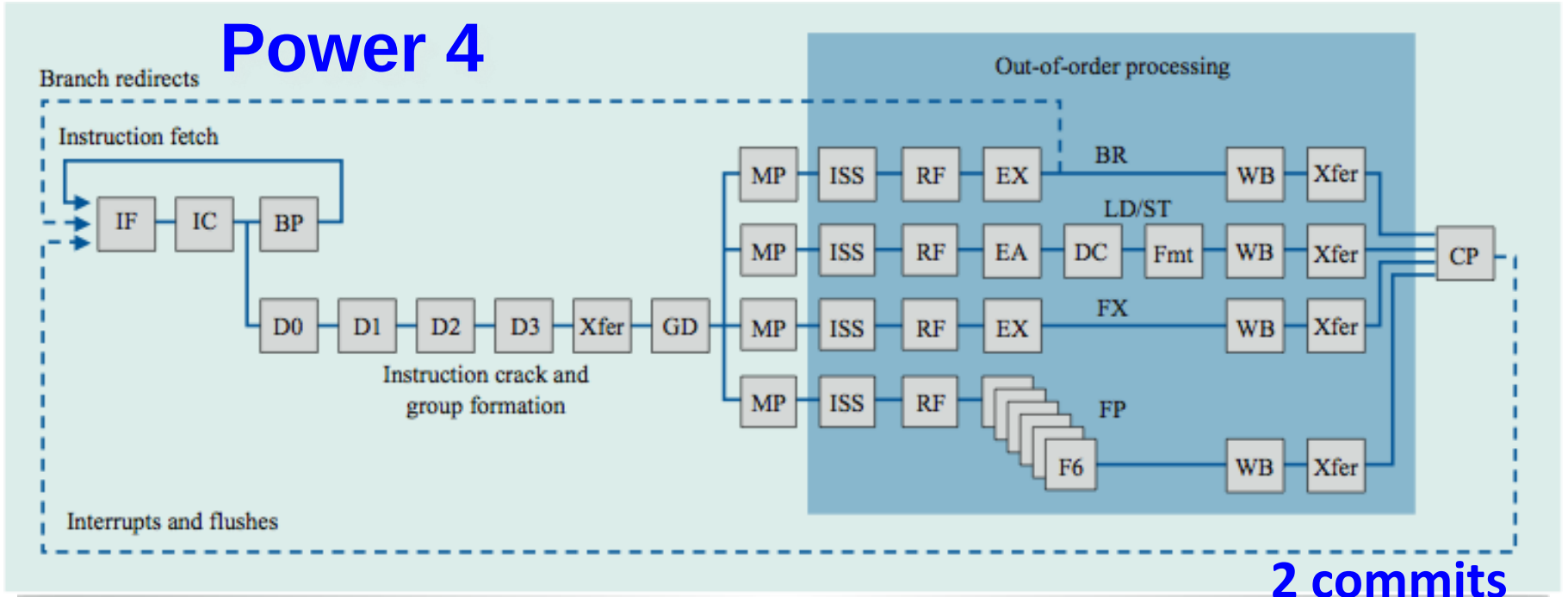
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

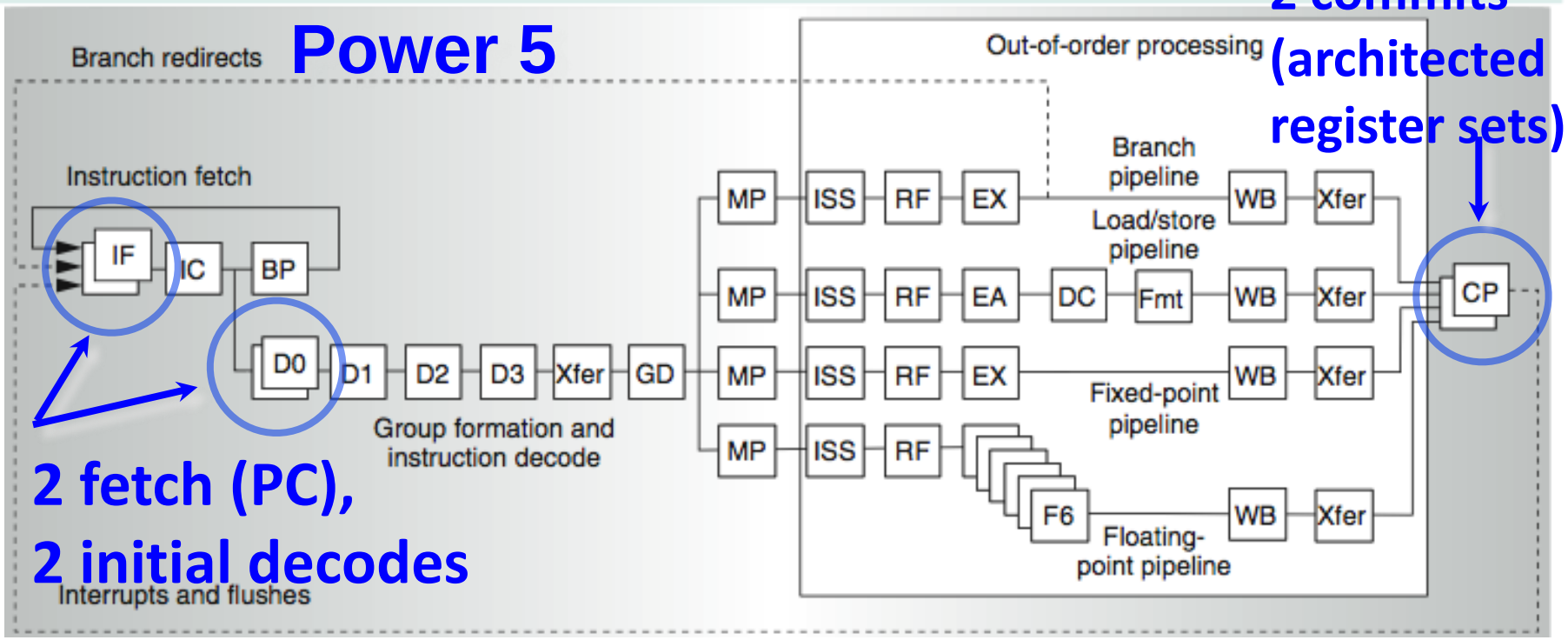
- 增加多上下文切换以及取指引擎可以从多个线程取指令，并可同时发射
- 使用OoO superscalar处理器的发射宽度，从发射队列中选择指令发射，这些指令可来源于多个线程
- OoO 指令窗口已经具备从多个线程调度指令的绝大多数电路
- 任何单线程程序可以充分使用整个系统资源

IBM Power 4

Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.



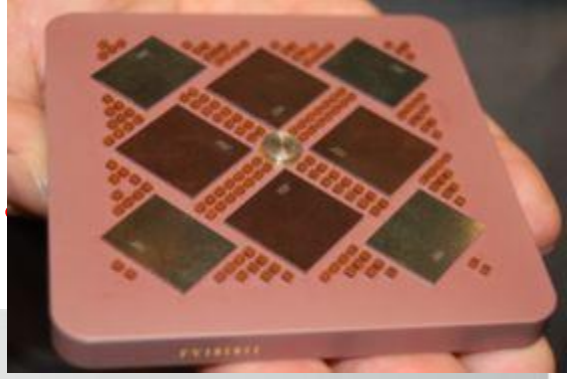


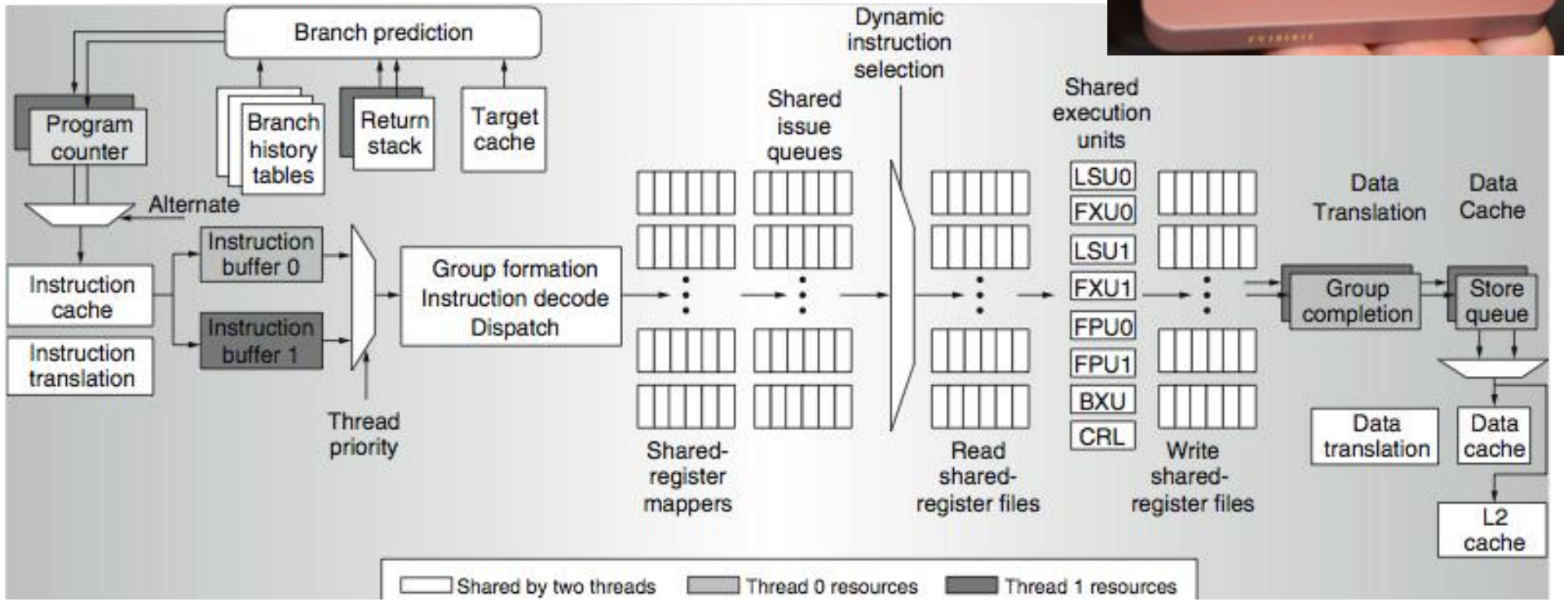


2 commits (architected register sets)

2 fetch (PC), 2 initial decodes

Power 5 data flow





Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Changes in Power 5 to support SMT

- Increased associativity of L1 instruction cache and the instruction address translation buffers
- Added per-thread load and store queues
- Increased size of the L2 (1.92 vs. 1.44 MB) and L3 caches
- Added separate instruction prefetch and buffering per thread
- Increased the number of virtual registers from 152 to 240
- Increased the size of several issue queues
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

Pentium-4 Hyperthreading (2002)

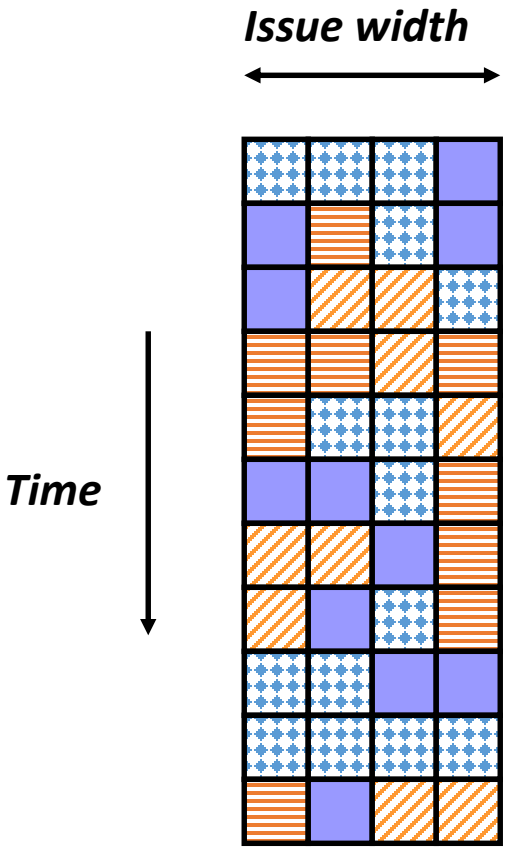
- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading
- Hyperthreading dropped on OoO P6 based followons to Pentium-4 (Pentium-M, Core Duo, Core 2 Duo), until revived with Nehalem generation machines in 2008.
- Intel Atom (in-order x86 core) has two-way vertical multithreading

Initial Performance of SMT

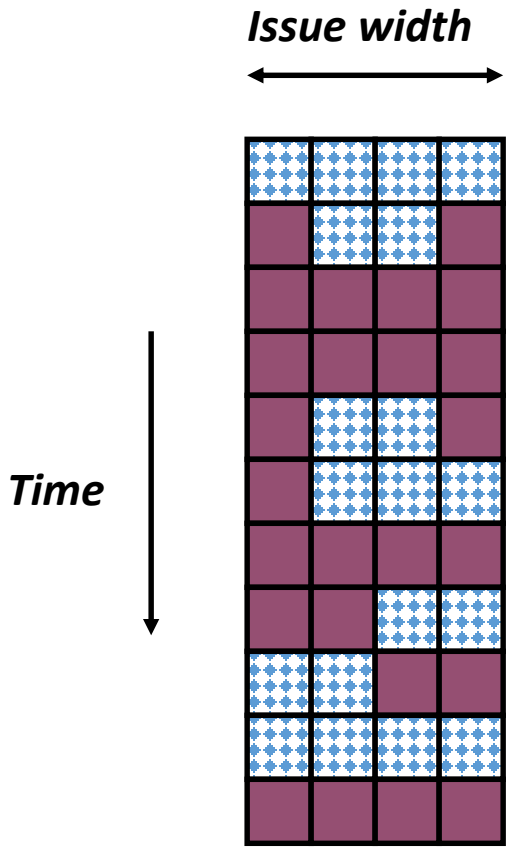
- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium 4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average was 1.20
- Power 5, 8-processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- Power 5 running 2 copies of each app speedup between 0.89 and 1.41
  - Most gained some
  - Fl.Pt. apps had most cache conflicts and least gains

SMT adaptation to parallelism type

For regions with high thread level parallelism (TLP) entire machine width is shared by all threads

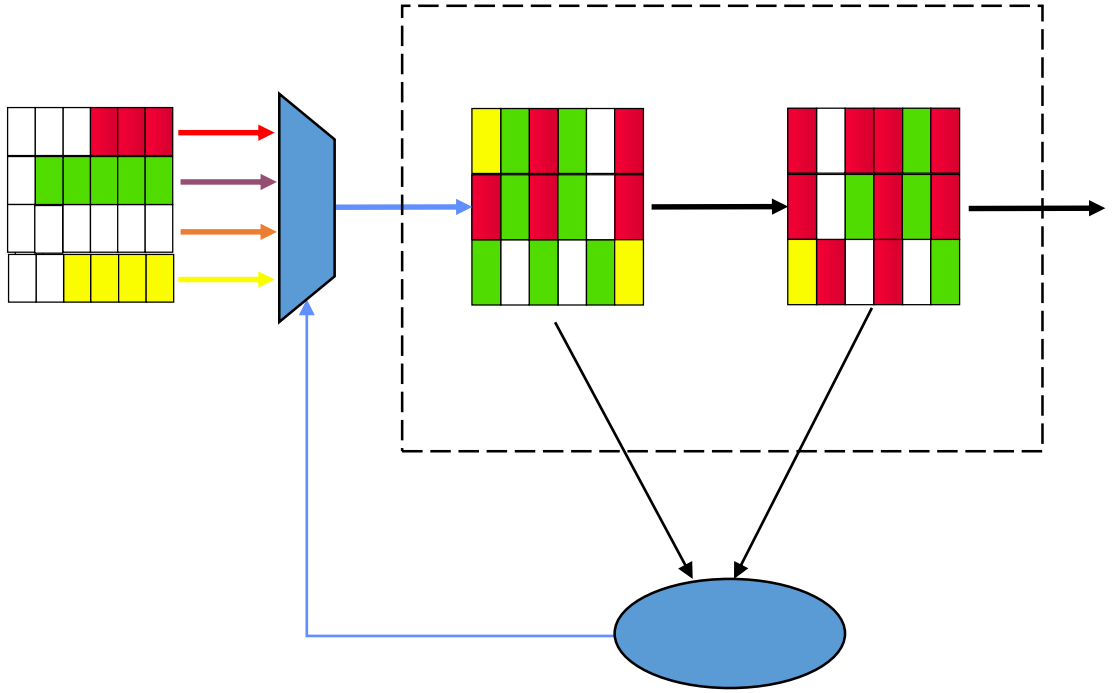


For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)



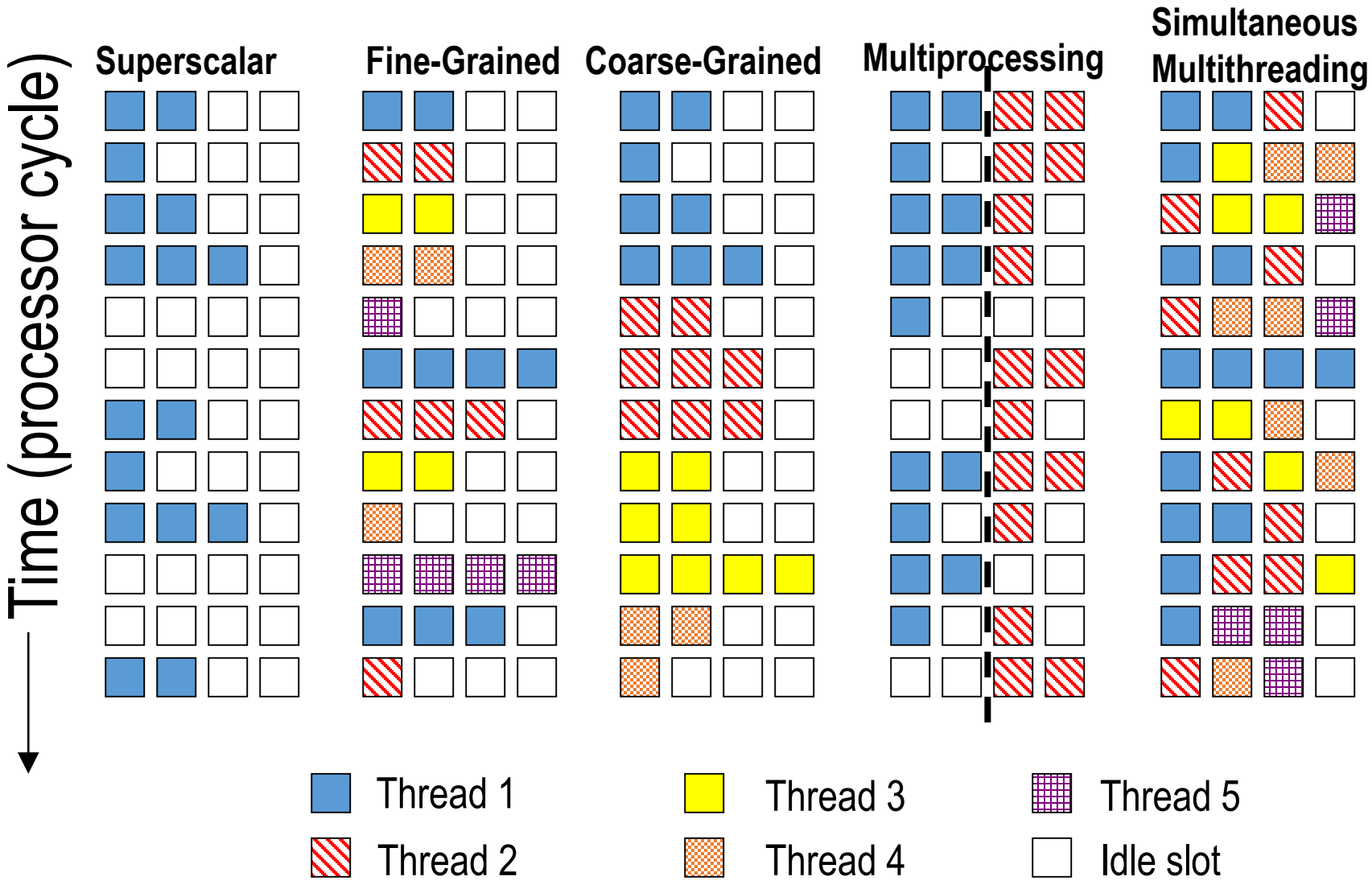
Icount Choosing Policy

Fetch from thread with the least instructions in flight.



Why does this enhance throughput?

Summary: Multithreaded Categories



Acknowledgements

- These slides contain material developed and copyright by:
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - David Patterson (UCB)
  - Chenxi Zhang (Tongji)
- UCB material derived from course CS152、CS252