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

- **存储器访问的冲突消解**
  - Total Ordering
  - Partial Ordering
  - Load Ordering, Store Ordering
  - Store Ordering

- **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

Review: 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**

Review：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 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.

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(4学时)

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

5.5-1 存储器访问冲突消解

5.5-2 多发射技术

5.6 多线程技术

5.6 多线程技术

多发射处理器局限性

多线程处理器基本思想

多线程处理器分类

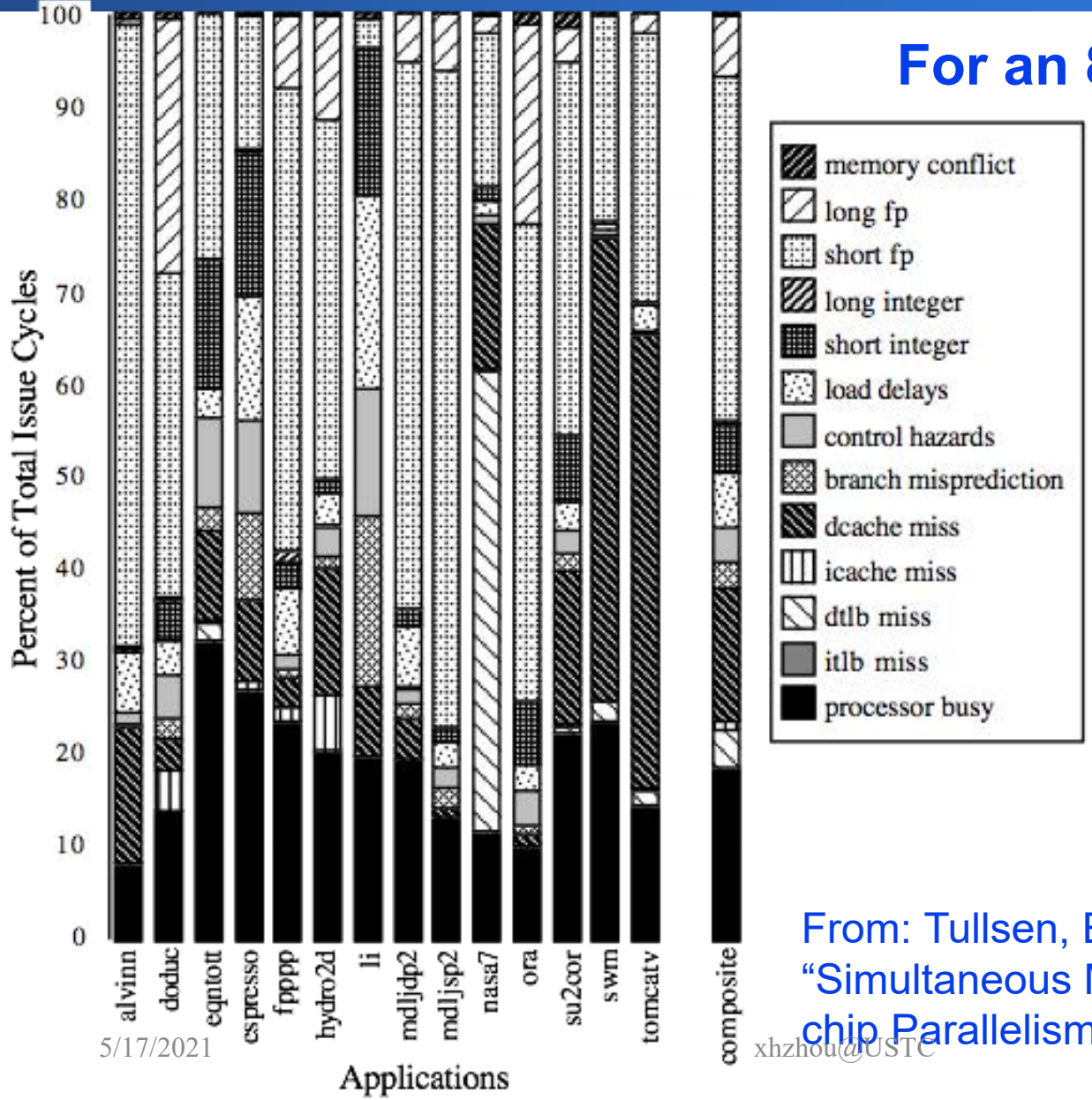
多发射处理器受到的限制（1/2）

- **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.

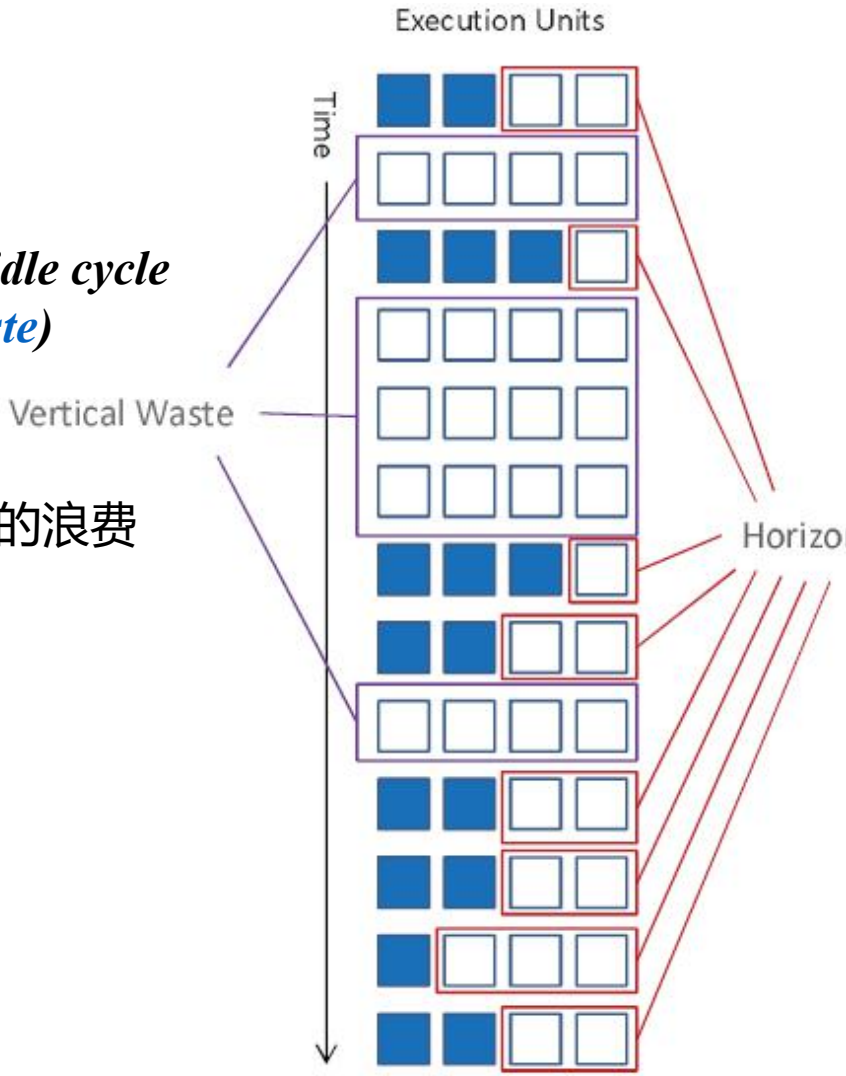
Sources of all unused issue cycles in an 8-issue superscalar processor. Processor busy represents the utilized issue slots; all others represent wasted issue slots.

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

Superscalar Machine Efficiency

Completely idle cycle (vertical waste)

时间维度上的浪费



Partially filled cycle, i.e., IPC < 4 (horizontal waste)

空间维度上的浪费

5.6 多线程技术

多发射处理器局限性

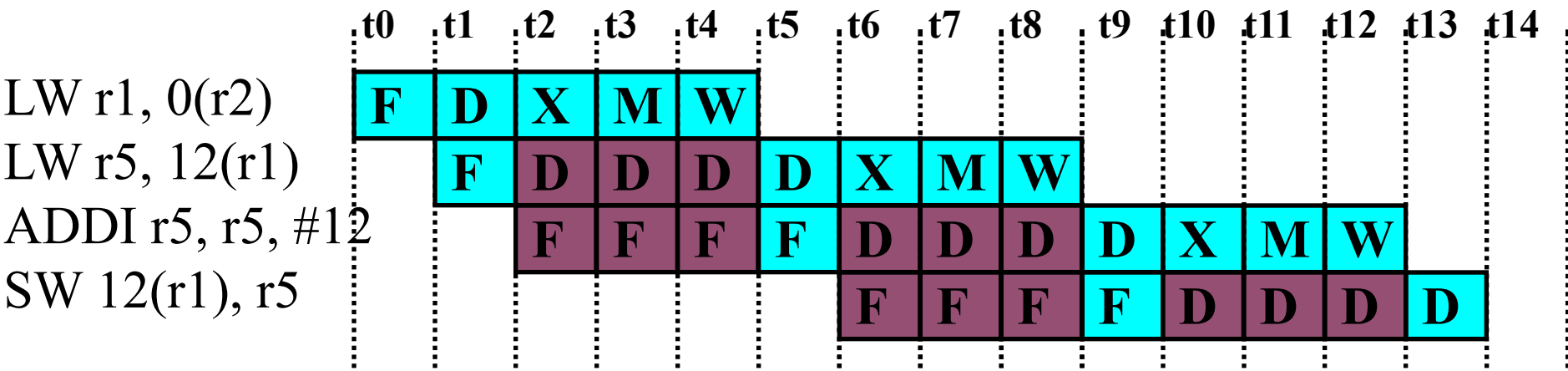
多线程处理器基本思想

多线程处理器分类

Multithreading

- **背景：** 从单线程程序挖掘指令集并行越来越困难
- **前提：** 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
- **基本思想：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



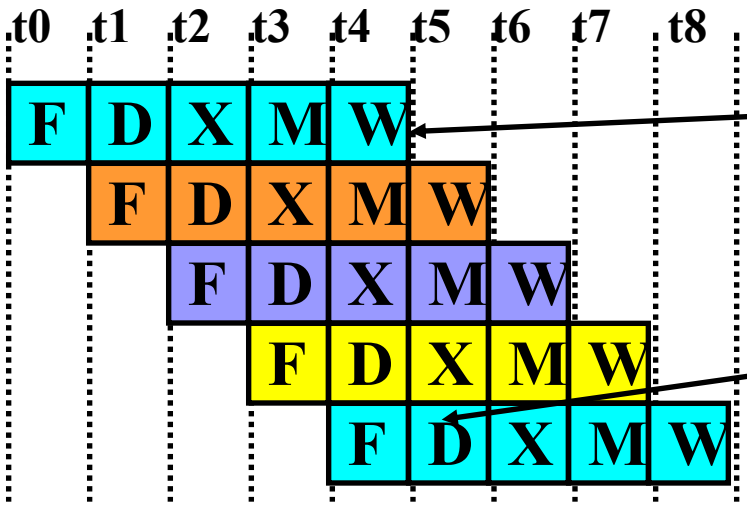
- 每条指令与前一条指令存在RAW相关
- 如何处理相关?
  - 使用interlock机制(slow)
  - 或定向路径

Multithreading

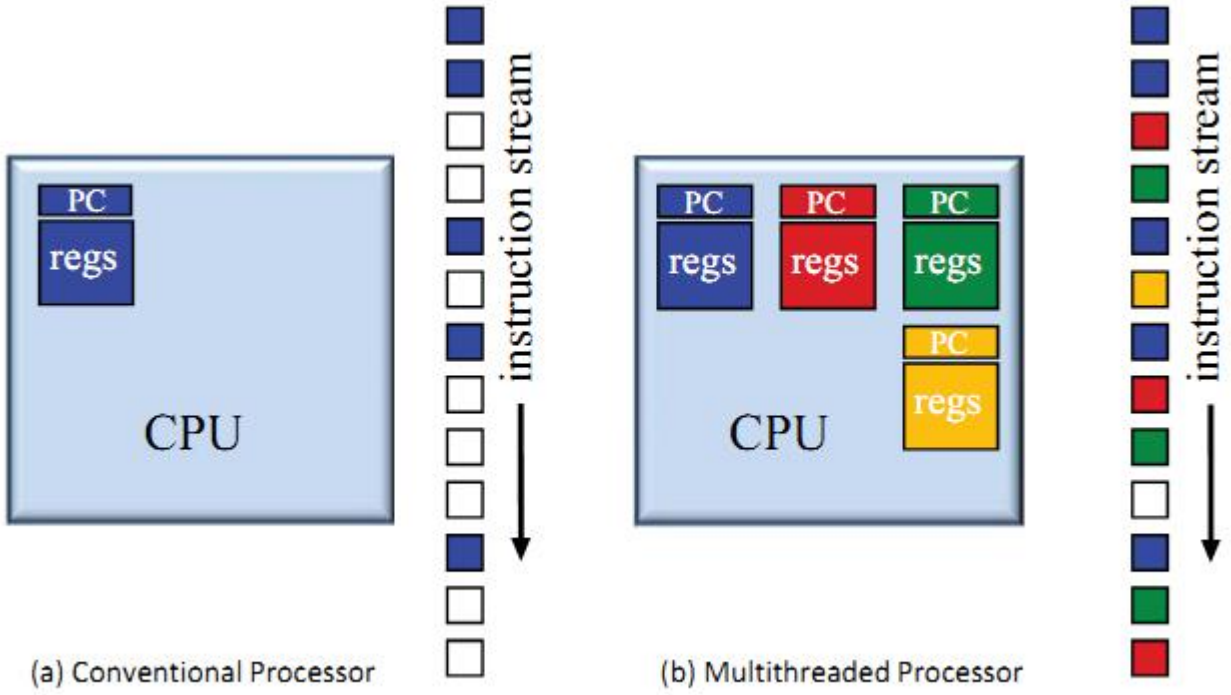
- **如何保证流水线中指令间无数据依赖关系?**
- **一种办法：在相同的流水线中交叉执行来自不同线程的指令**

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)



Prior instruction in a thread always completes write-back before next instruction in same thread reads register file



A conventional processor compared with a multithreaded processor.

CDC 6600 Peripheral Processors (Cray, 1964)

- **容忍延时：**
  - 当一个线程遇到长延时操作时，处理器可以从另一个线程执行有用的操作
- **CDC 6600 I/O处理器**
  - 第一个支持多线程模式的硬件系统
  - 10个I/O线程交替执行
  - 10 个“virtual” I/O 处理器
  - 流水线控制采用固定交叉方式
  - 流水线每段100ns
  - 每个虚拟处理器每隔1000ns执行一条指令
  - 累加器型ISA有利于保存处理器状态

CDC 6600 Architecture

存储器(storage) 包括:

- **PPU Private Storage**
  - 字长： 12bits
  - 容量：4096字
- **Central Storage**
  - 字长：60bits
  - 地址格式：字地址（12位）+ bank地址（5位）
  - 容量：128Kwords, 32 banks(60bits)
- **Extended Core Storage**
  - bank字长：480bits
  - Bank数：最多16
  - bank容量：125000 “central”字（60bits)
  - 并行传输宽度：60bits
  - 吞吐率：10个“Central”字 每1000ns
  - 读写周期：3.2 microseconds

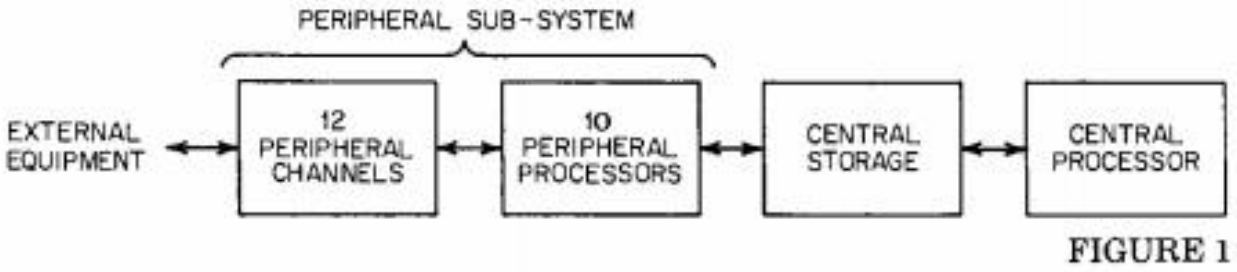


FIGURE 1

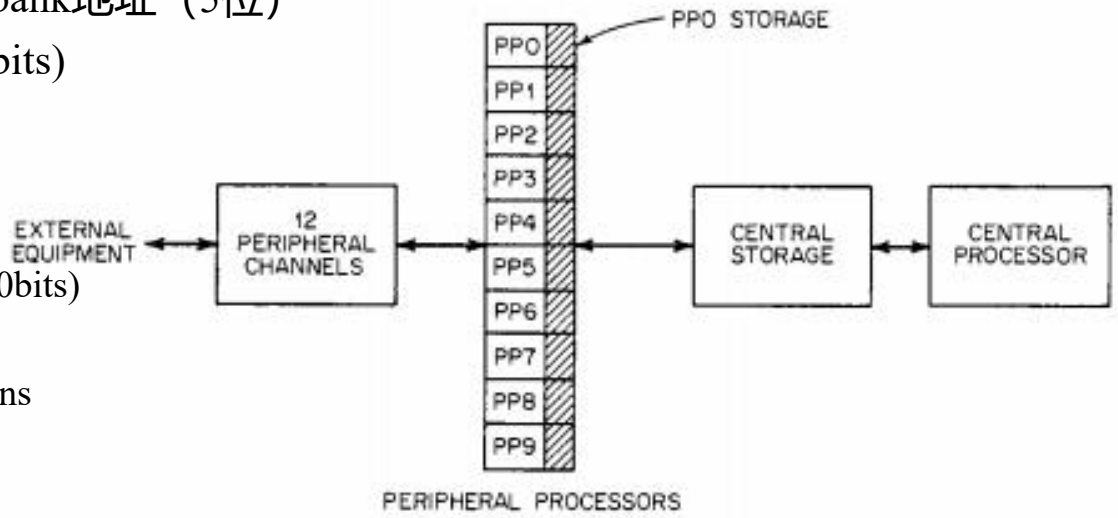


FIGURE 2

PPU：Peripheral Processor Unit

J. E. Thornton，Design of a Computer_ The Control Data 6600 (1970, Scott, Foresman and Company)

两种时钟周期

MAJOR CYCLE：1000 ns； 与PPU Storage的存储器读写周期一致

MINOR CYCLE： 100ns： ALU运算, 以及传输一个字的时间 （字长12bit）

例如：Load (d)； A <- mem[d]

Add X；

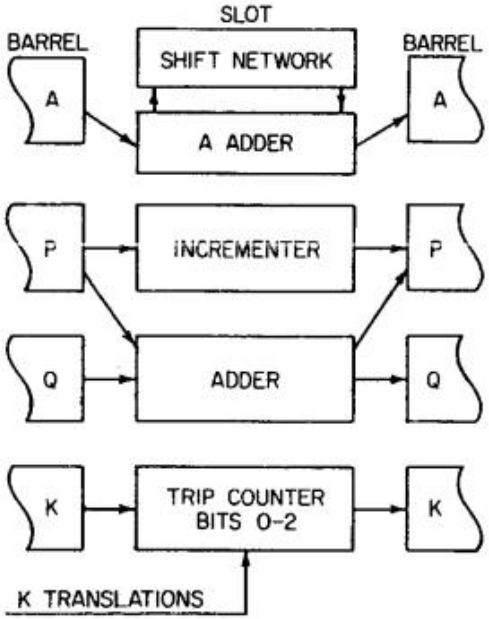


FIGURE 88 Elements of the slot.

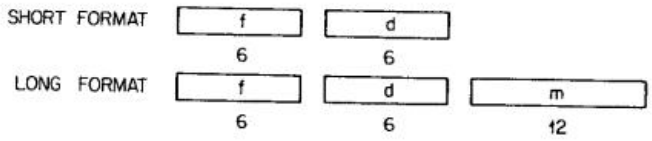


FIGURE 84

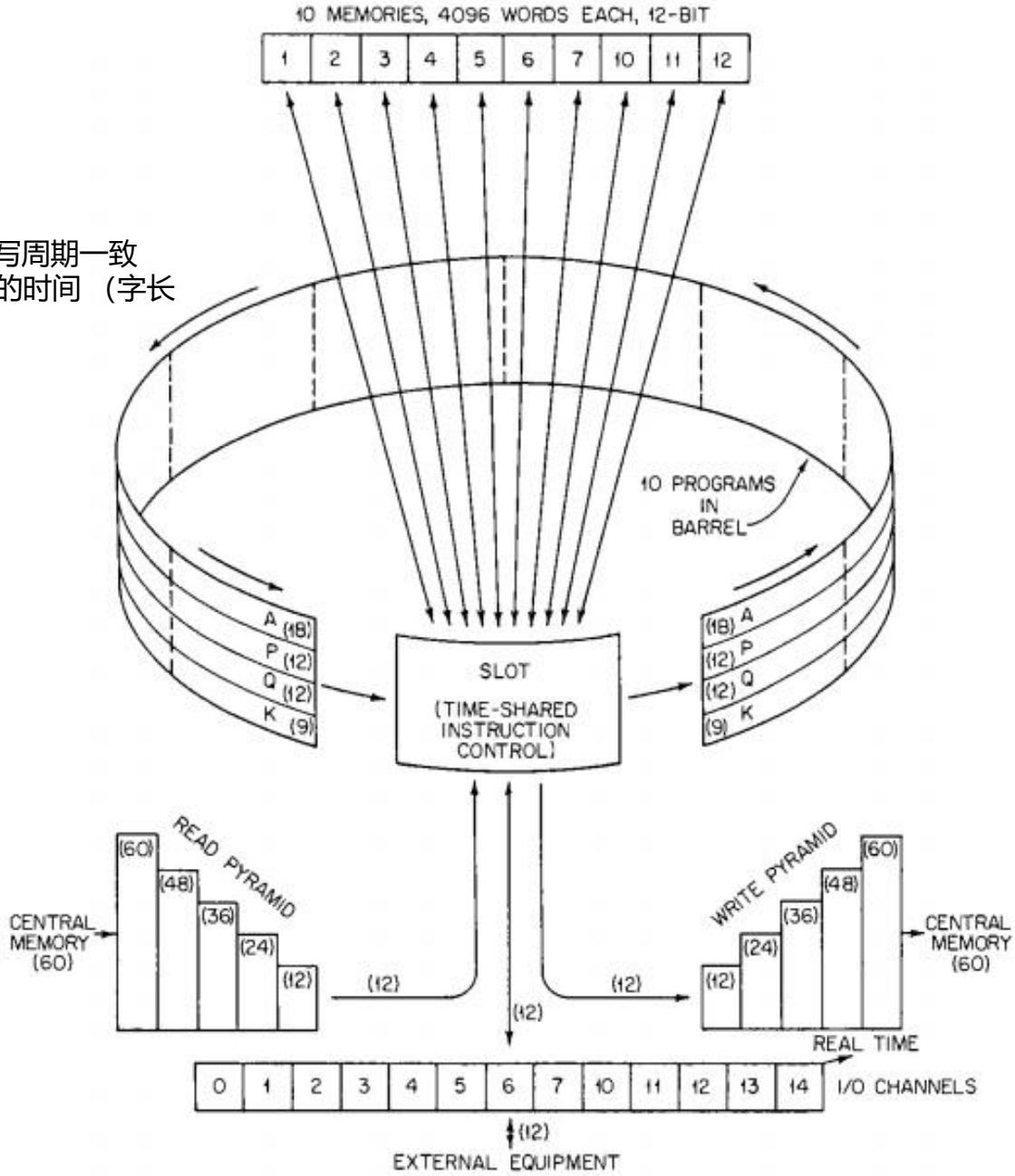
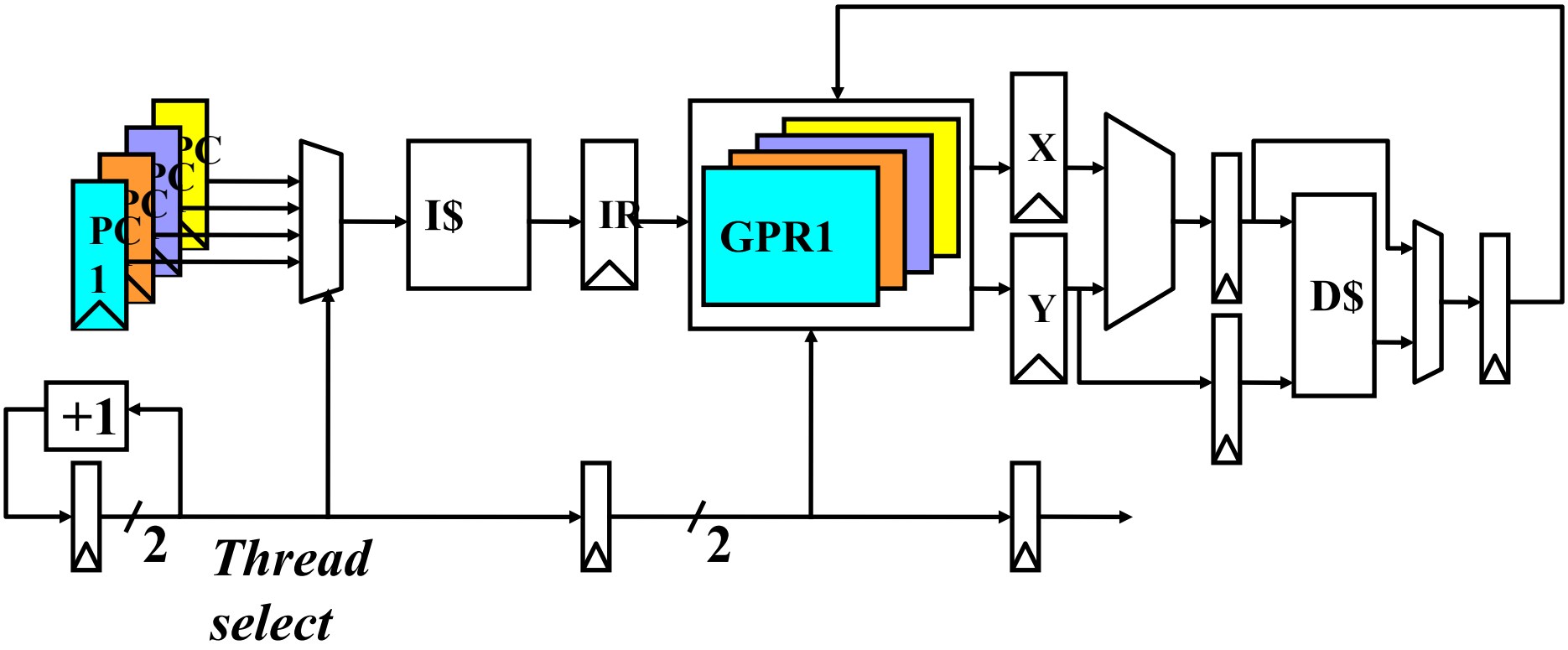


FIGURE 83 Peripheral processors.

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（针对每个线程，CPU似乎运行的慢一些)

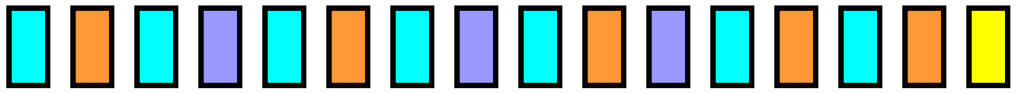


Multithreading Costs

- **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



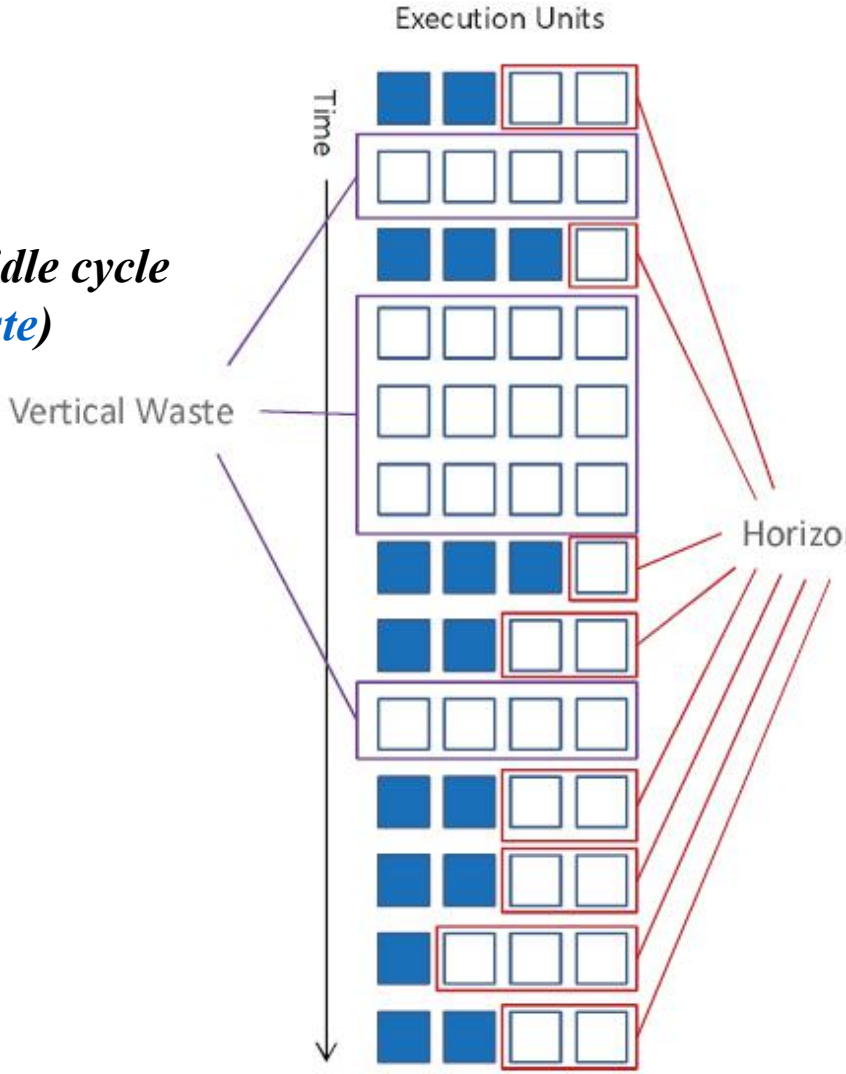
- **硬件控制的线程调度 (HEP, 1982)**
  - 硬件跟踪哪些线程处于ready状态
  - 根据优先级方案选择线程执行

5.6 多线程技术

多发射处理器局限性 → 多线程处理器基本思想 → 多线程处理器分类

1. Chip Multiprocessing
2. Coarse-Grain Multithreading
3. Fine-Grain Multithreading
4. Simultaneous Multithreading

Superscalar Machine Efficiency



Completely idle cycle (vertical waste)

$$IPC = \frac{1}{CPI}$$

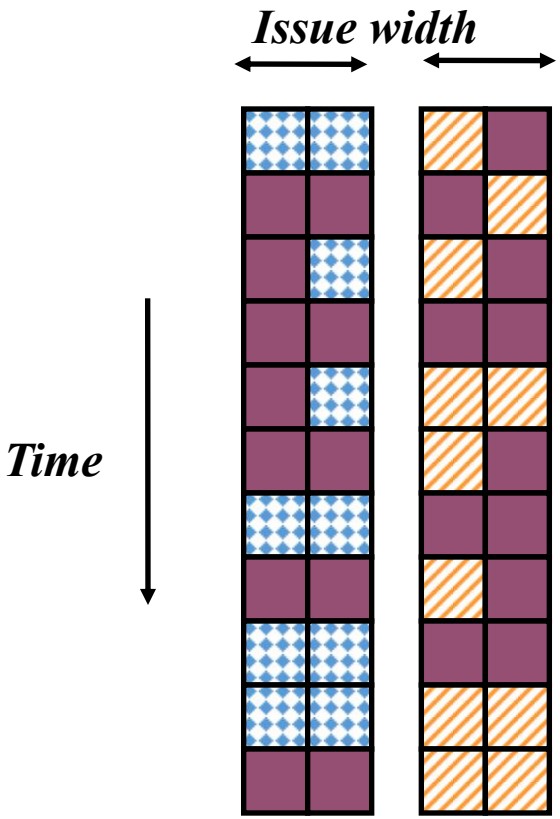
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

问题：如何充分利用资源，提高并行度？

-- multithreading

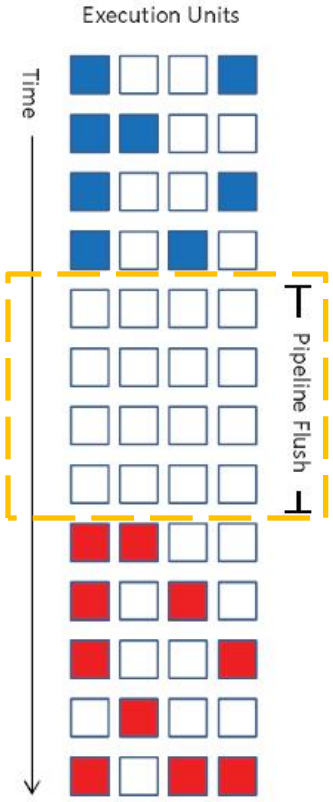
Chip Multiprocessing (CMP)

- **分成多个处理器后的效果?**
  - 同一时钟周期可以运行不同线程的指令
  - 单个处理器核同一时钟周期无法执行不同线程的指令
  - 从单个核看，没有减少水平浪费和垂直浪费。
  - 由于发射宽度在核间进行了静态分配，导致水平和垂直方向浪费减少
- **例如：2核2发射的CMP**
  - 发射宽度为4，即同时可以发射4条指令
  - 当1个线程stall，那么垂直浪费最多为2条指令



Vertical(Coarse-Grain) Multithreading

- **粗粒度多线程方式**
  - 当线程运行时存在较长时间延时时切换到另一线程
    - 例如：Cache失效时
    - 等待同步结束
  - 同一线程指令间的较短延时不切换
- **如果基于粗粒度的时钟周期交叉运行模式，结果怎样?**
  - 减少了垂直方向的浪费，但仍然存在垂直方向的浪费
  - 减少水平方向的浪费?



A *Coarse-Grain Multithreaded* architecture has the ability to switch between threads with only a short hardware context switch latency < 10 cycles

线程间切换速度快！<10 cycles

Coarse-Grain Multithreading

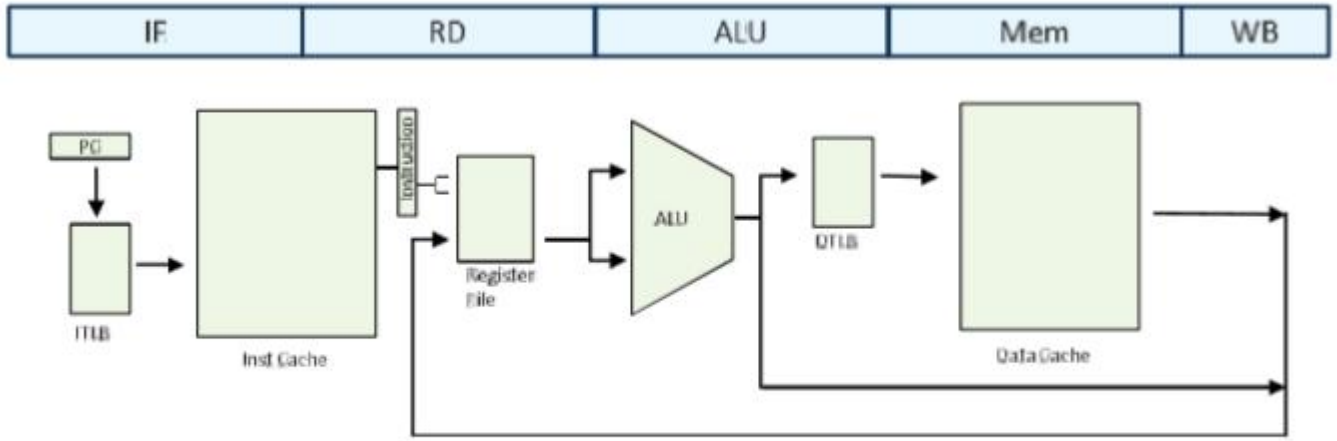


Figure 3.1: The MIPS R3000 Pipeline. This pipeline does not support multithreading.

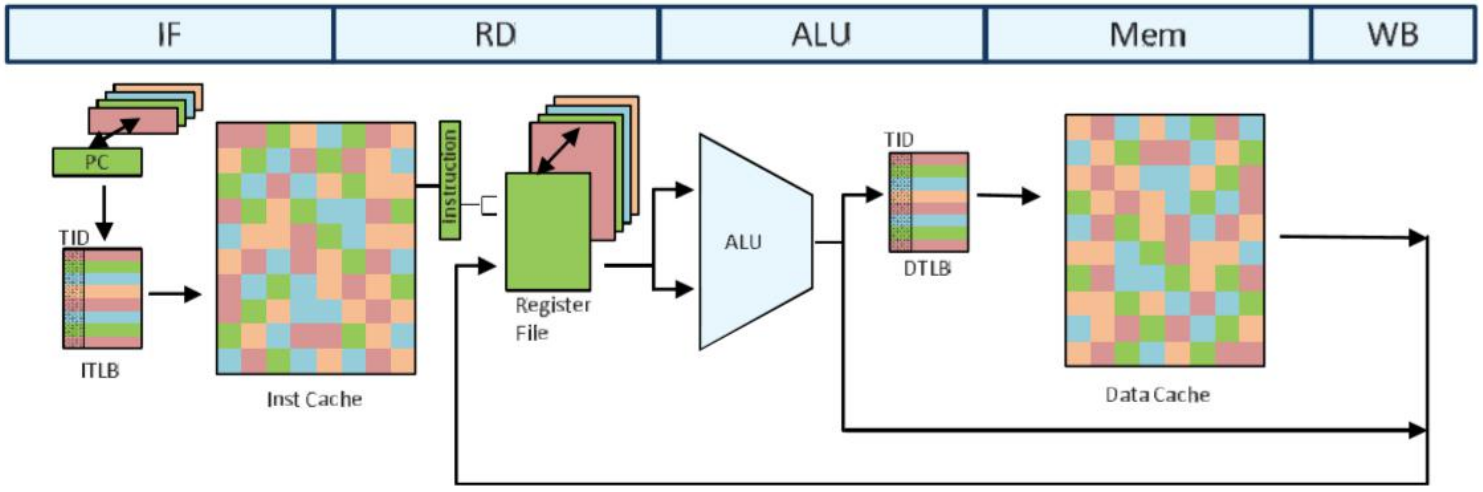


Figure 3.2: The MIPS R3000 Pipeline with support for Coarse-Grain Multithreading.

CGMT Context swap

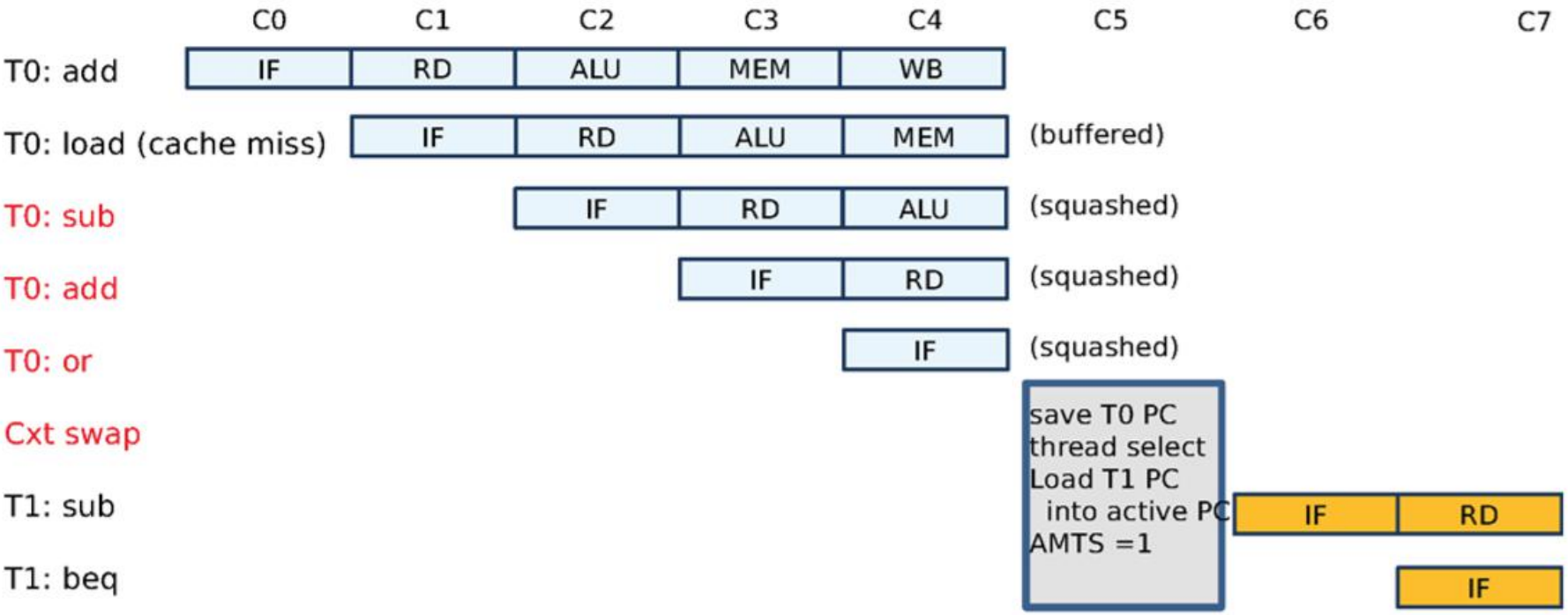
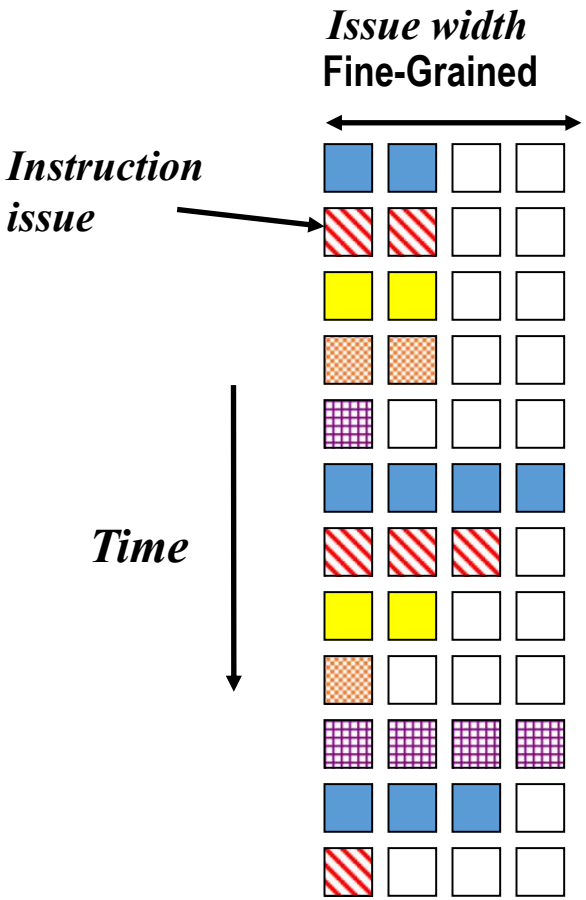


Figure 3.3: The operation of a CGMT context swap in our reference design.

Fine-Grain Multithreading



- **细粒度多线程**
  - 多个线程的指令交叉执行
- **如果基于细粒度的时钟周期交叉运行模式，结果怎样?**
  - 减少垂直方向的浪费
    - 当线程数足够多时，可消除垂直方向的浪费
  - 仍然存在水平方向的浪费

Fine-Grain Multithreading

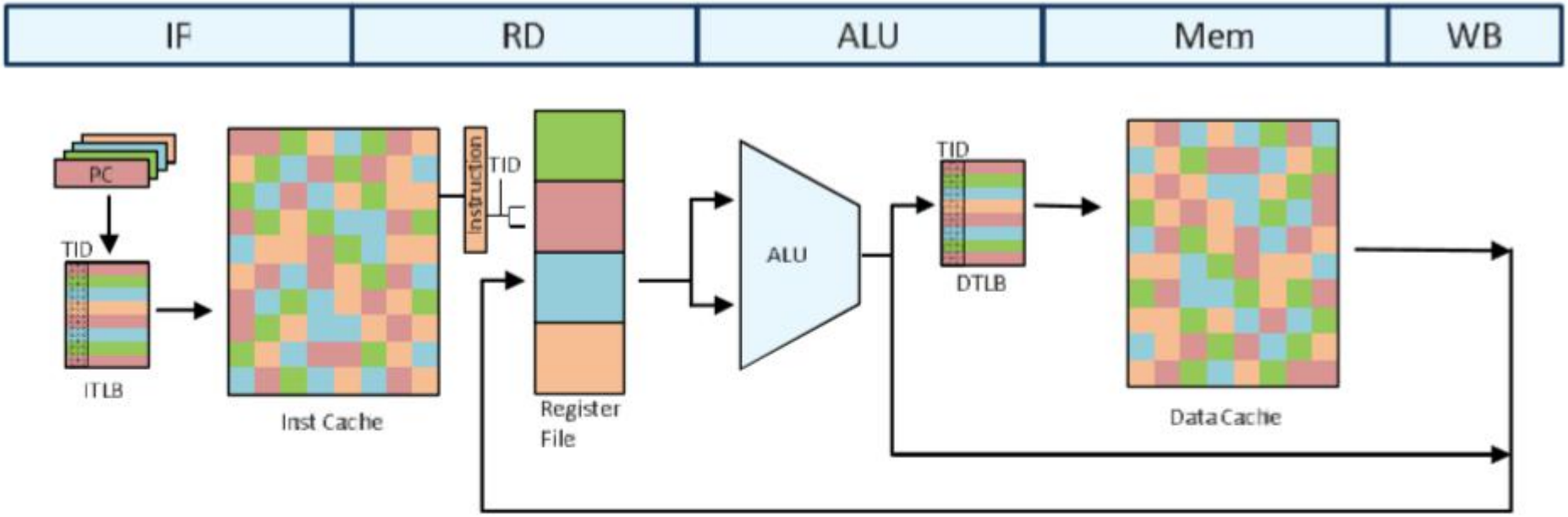
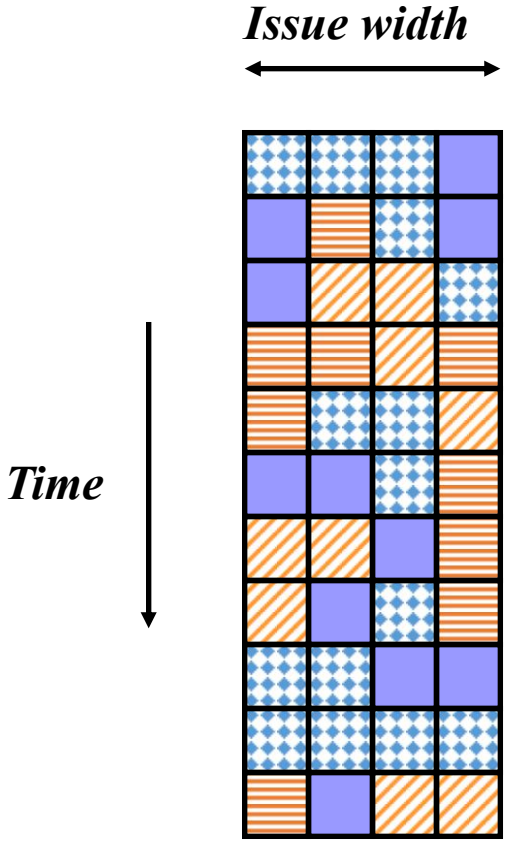


Figure 4.1: The MIPS R3000 Pipeline with support for Fine-Grain Multithreading. Although our target design would support 8 threads to maximize throughput, we show a 4-thread implementation for simplicity.

要点：硬件实现更快的上下文切换

Ideal Superscalar Multithreading [Tullsen, Eggers, Levy, UW, 1995]

- **采用多线程交叉模式使用多个issue slots**
- **Simultaneous Multithreading (SMT)**



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**
  - Alpha AXP 21464
  - Intel Pentium 4， Intel Nehalem i7
  - IBM Power5

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（Out-of-Order) 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使用整个系统资源

仅支持单线程的MIPS R1000

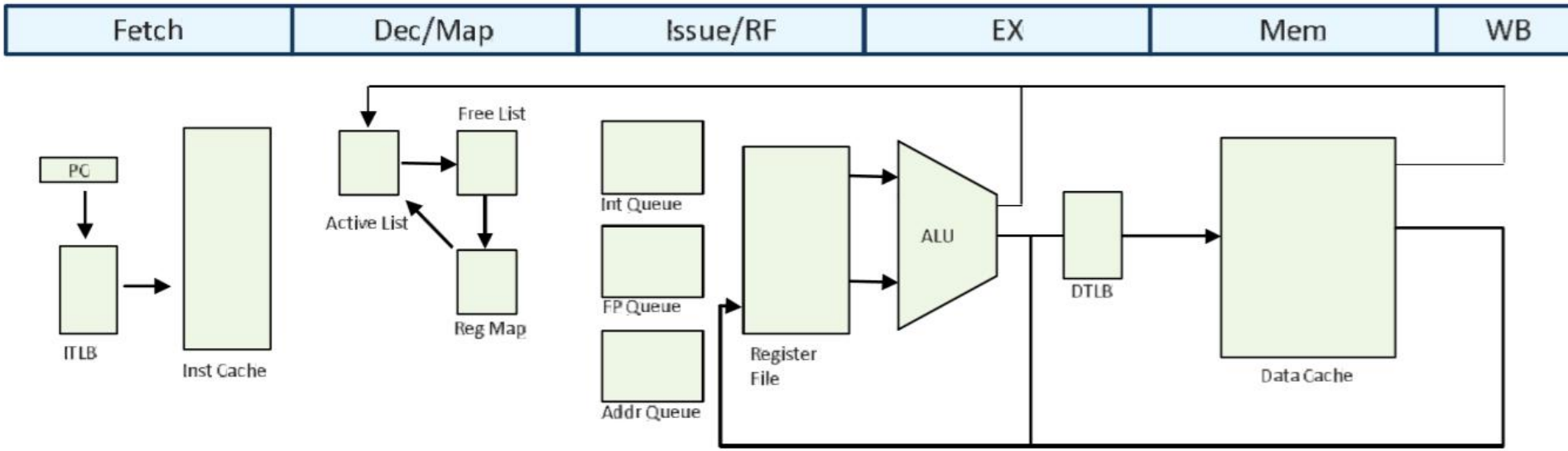


Figure 5.1: The pipeline of the MIPS R10000 processor, as seen by a load instruction.

支持同时多线程的MIPS R10000

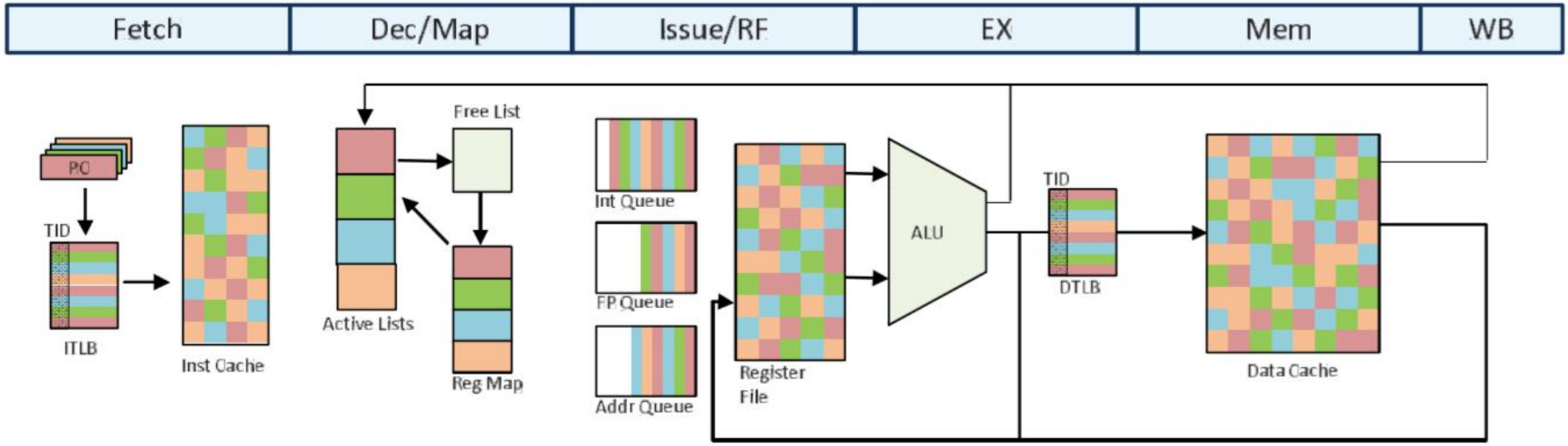
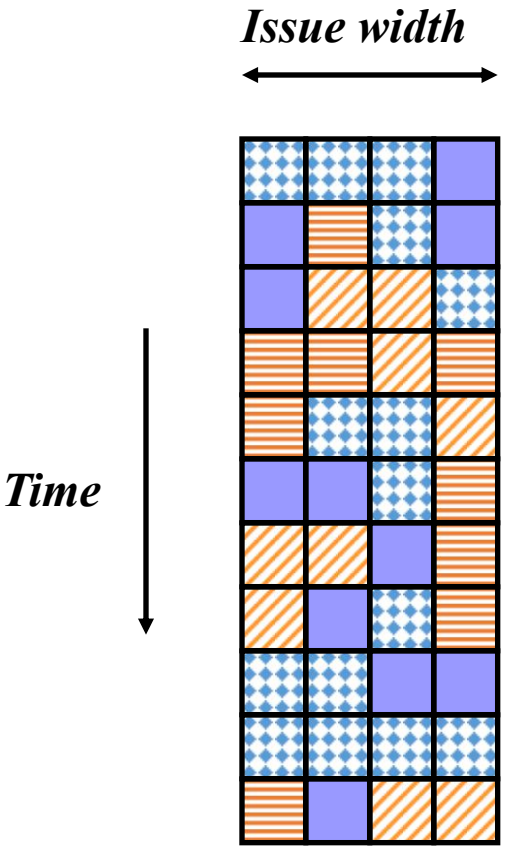


Figure 5.2: The MIPS R10000 pipeline with support added for simultaneous multithreading.

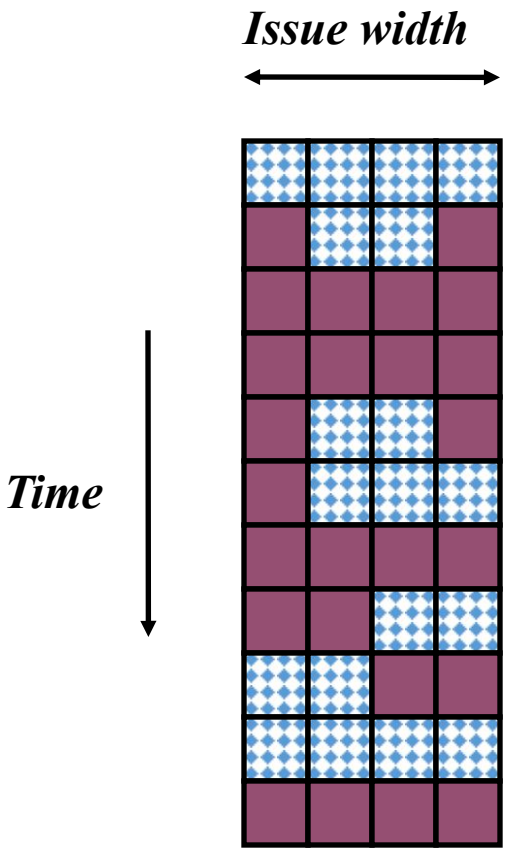
SMT adaptation to parallelism type

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



同时有4个线程并行执行

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



同时有2个线程并行执行

Performance

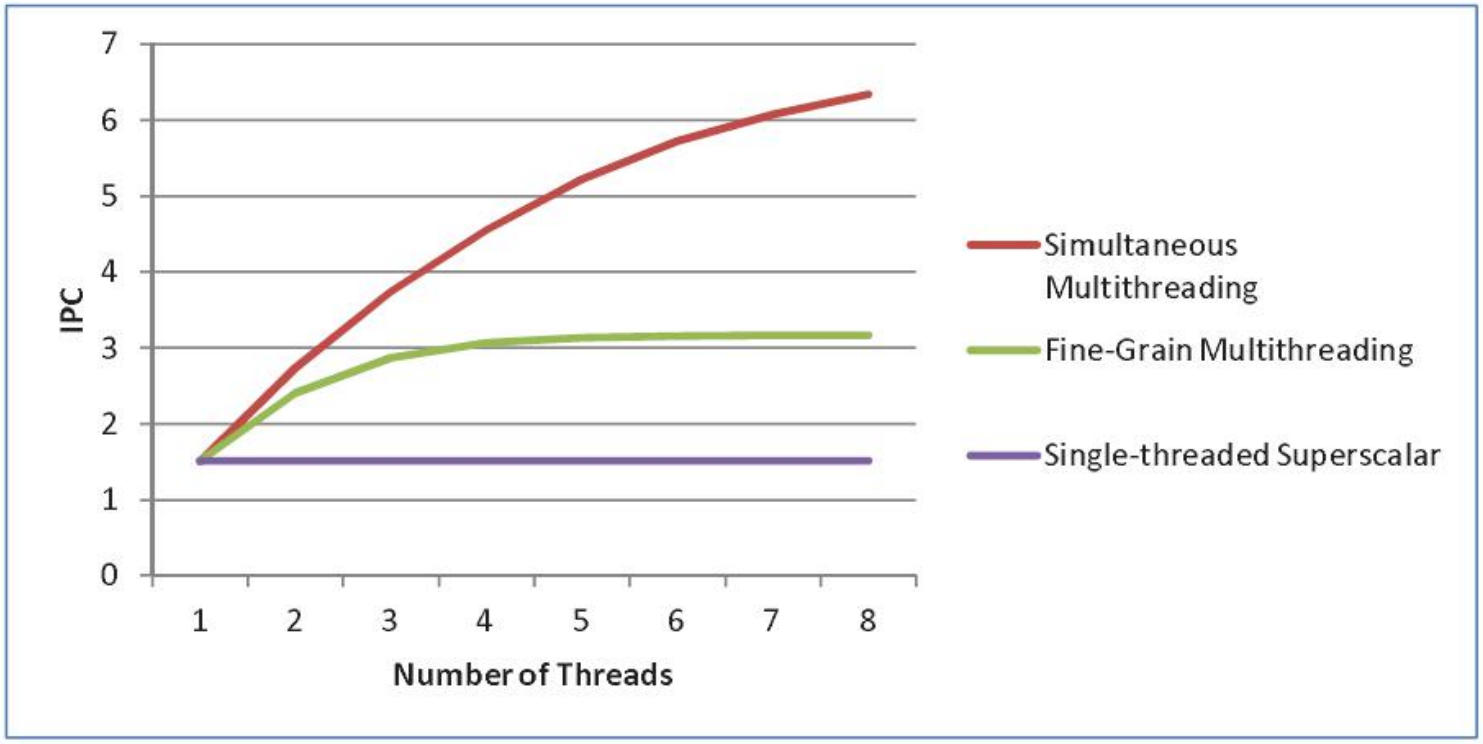
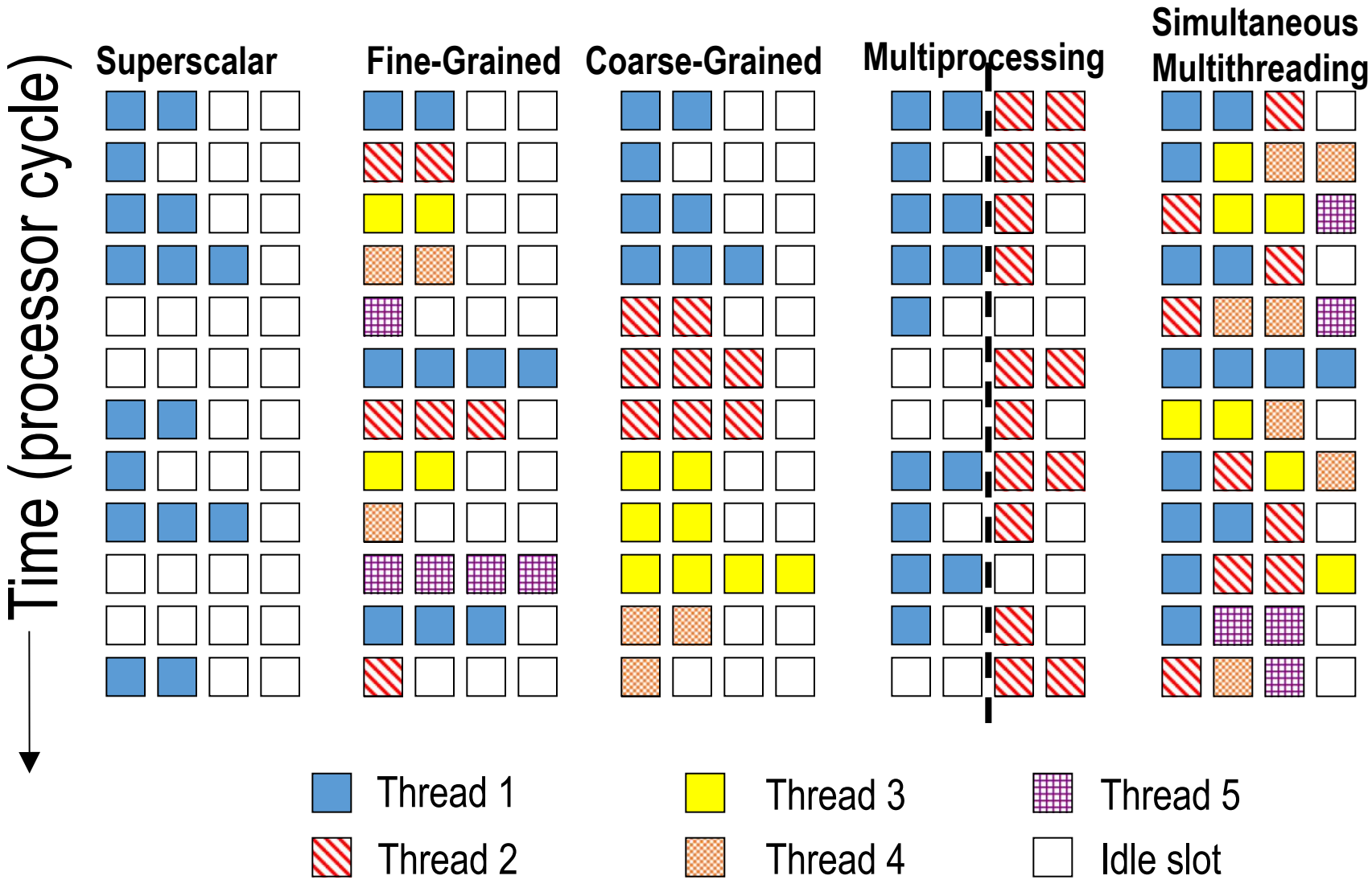


Figure 2.7: The performance of fine-grain and simultaneous multithreading models on a wide superscalar processor, as the number of threads increases. Reprinted from Tullsen et al. [1995].

Summary: Multithreaded Categories

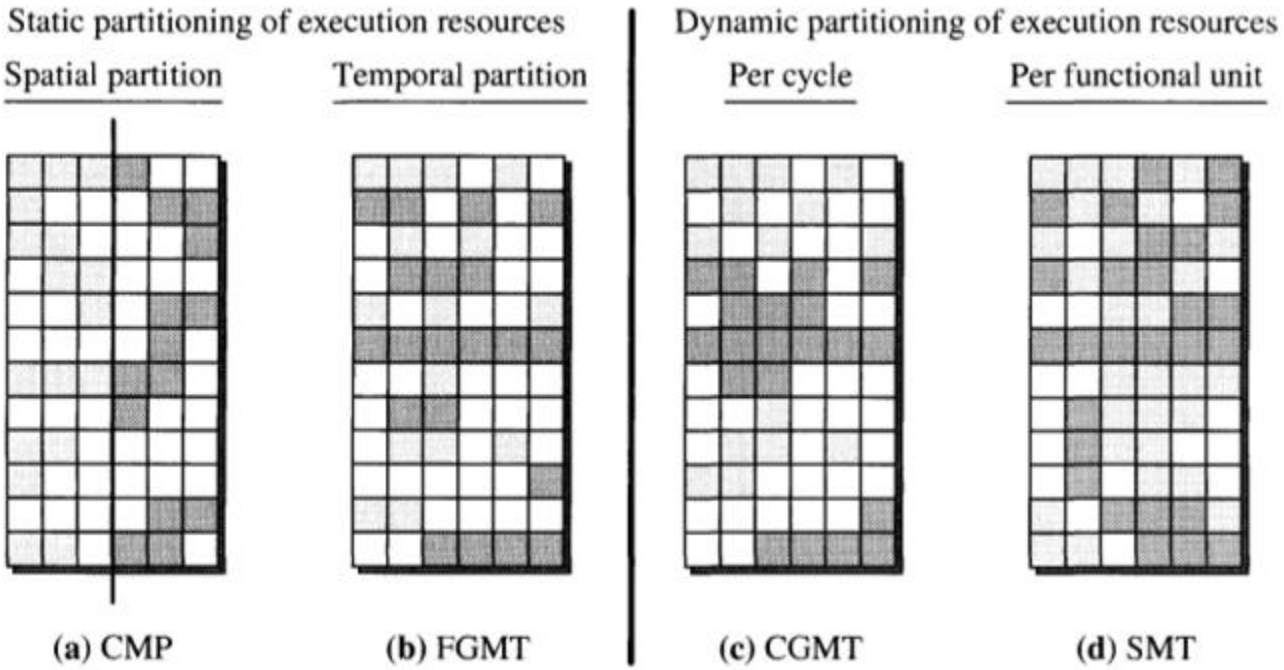


各种多线程技术资源使用情况（1/2)

多线程技术	线程间共享的资源	上下文切换机制
None	共享：所有资源	操作系统负责切换
FGMT	独占：寄存器文件，控制逻辑/状态	每个Cycle切换
CGMT	独占：I-fetch buffer, 寄存器文件，控制逻辑/状态	流水线较长stall时切换
SMT	独占：I-fetch buffer，return address stack, 寄存器文件，控制逻辑/状态，reorder buffer，store queue等	所有上下文处于活跃状态，没有切换问题
CMP	共享：二级Cache，系统互联	所有上下文处于活跃状态，没有切换问题

John Paul Shen，Mikko H. Lipasti；Modern Processor Design：Fundamentals of Superscalar Processors； 2013, Waveland Press

各种多线程技术资源使用情况（2/2)



(a) CMP	空间维度划分发射宽度；静态划分执行所需资源
(b) FGMT	时间维度划分发射宽度；静态划分执行所需资源
(c) CGMT	长延时时切换线程；每个cycle动态划分执行所需的资源
(d) SMT	每个cycle发射的指令可来自不同线程，动态划分执行所需的资源

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**