计算机体系结构

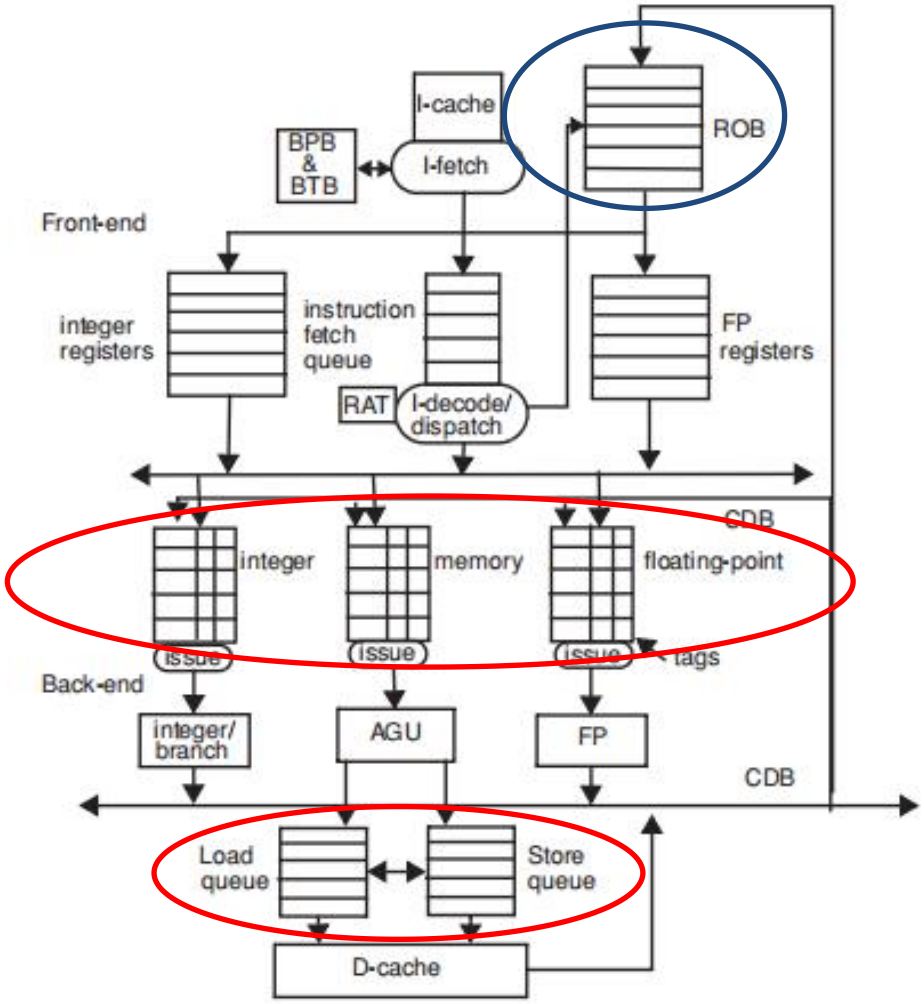
周学海
xhzhou@ustc.edu.cn
0551-63606864
中国科学技术大学

Review: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
- **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
  - 预测错误或有异常（中断）时，刷新ROB
  - P191 Figure 3.14 (英文版)，P141 Figure 3-9 (中文版)

Review: Reorder Buffer

- **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)
- **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（按程序执行逻辑序提交）
  - 3 个域: 指令类型,目的地址, 值



Review

- **存储器访问的冲突（歧义）消解**
  - 非投机方式的冲突消解
    - Total Ordering
    - Partial Ordering
      - Load指令前的store指令已经完成了地址计算，有可能乱序执行load操作
    - Load Ordering, Store Ordering
      - Load指令前的存储器访问指令已经完成了地址计算，load队头的load操作有可能在store指令之前执行访存操作。
  - 投机方式的执行
    - Store Ordering
    - 假设Load操作与之前未计算出有效地址的store操作无关。
- **问题：给出四种访问方式挖掘并行性的能力排序。**

第5章 指令级并行

5.1 指令级并行的基本概念及静态指令流调度

ILP及挑战性问题

软件方法挖掘指令集并行

基本块内的指令集并行

5.2硬件方法挖掘指令级并行

5.2-1 指令流动态调度方法之一：Scoreboard

5.2-2 指令流动态调度方法之二：Tomasulo

5.3 分支预测方法

5.4 基于硬件的推测执行

5.5 存储器访问冲突消解及多发射技术

5.6 多线程技术

5.5-1 存储器访问冲突消解

5.5-2 多发射技术

Memory Disambiguation

TABLE 6.1: Memory disambiguation schemes.

NAME	SPECULATIVE	DESCRIPTION
Total Ordering	No	All memory accesses are processed in order.
Partial Ordering	No	All stores are processed in order, but loads execute out of order as long as all previous stores have computed their address.
Load Ordering Store Ordering	No	Execution between loads and stores is out of order, but all loads execute in order among them, and all stores execute in order among them.
Store Ordering	Yes	Stores execute in order, but loads execute completely out of order.

- **非投机方式的基本原则：当前存储器指令之前的store指令计算存储器地址后，才能执行当前的存储器操作**

Tomasulo Loop Example

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

- **假设**
  - Load和store部件：计算访存地址 需要 2 cycle；对Cache访问 需要 1个cycle
  - 浮点操作执行：需要6个cycle
  - Store操作内部分解为两个操作操作：SD-A 计算访存地址；SD-D 对Cache访问
  - 其他整型类执行：需要2个cycle

Tomsasulo算法执行示例（Total Ordering - 无推断）

		Issue	Exe Start	Exe End	Cache	CDB	备注
I1	LD F0, 0(R1)	1	2	3	（4）	（5）	
I2	MULTD F4, F0, F2	2	6	11	--	（12）	等待F0 （I1)
I3	SD-A, F4, 0(R1)	3	4	5	---	---	
I4	SD-D F4, 0(R1)	4	13	14	(15)	---	等待F4 (I2)
I5	SUBI R1, R1, #8	5	6	7	---	(8)	
I6	BNEZ R1, Loop	6	9	10		(11)	等待R1 (i5)
I7	LD F0, 0(R1)	12	14	15	(16)	(17)	
I8	MULTD F4, F0, F2	13	18	23	--	(24)	
I9	SD-A, F4, 0(R1)	14	15	16	---	---	
I10	SD-D F4, 0(R1)	15	25	26	(27)	(28)	
I11	SUBI R1, R1, #8	16	17	18	---	(19)	
I12	BNEZ R1, Loop	17	20	21	---	(22)	

Load Ordering. Store Ordering

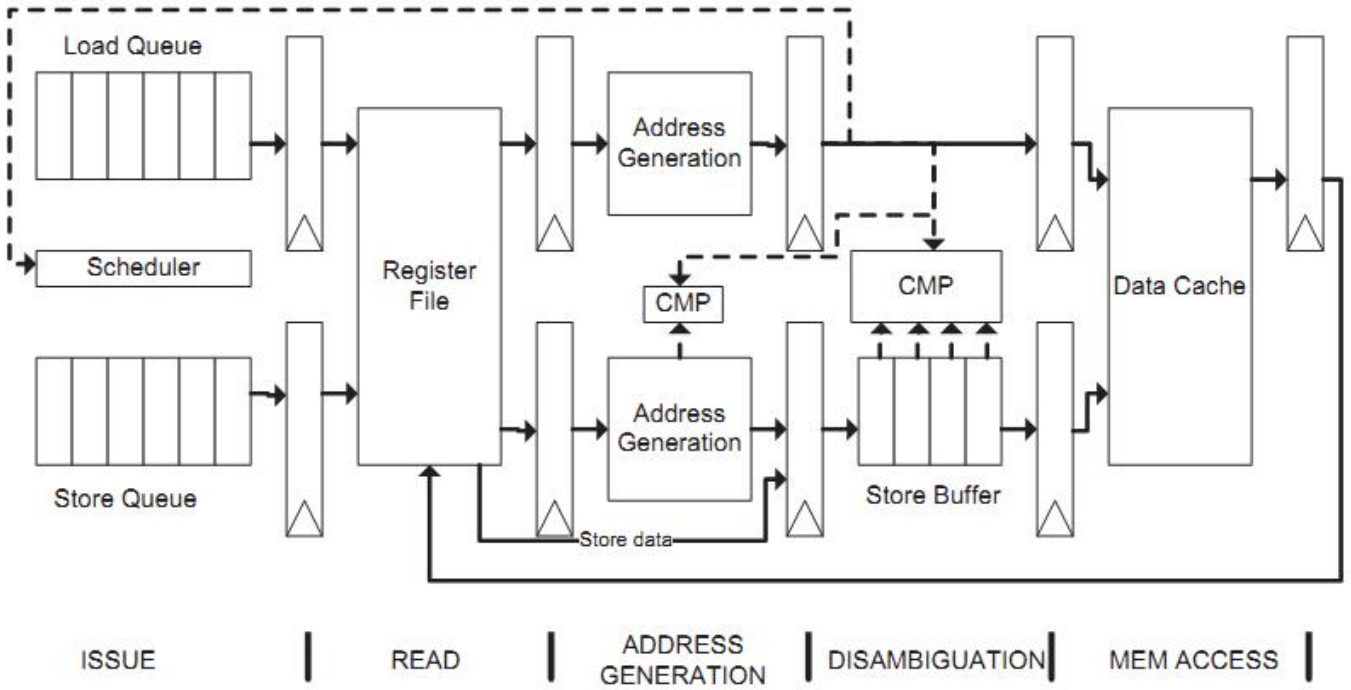


FIGURE 6.7: Schematics of the AMD K6 pipeline to implement a load-ordering store-ordering memory disambiguation policy.

Load queue: 以程序序存储load指令，按照FIFO方式流出

Address generation: 生成存储器访问有效地址

Store queue: 以程序序存储store指令，按照FIFO方式流出

Store buffer: 以程序序保留store操作（有效地址，值），一直到其成为最早的store操作，才实际更新存储器

Store 操作

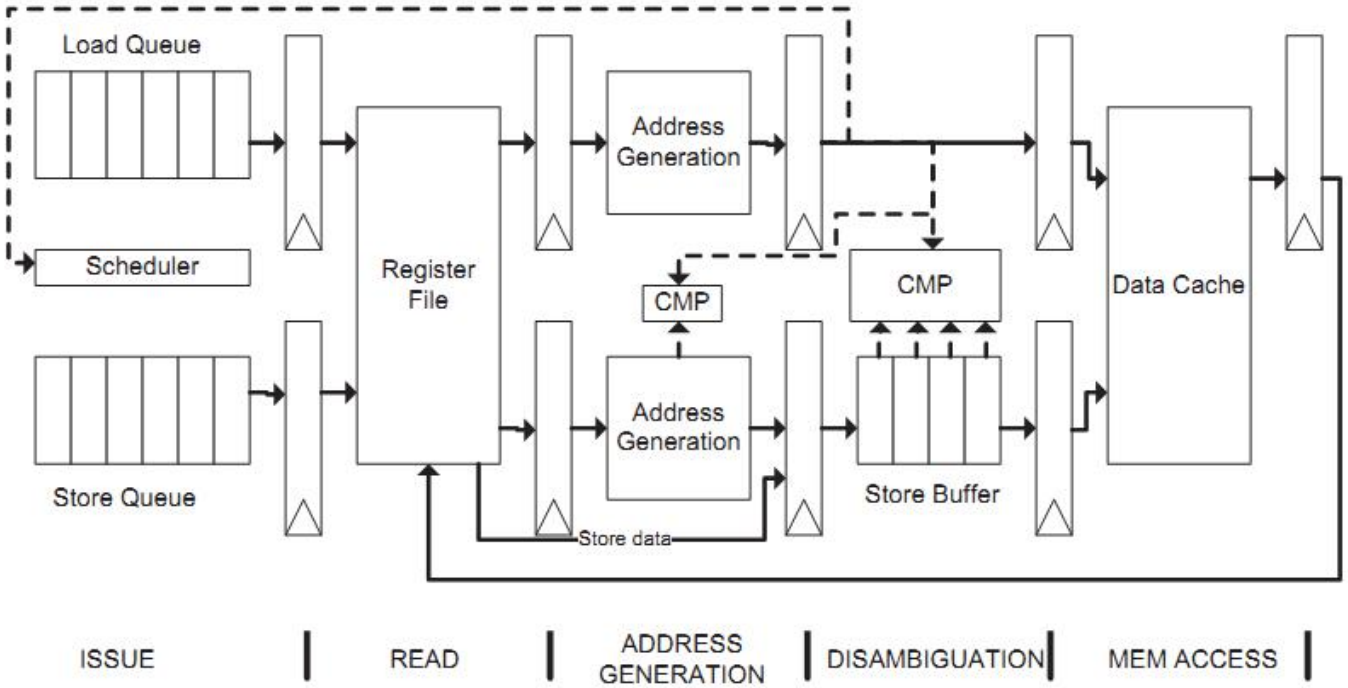


FIGURE 6.7: Schematics of the AMD K6 pipeline to implement a load-ordering store-ordering memory disambiguation policy.

- **处于Store队列对头，并且计算地址的操作数准备好，就向前流动，在Address Generation阶段计算地址**
- **若要存储的数据准备好，则读该数据，若未准备好，则流水线停顿**
- **Disambiguation阶段：StoreBuffer中按程序序，执行最先的Store操作**

Load 操作

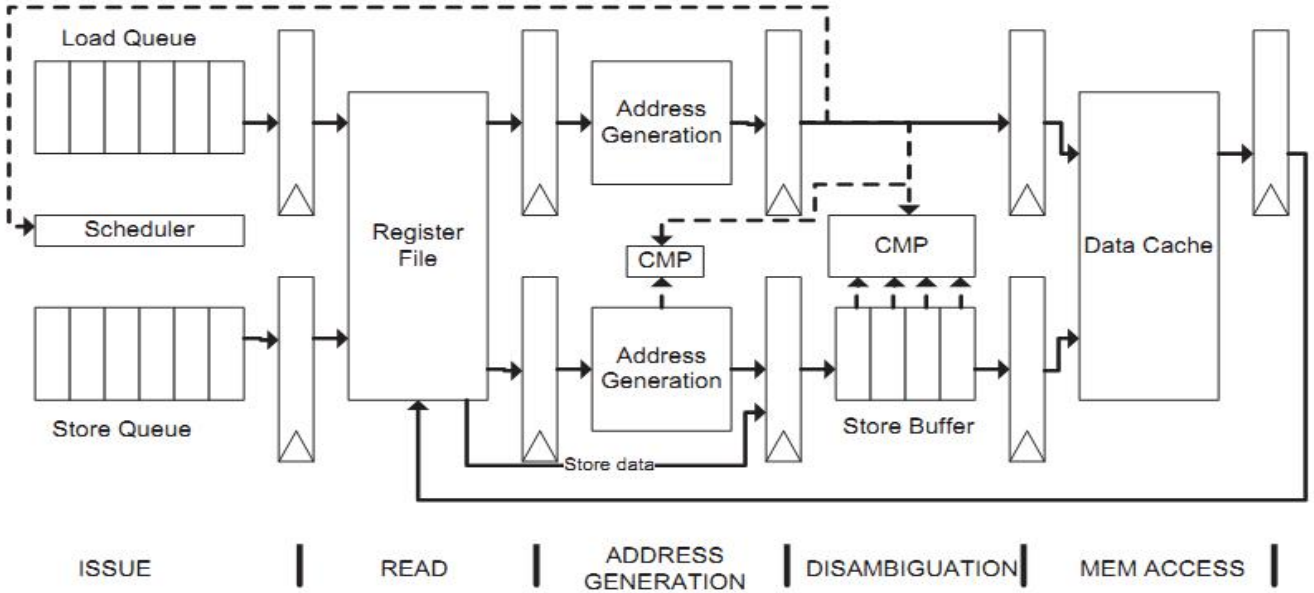


FIGURE 6.7: Schematics of the AMD K6 pipeline to implement a load-ordering store-ordering memory disambiguation policy.

- **处于Load队列对头，并且计算地址的操作数准备好，就向前流动；在Address Generation阶段计算地址**
- **Disambiguation阶段：Load操作可以继续前行需同时满足的三个条件**
  - 在StoreBuffer中比该Load操作早的Store的地址与该Load地址不同
  - 如果正在进行地址计算的Store操作比该Load操作早，进行部分地址比较时，部分地址不同
  - 在StoreQueue中不存在比该Load早的Store操作。

Load ording，store ording - 分支预测

		Issue	Exe Start	Exe End	Cache	CDB	commit	备注
I1	LD F0, 0(R1)	1	2	3	（4）	（5）	6	
I2	MULTD F4, F0, F2	2	6	11	--	（12）	13	等待F0 （I1)
I3	SD-A, F4, 0(R1)	3	4	5	---	---	14	
I4	SD-D F4, 0(R1)	4	13	14	(15)	---	16	等待F4 (I2)
I5	SUBI R1, R1, #8	5	6	7	---	(8)	17	
I6	BNEZ R1, Loop	6	11	12		(13)	18	CDB冲突
I7	LD F0, 0(R1)	7	8	9	(10)	(11)	19	
I8	MULTD F4, F0, F2	8	12	17	--	(18)	20	
I9	SD-A, F4, 0(R1)	9	10	11	---	---	21	
I10	SD-D F4, 0(R1)	10	19	20	(21)	(22)	23	等待F4 (I8)
I11	SUBI R1, R1, #8	11	12	13	---	(14)	24	
I12	BNEZ R1, Loop	12	15	16	---	(17)	25	

Partial Ordering （MIPS R10000）

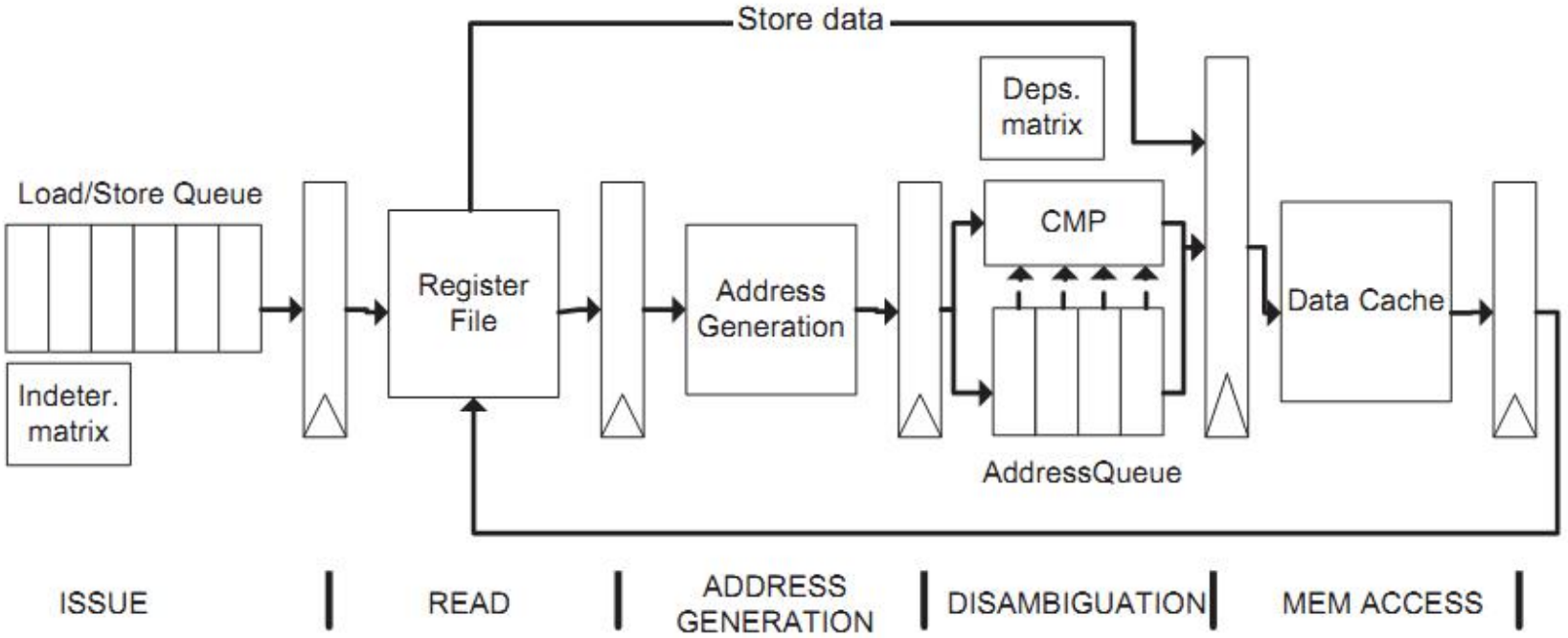


FIGURE 6.8: Schematics of the MIPS R10000 pipeline to implement the partial ordering memory disambiguation policy.

Load/store queue: 长度为16的队列，存储load/store指令，直到其操作数准备好。
Address generation: 计算存储器操作的有效地址

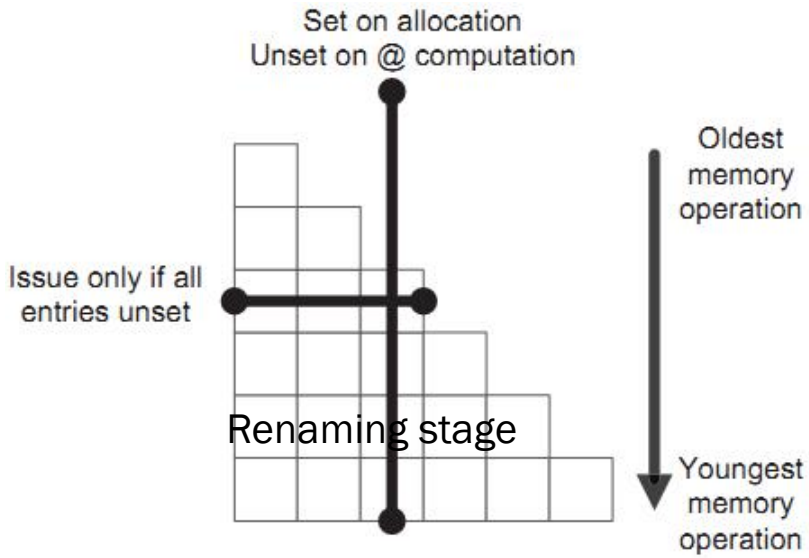


FIGURE 6.9: Example of a 6-entry indetermination matrix.

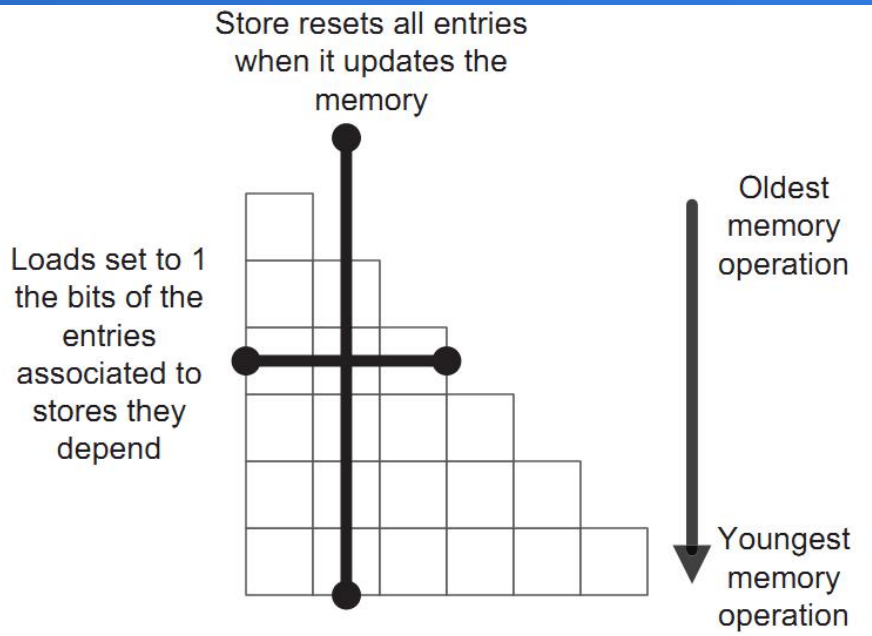


FIGURE 6.10: Example of a 6-entry dependency matrix.

- **Indetermination matrix**: 16x16 矩阵（half），每行每列代表队列中的存储器操作。当有存储器操作**进入队列**时，对应**列置位**。当存储器指令**计算有效地址**时，对应**列复位**。计算有效地址的条件之一，该操作对应行中非对角线元素为0
- **Dependency matrix**: 16x16 矩阵，每行每列对应load/store队列的存储器操作。Load操作如果依赖于前面的store，则将该load操作的行中对应该store的**列置位**。Store操作更新存储器时，将**列复位**该store操作对应的列。只有当load操作对应的行全0时，方可执行load操作
- **Address queue**:保存访问cache的loads/store操作的地址。如果是load操作，除了保存地址，还需要比较所有在该load操作之前的store的地址，如果匹配，则对dependency matrix对应位置位。

Speculative Memory Disambiguation

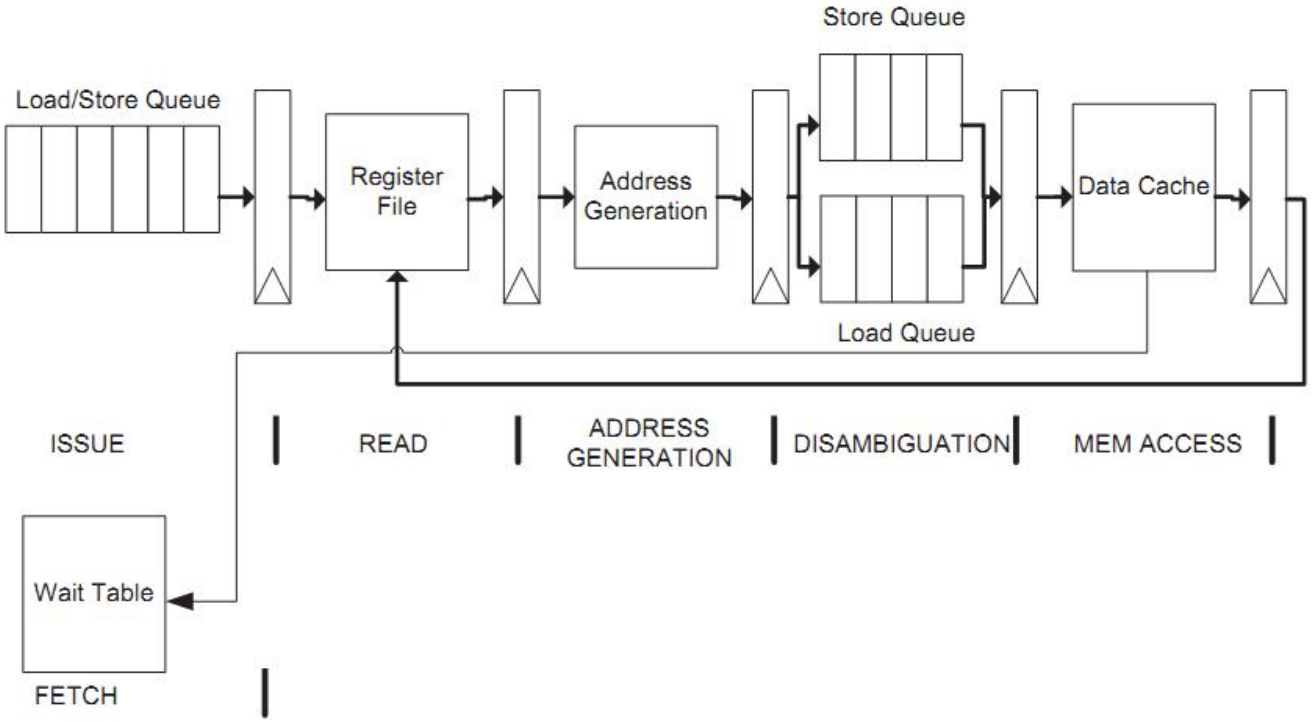


FIGURE 6.11: Schematics of the Alpha 21264 pipeline to implement a speculative memory disambiguation policy.

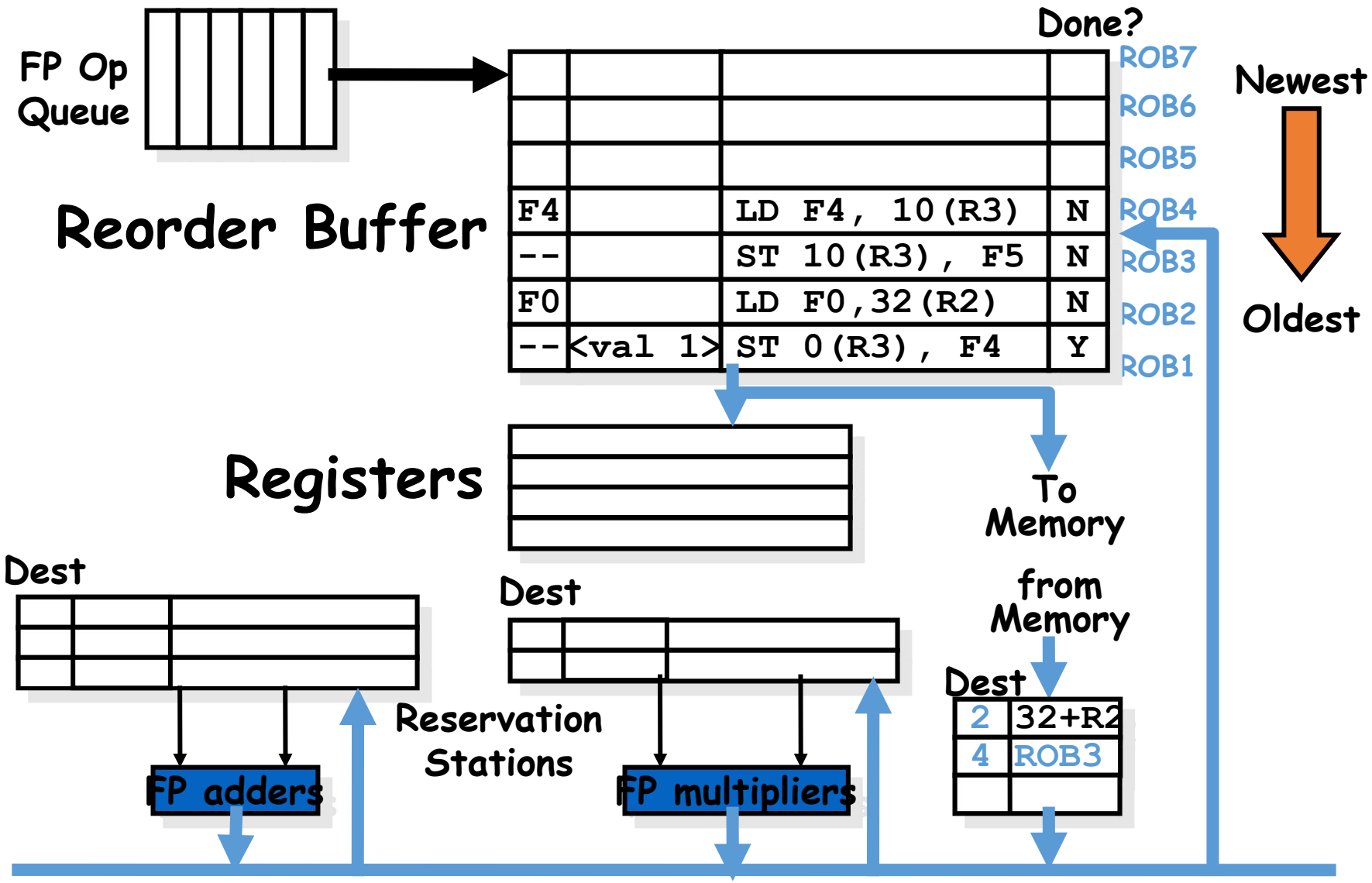
Load/Store Queue: 保存存储器操作的队列,直到可以计算有效地址

Load Queue: 该队列以程序序存储load操作的存储器物理地址。该队列有32项

Store Queue: 存储store操作的存储器物理地址以及要存储的值。该对列32项

Wait Table: 具有1024 项，每项1位的表，由存储器指令虚拟地址索引. 当我们检测到一个load操作超越了与它相关的store操作时，该load对应项置1，取指部件读取该信息，将不会再提前发射出去。该表每隔16384周期复位一次，否则该表可能所有项均为1

Memory Disambiguation:



5.5-1 存储器访问冲突消解

5.5-2 多发射技术

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**
- **是否能够使CPI < 1？ 多发射处理器**
- **两种基本方法: Superscalar 、VLIW**
- **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
- **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（操作的数量）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
    - DSA （Domain Specific Architecture）
  - 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)”

Machine Parallelism

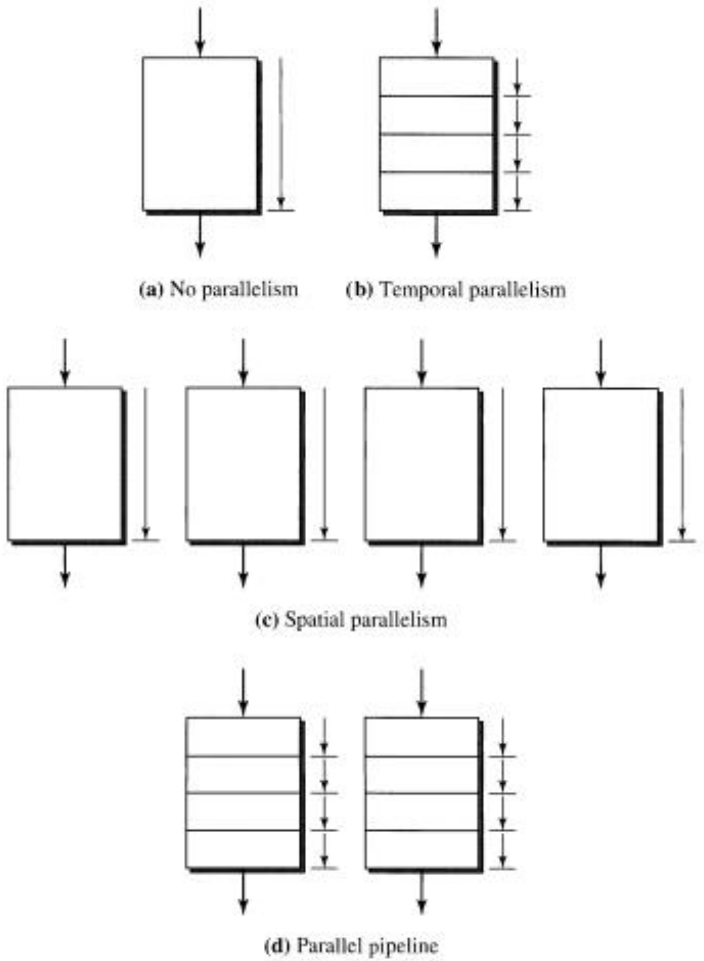


Figure 4.2
Machine Parallelism: (a) No Parallelism (Nonpipelined); (b) Temporal Parallelism (Pipelined); (c) Spatial Parallelism (Multiple Units); (d) Combined Temporal and Spatial Parallelism.

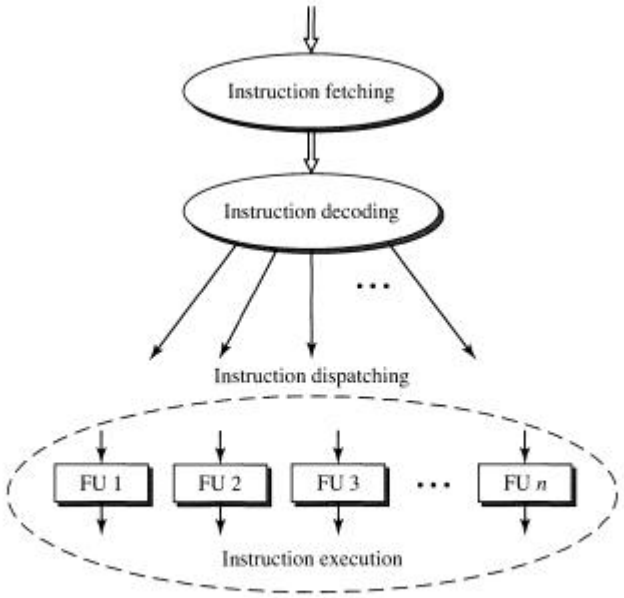


Figure 4.16
The Necessity of Instruction Dispatching in a Superscalar Pipeline.

简单的超标量流水线

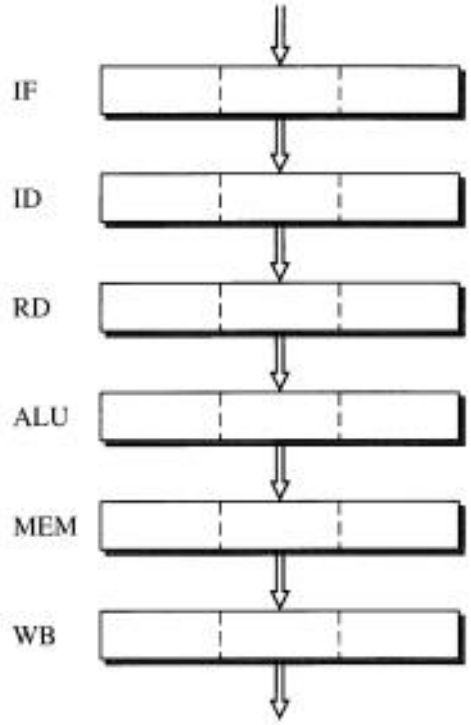


Figure 4.3
A Parallel Pipeline of Width $s = 3$.

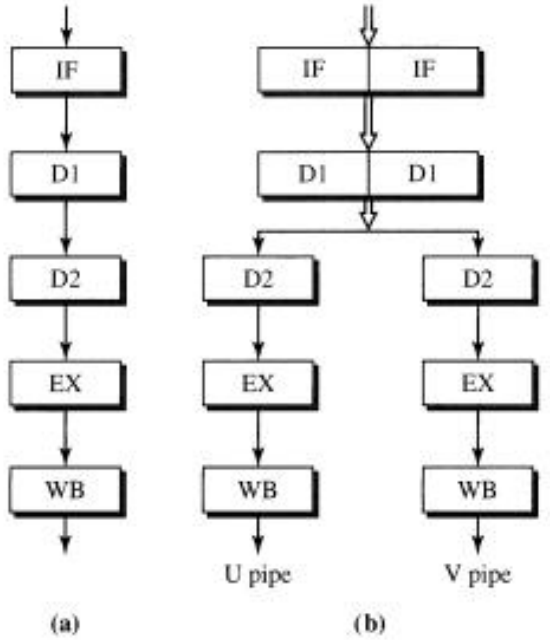


Figure 4.4
(a) The Five-Stage i486 Scalar Pipeline;
(b) The Five-Stage Pentium Parallel Pipeline of Width $s = 2$.

具有多种执行部件（流水化）

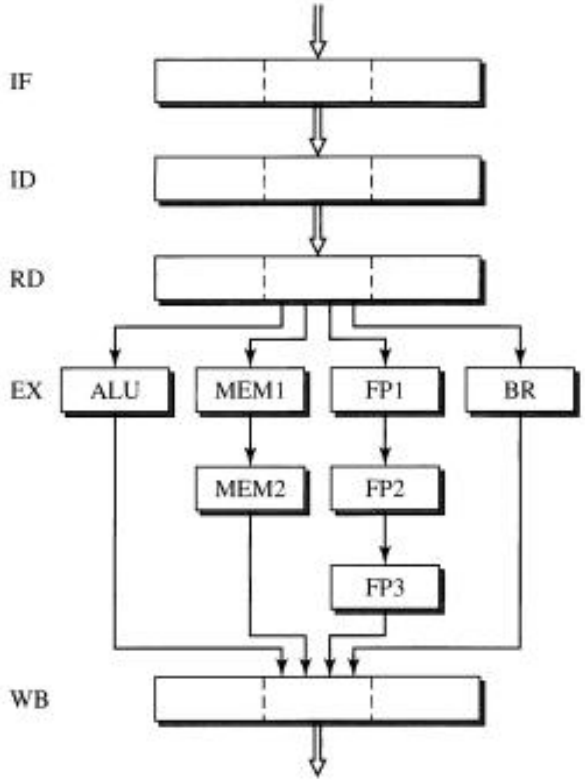


Figure 4.5
A Diversified Parallel Pipeline with Four Execution Pipes.

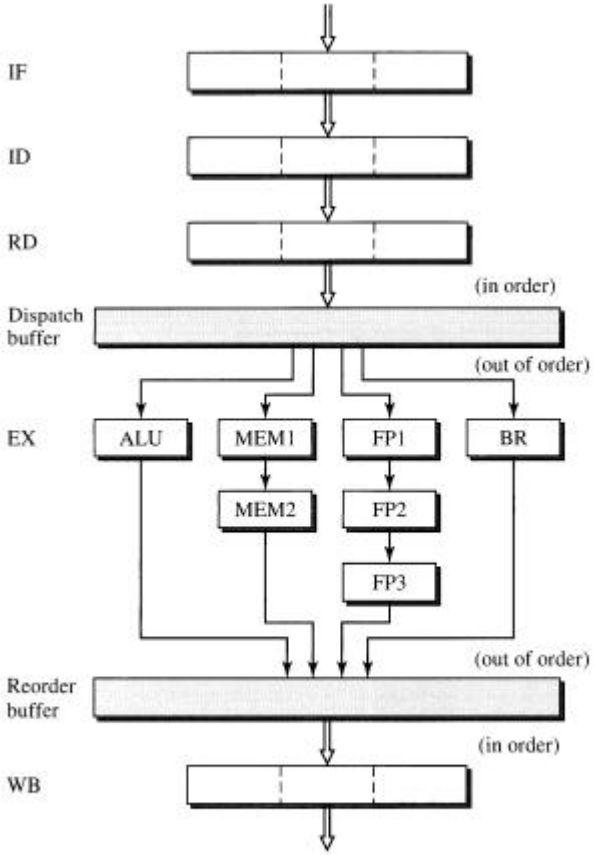


Figure 4.9
A Dynamic Pipeline of Width $s = 3$.

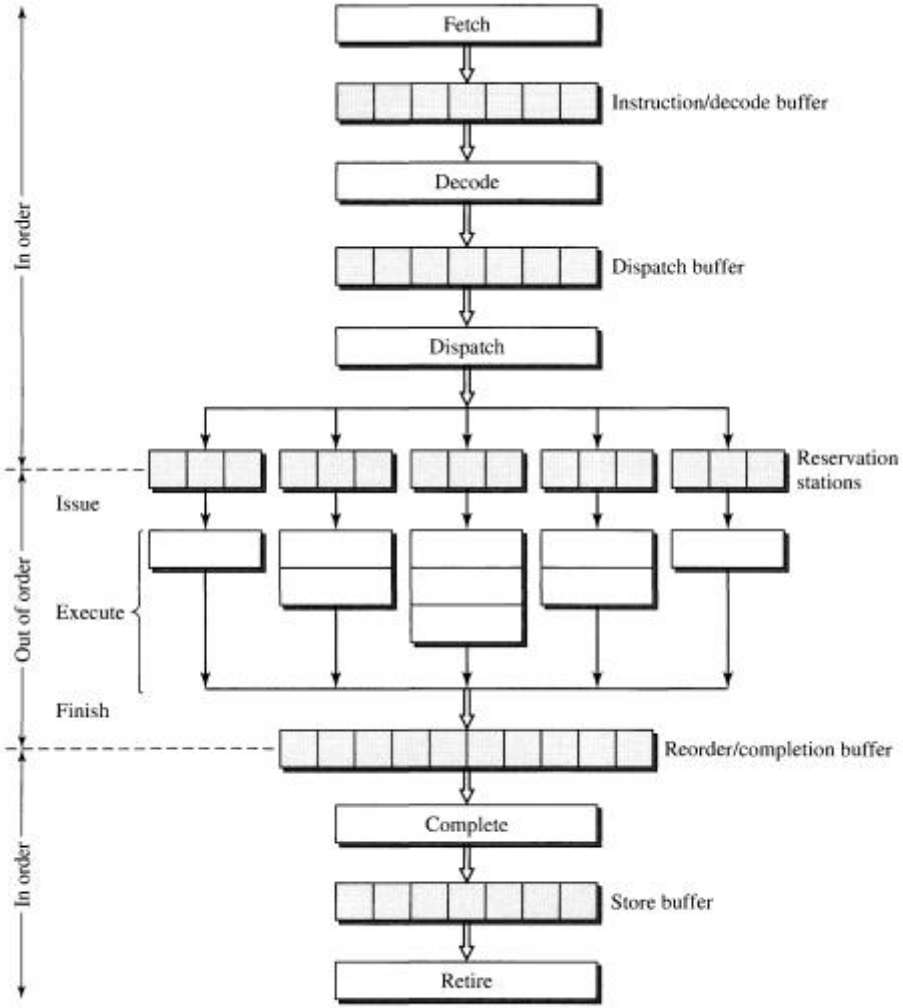


Figure 4.20
A Dynamic Pipeline with Reservation Station and Reorder Buffer.

超标量流水线中的分支预测

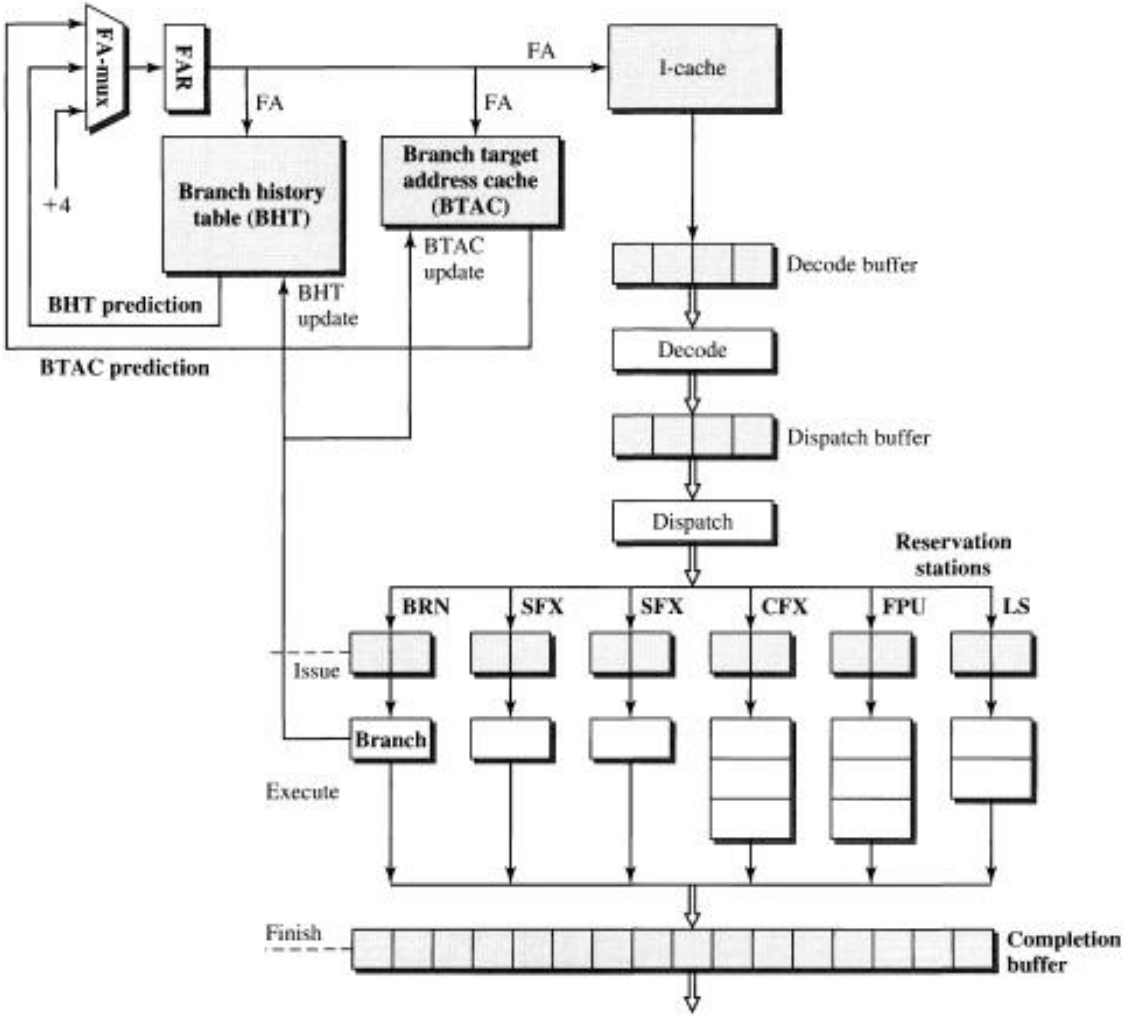
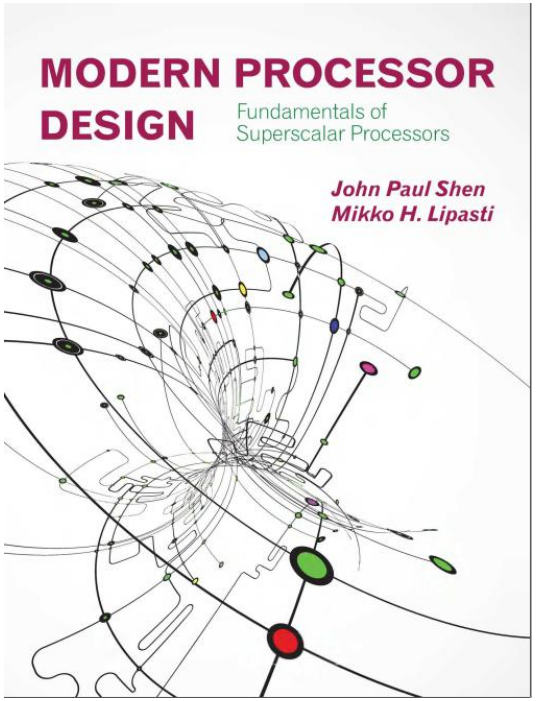


Figure 5.10
Branch Prediction in the PowerPC 604 Superscalar Microprocessor.



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	Mostly in the embedded space: MIPS and ARM, including the Cortex-A53
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Figure 3.19 The five primary approaches in use for multiple-issue processors and the primary characteristics that distinguish them. This chapter has focused on the hardware-intensive techniques, which are all some form of superscalar. Appendix H focuses on compiler-based approaches. The EPIC approach, as embodied in the IA-64 architecture, extends many of the concepts of the early VLIW approaches, providing a blend of static and dynamic approaches.

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

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

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
SUBI R1,R1,#40		9
SD +16(R1),F16		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)

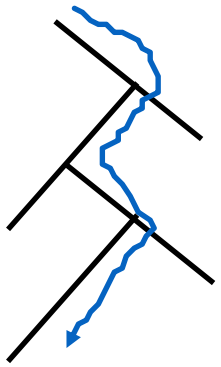
Average: 2.5 ops per clock, 50% efficiency

LD to ADDD: 1 Cycle

ADDD to SD: 2 Cycles

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推断执行硬件成本小**

Superscalar vs. VLIW

- **Superscalar**
  - 代码量较小
  - 二进制兼容性好
- **VLIW**
  - 译码、发射指令的硬件设计简单
  - 需要访问更多的寄存器，一般使用多个寄存器文件

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.

Performance of Dynamic SS

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.

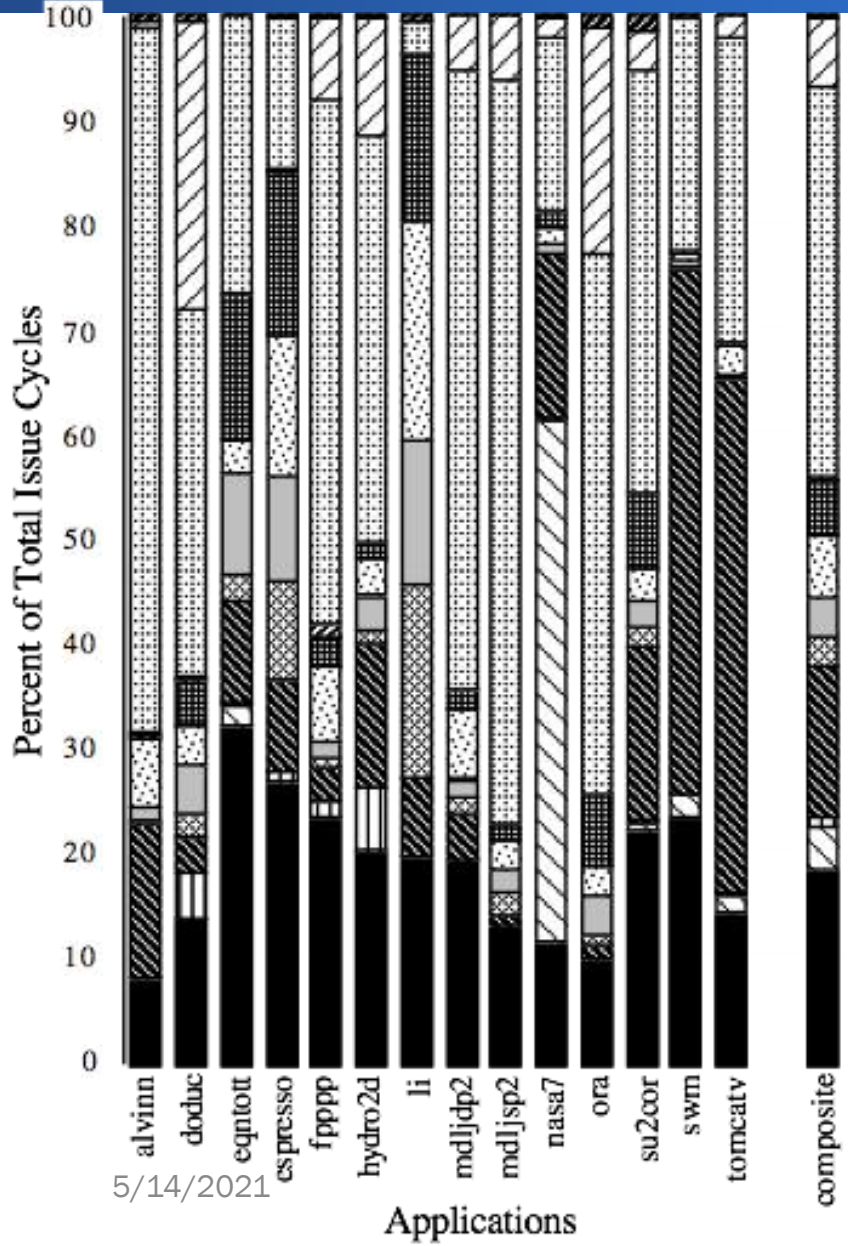
多发射处理器受到的限制（1/2）

- **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

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



For an 8-way superscalar.

Sources of all unused issue cycles in an 8-issue superscalar processor. Processor busy represents the utilized issue slots; all others represent wasted issue slots.

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

Summary #1/3

- **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)
- **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: 在控制相关还没有解决情况下，就开始执行
  - 推断执行利用了ROB撤销指令执行的机制
    - 处理预测错误时，撤销 推测执行的指令
  - 基于BHT的分支预测技术
  - 基于BTB的分支预测技术

Summary #2/3

- **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**
- **不足之处:**
  - Too many value copy operations
    - Register File →RS→ROB→Register File
  - Too many muxes/busses (CDB)
    - Values are from everywhere to everywhere else!
  - Reservation Stations mix values(data) and tags (control)
    - Slow down max clock frequency

Summary #3/3

- **存储器访问的冲突消解**
  - 非投机方式的冲突消解
    - Total Ordering
    - Partial Ordering
      - Load指令前的store指令已经完成了地址计算，有可能乱序执行存储器load操作
    - Load Ordering, Store Ordering
      - Load指令前的存储器访问指令已经完成了地址计算，load队头的load操作有可能在store指令之前执行访存操作。
  - 投机方式的执行
    - Store Ordering
    - 假设Load操作与之前未计算出有效地址的store操作无关。
- **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

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - John Hennessy (Standford)and David Patterson (UCB)
  - Chenxi Zhang (Tongji)
  - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**