《计算机系统结构》课程直播

2020. 4.30

听不到声音请及时调试声音设备，可以下课后补签到

请将ZOOM名称改为“姓名”；

本节内容

- ❑ ILP（指令级并行性）的局限性
- ❑ 并发多线程处理器
  - ● 用ILP开发TLP（线程级并行性）

From : H&P Computer Architecture: A Quantitative Approach,
Fifth Edition, （5th edition)

ILP受限

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)

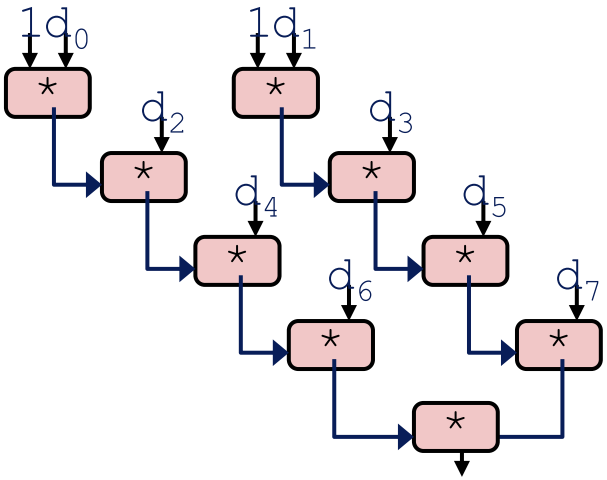
- 由Superscalar或VLIW的实现带来的特殊问题
  - Superscalar的译码、发射问题
    - 到底能发射多少条指令?
  - VLIW 代码量问题：循环展开 + VLIW中无用的区域
  - VLIW 互锁 => 1 个相关导致所有指令停顿
  - VLIW 的二进制兼容问题

回顾：一个例子：循环展开、 独立计算(2x2)

```
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    long length = vec_length(v);
    long limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x0 = IDENT;
    data_t x1 = IDENT;
    long i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
       x0 = x0 OP d[i];
       x1 = x1 OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
     x0 = x0 OP d[i];
    }
    *dest = x0 OP x1;
}
```

循环展开、 独立计算

```
x0 = x0 OP d[i];
x1 = x1 OP d[i+1];
```



改变之处:

- 两个独立的计算“流”

性能分析:

N 个元素, D cycles latency/op

(N/2+1)*D cycles, **CPE = D/2**

循环展开、 独立计算: Double *

- Case
  - Intel Haswell
  - Double FP Multiplication
  - Latency bound: 5.00. Throughput bound: 0.50

Accumulators

FP *	Unrolling Factor L							
K	1	2	3	4	6	8	10	12
1	5.01	5.01	5.01	5.01	5.01	5.01	5.01	
2		2.51		2.51		2.51		
3			1.67					
4				1.25		1.26		
6					0.84			0.88
8						0.63		
10							0.51	
12								0.52

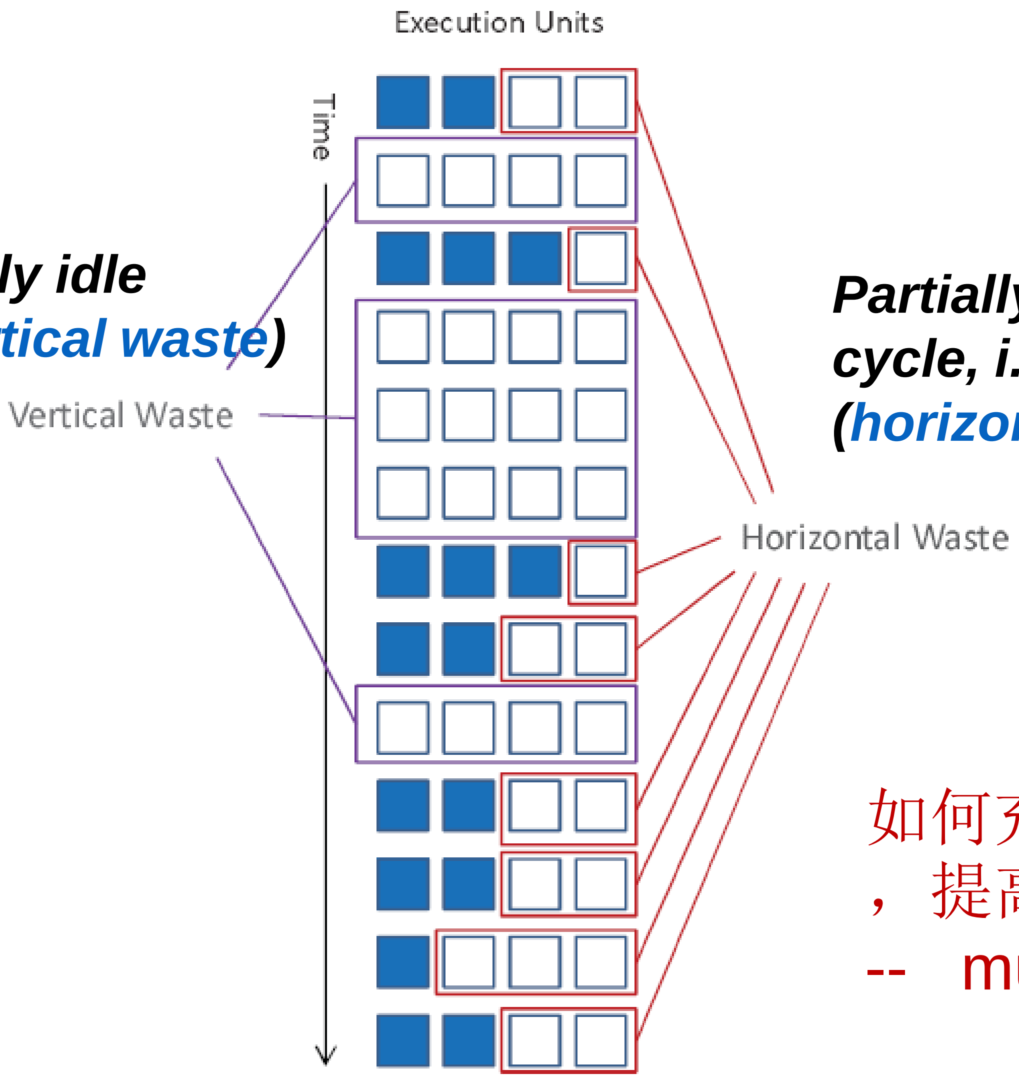
获得的性能

Method	Integer		Double FP	
Operation	Add	Mult	Add	Mult
Best	0.54	1.01	1.01	0.52
Latency Bound	1.00	3.00	3.00	5.00
Throughput Bound	0.50	1.00	1.00	0.50

- 受限于功能单元的吞吐率
- 相比未优化的基准程序快了42倍

Superscalar Machine Efficiency

Completely idle cycle (vertical waste)



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

如何充分利用资源，提高并行度？

-- multithreading

如何让空闲的部件忙起来?

多线程!

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

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
LW r1, 0(r2)	F	D	X	M	W										
LW r5, 12(r1)		F	D	D	D	D	X	M	W						
ADDI r5, r5, #12			F	F	F	F	D	D	D	D	X	M	W		
SW 12(r1), r5							F	F	F	F	D	D	D	D	

- 每条指令与前一条指令存在RAW相关

❑ 如何处理相关?

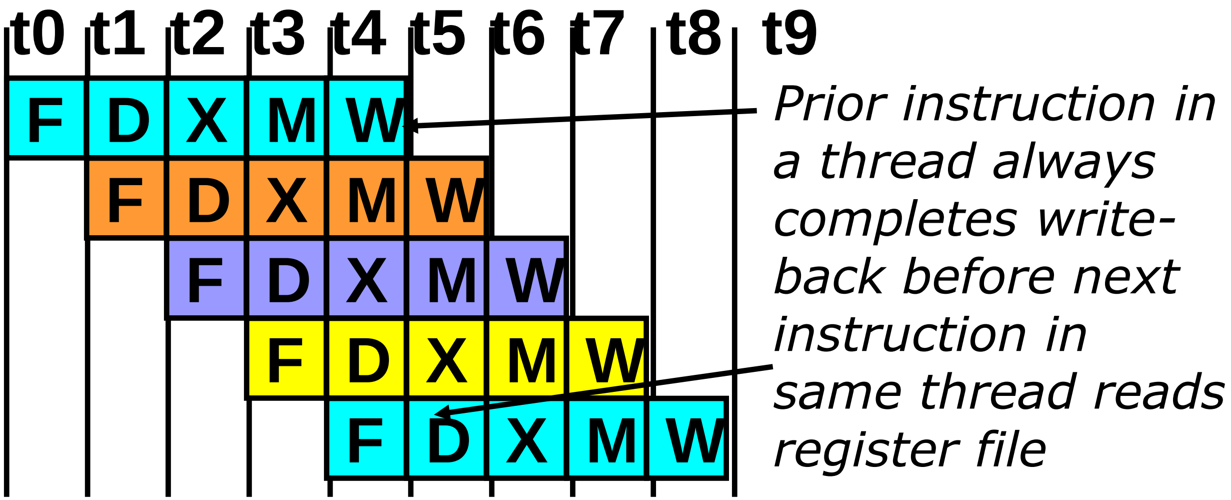
- 使用interlock机制(slow)
- 或定向路径

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

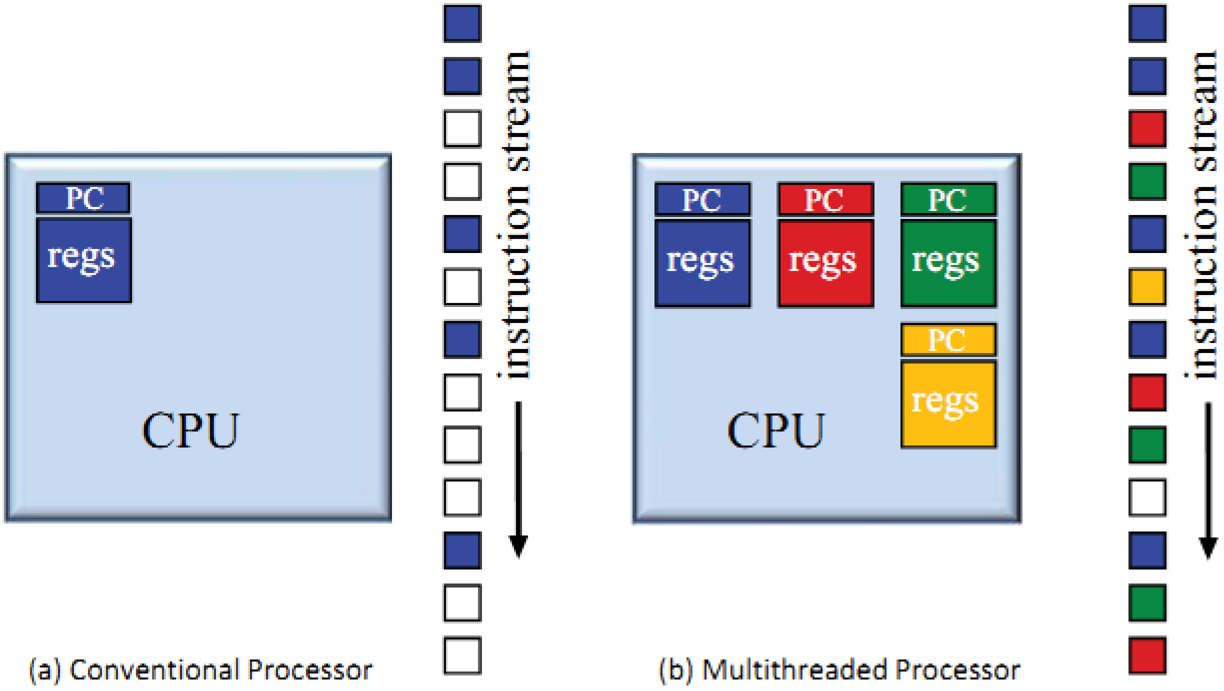
Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

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)



	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9
T1: LW r1, 0(r2)	F	D	X	M	W					
T2: ADD r7, r1, r4		F	D	X	M	W				
T3: XORI r5, r4, #12			F	D	X	M	W			
T4: SW 0(r7), r5				F	D	X	M	W		
T1: LW r5, 12(r1)					F	D	X	M	W	

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



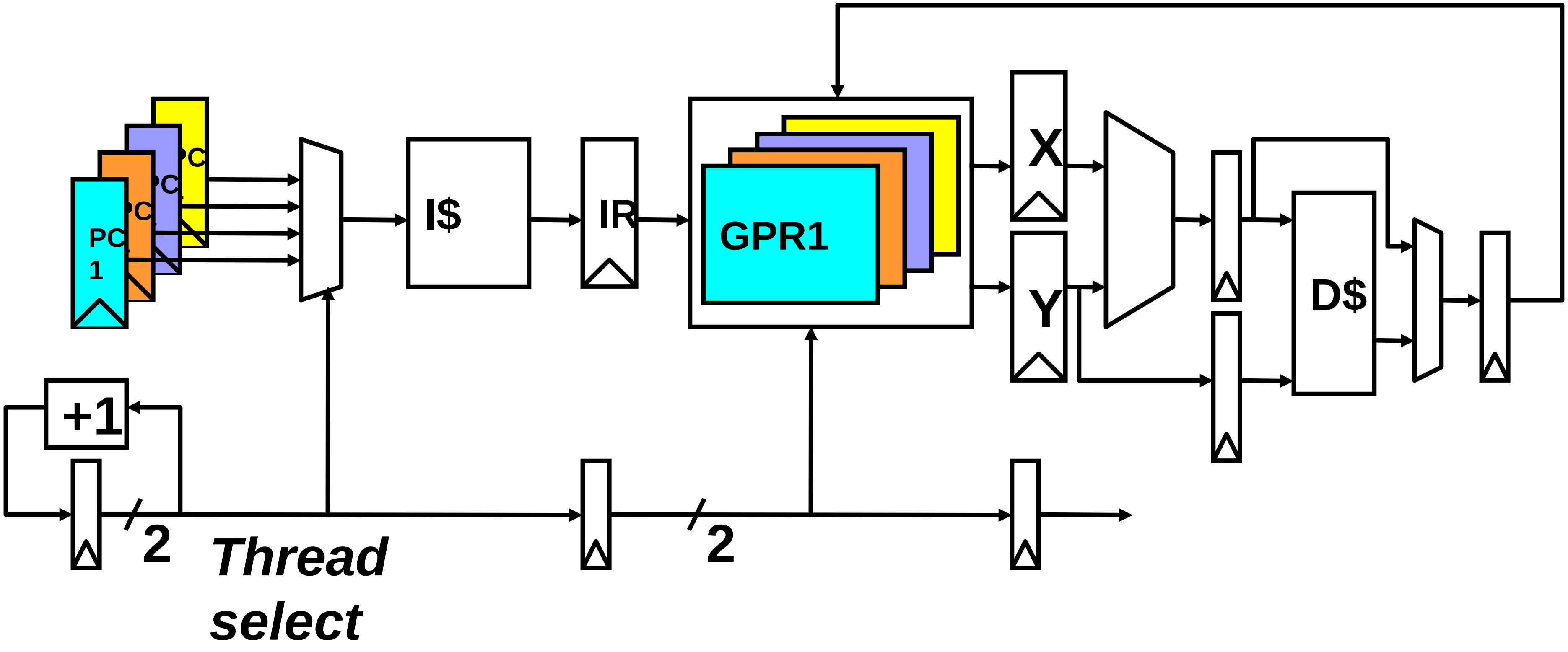
(a) Conventional Processor

(b) Multithreaded Processor

A conventional processor compared with a multithreaded processor.

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

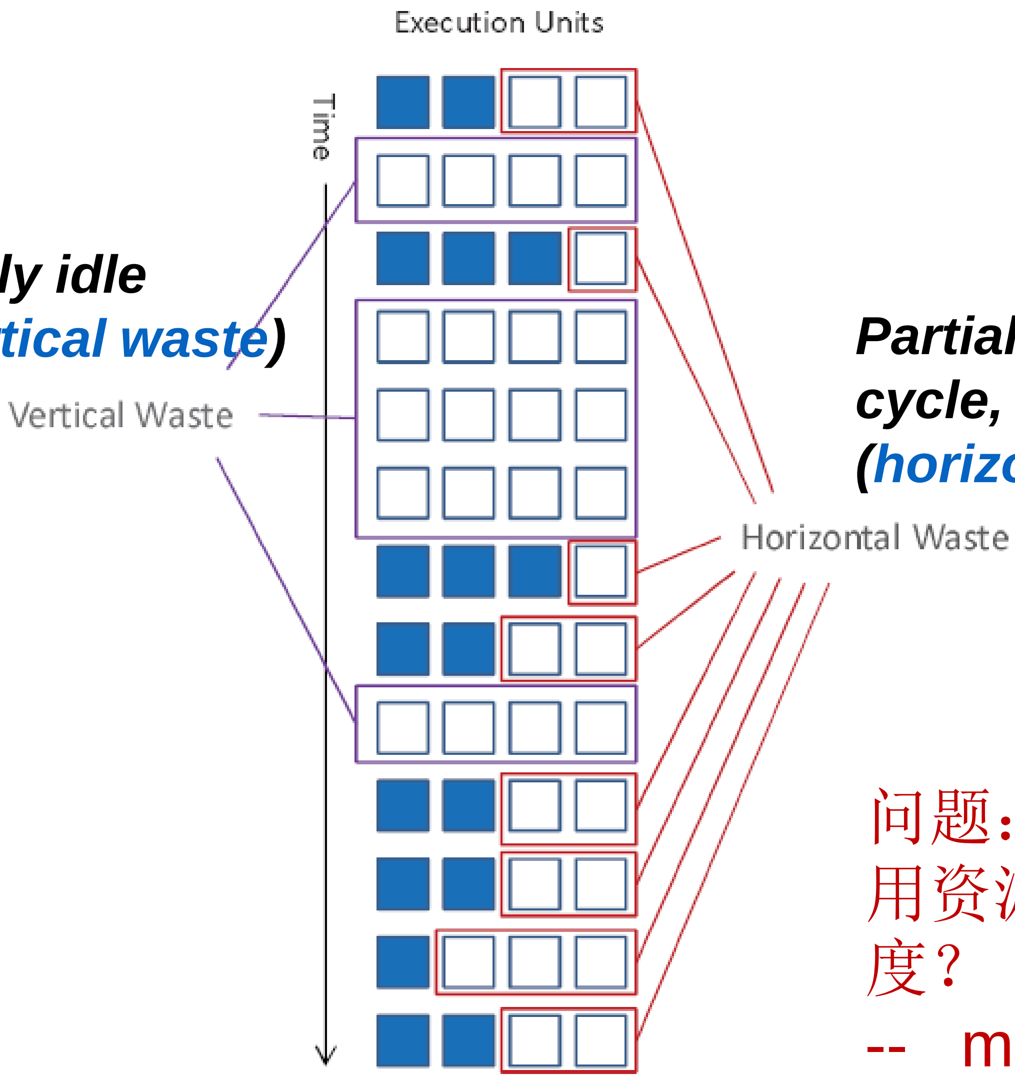
- 每个线程需要拥有自己的用户态信息（user state)： 包括PC、GPRs

- 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

Superscalar Machine Efficiency

Completely idle cycle (vertical waste)



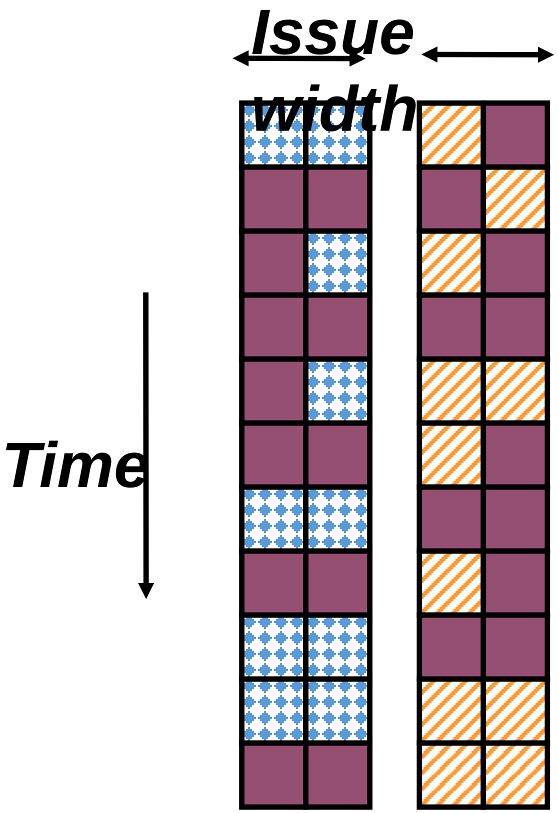
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

问题：如何充分利用资源，提高并行度？

-- multithreading

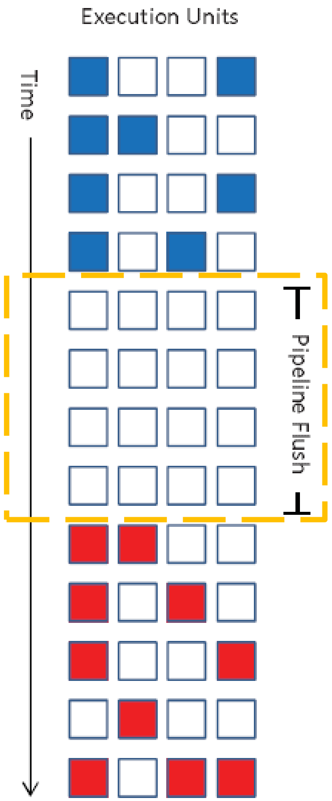
Chip Multiprocessing (CMP)

- 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4核2发射的CMP
  - 发射宽度为8，即同时可以发射8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?
  - 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

Coarse-Grain Multithreading

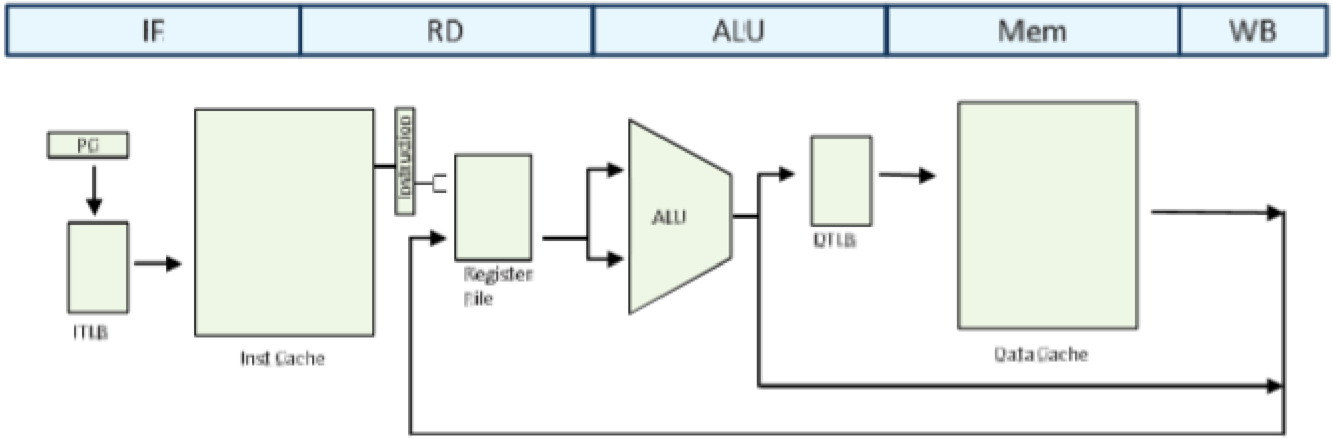


Figure 3.1: The MIPS R3000 Pipeline. This pipeline does not support multithreading.

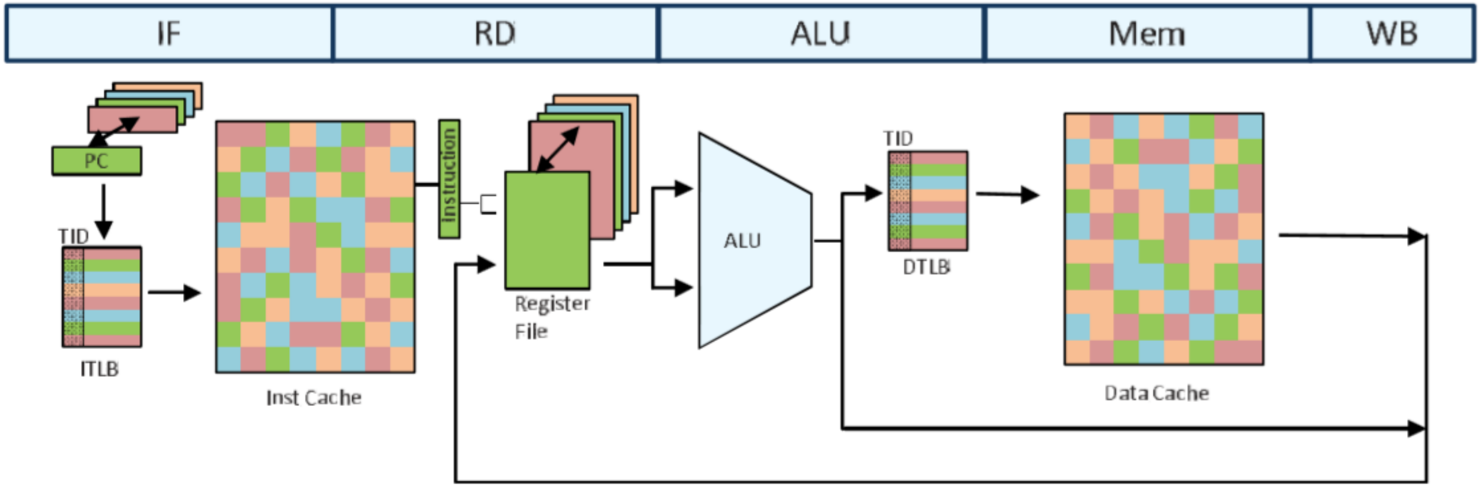
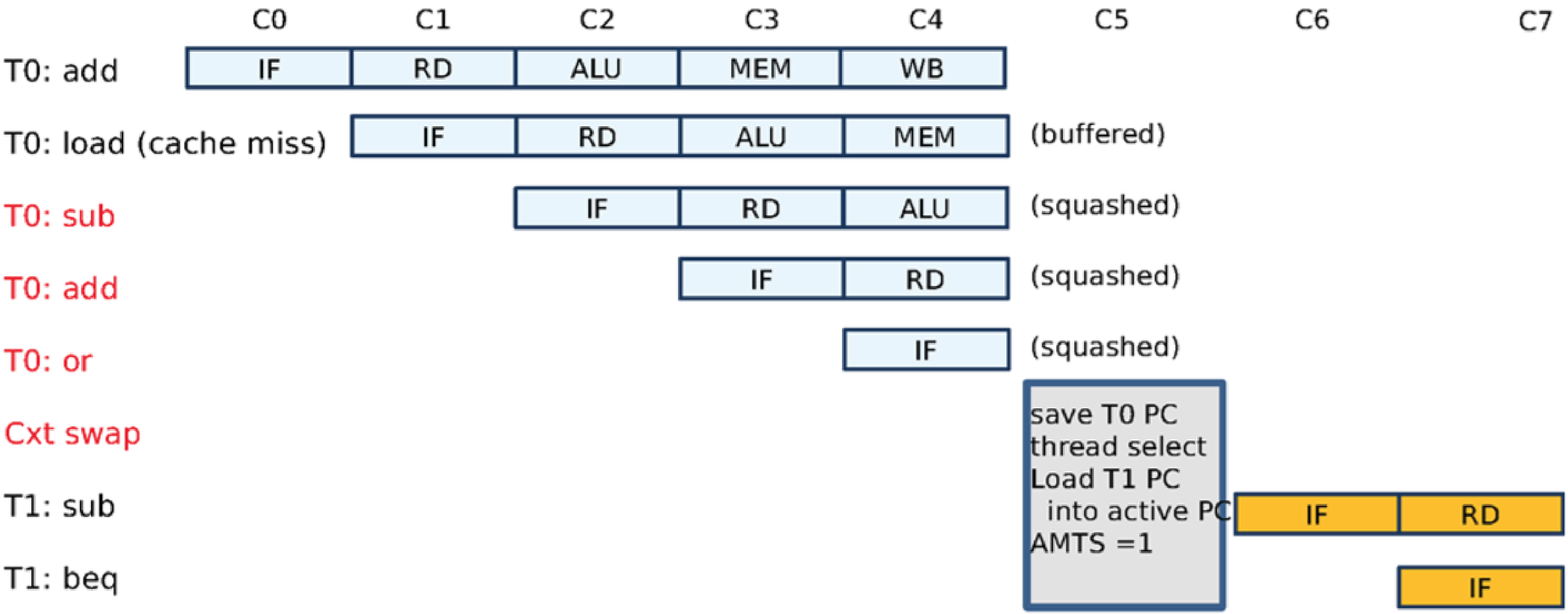


Figure 3.2: The MIPS R3000 Pipeline with support for Coarse-Grain Multithreading.

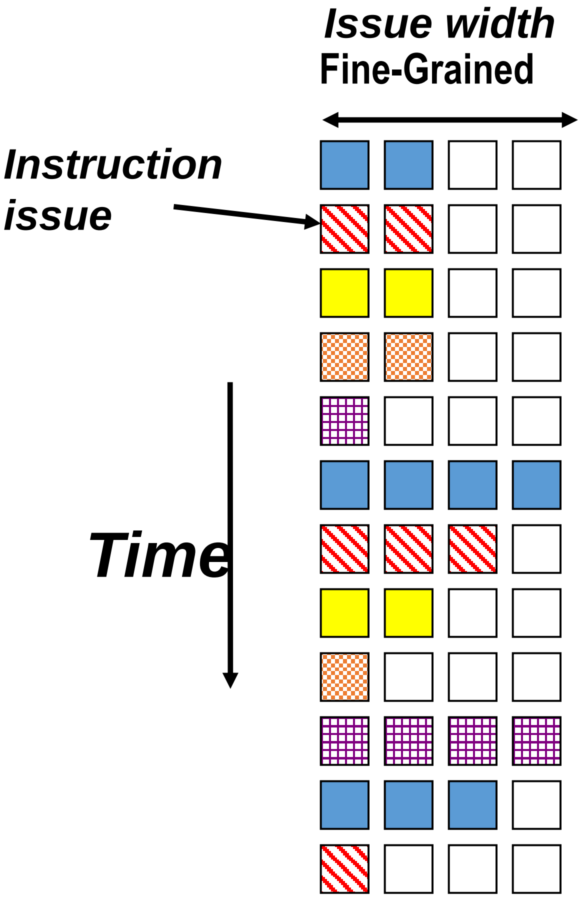
CGMT Context swap



	C0	C1	C2	C3	C4	C5	C6	C7
T0: add	IF	RD	ALU	MEM	WB			
T0: load (cache miss)		IF	RD	ALU	MEM	(buffered)		
T0: sub			IF	RD	ALU	(squashed)		
T0: add				IF	RD	(squashed)		
T0: or					IF	(squashed)		
Cxt swap						save T0 PC thread select Load T1 PC into active PC AMTS =1		
T1: sub							IF	RD
T1: beq								IF

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？
  - ● 减少甚至消除垂直方向的浪费
  - ● 仍然存在水平方向的浪费

Fine-Grain Multithreading

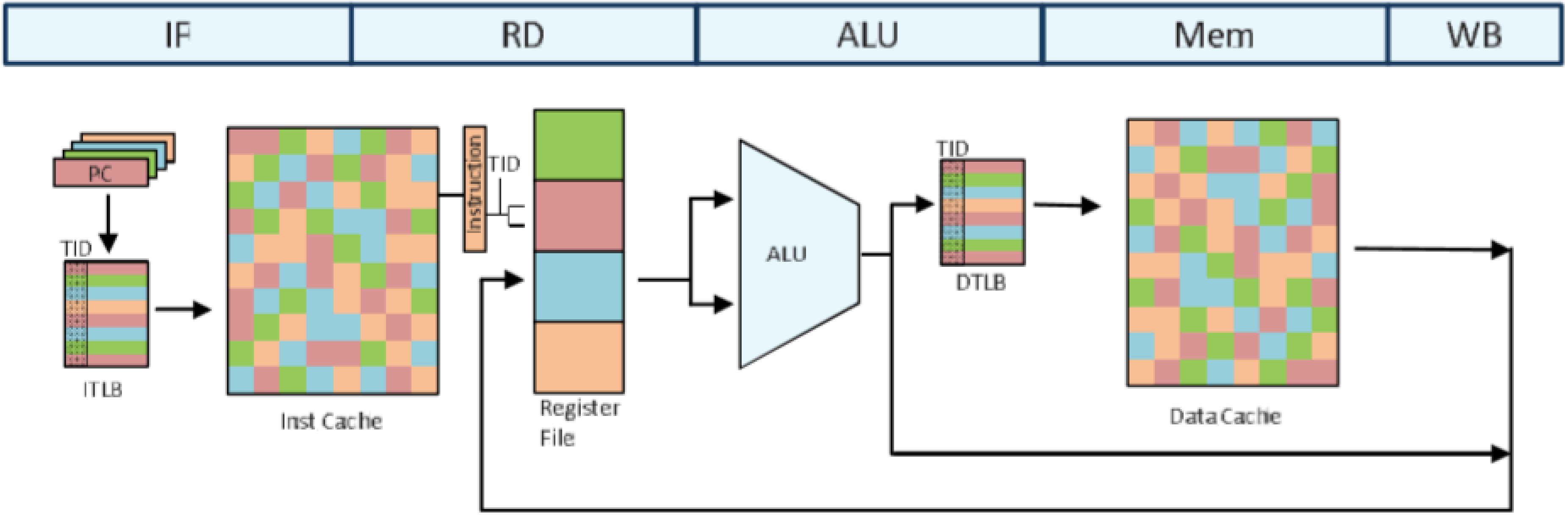
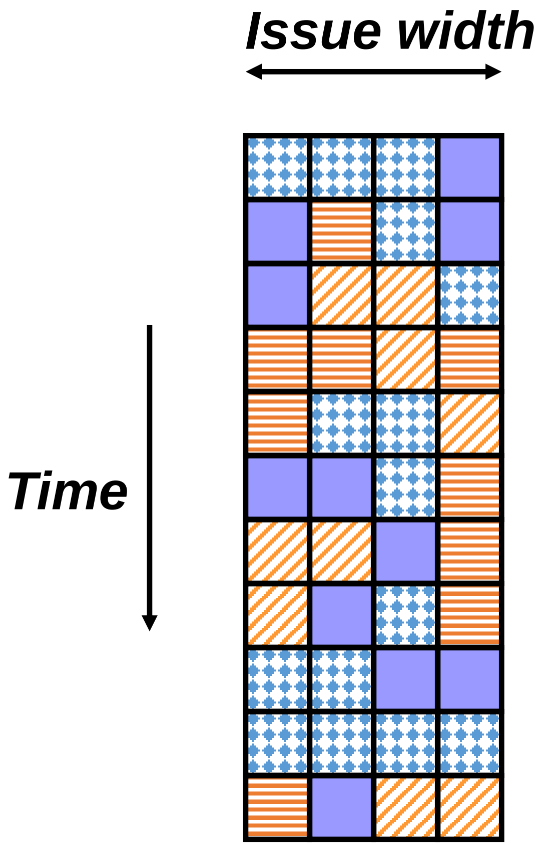


Figure 4.1: The MIPS R3000 Pipeline with support for Fine-Grain Multithreading. Although our target design would support 8 threads to maximize throughput, we show a 4-thread implementation for simplicity.

Ideal Superscalar Multithreading

- 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

- 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

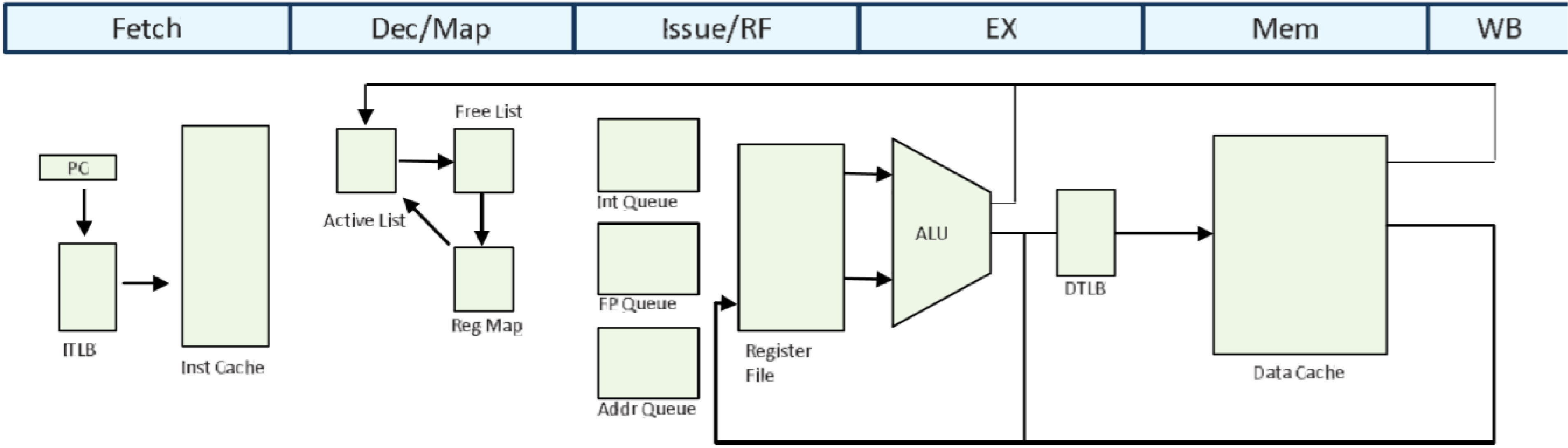


Figure 5.1: The pipeline of the MIPS R10000 processor, as seen by a load instruction.

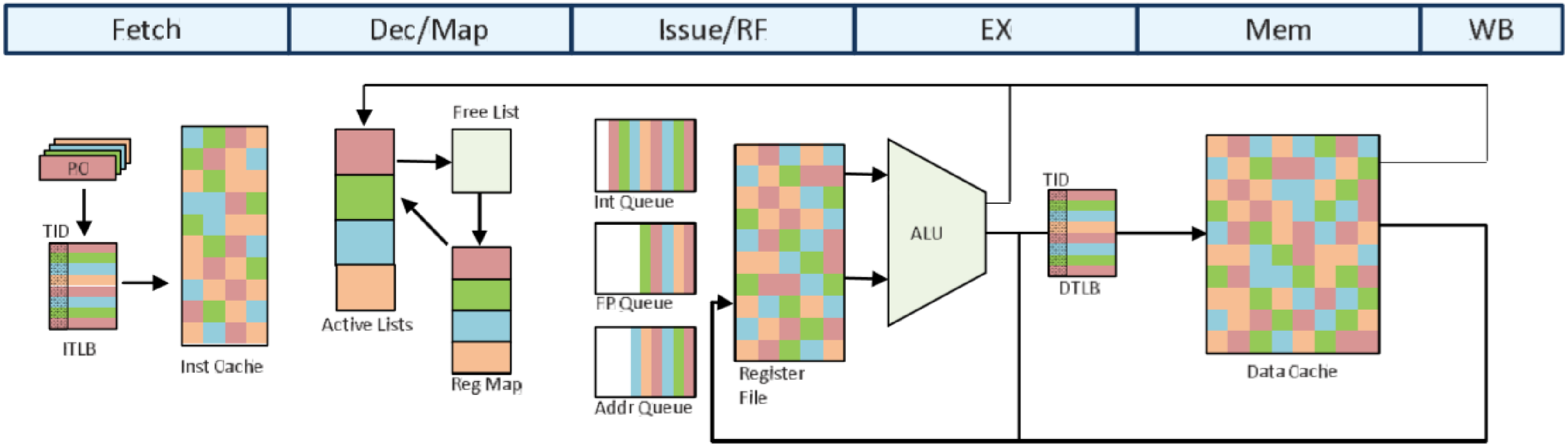
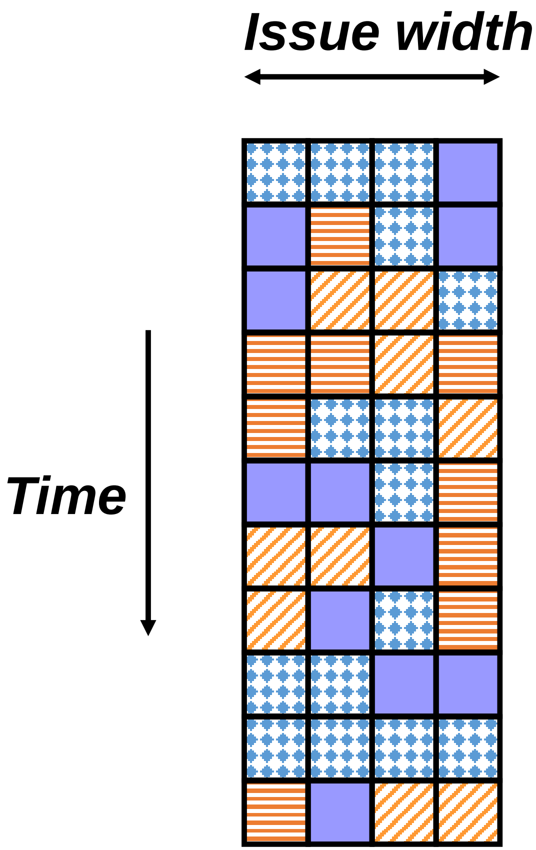


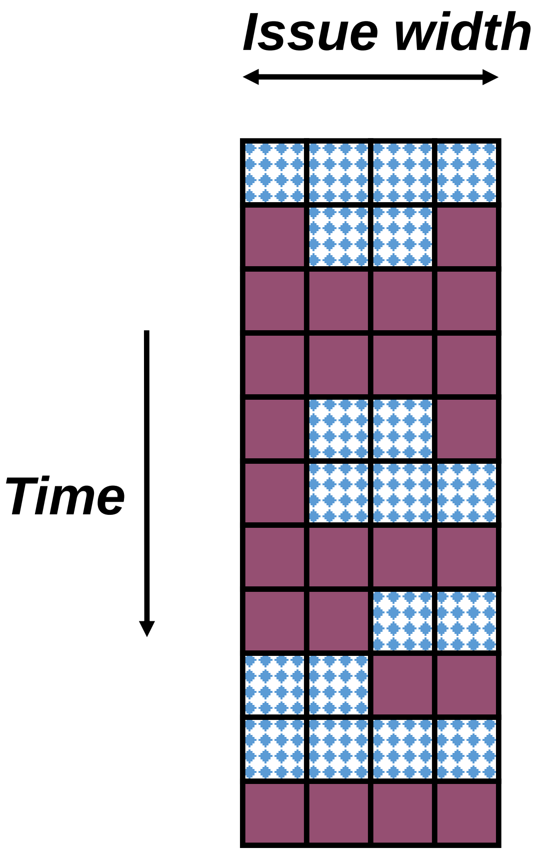
Figure 5.2: The MIPS R10000 pipeline with support added for simultaneous multithreading.

SMT adaptation to parallelism type

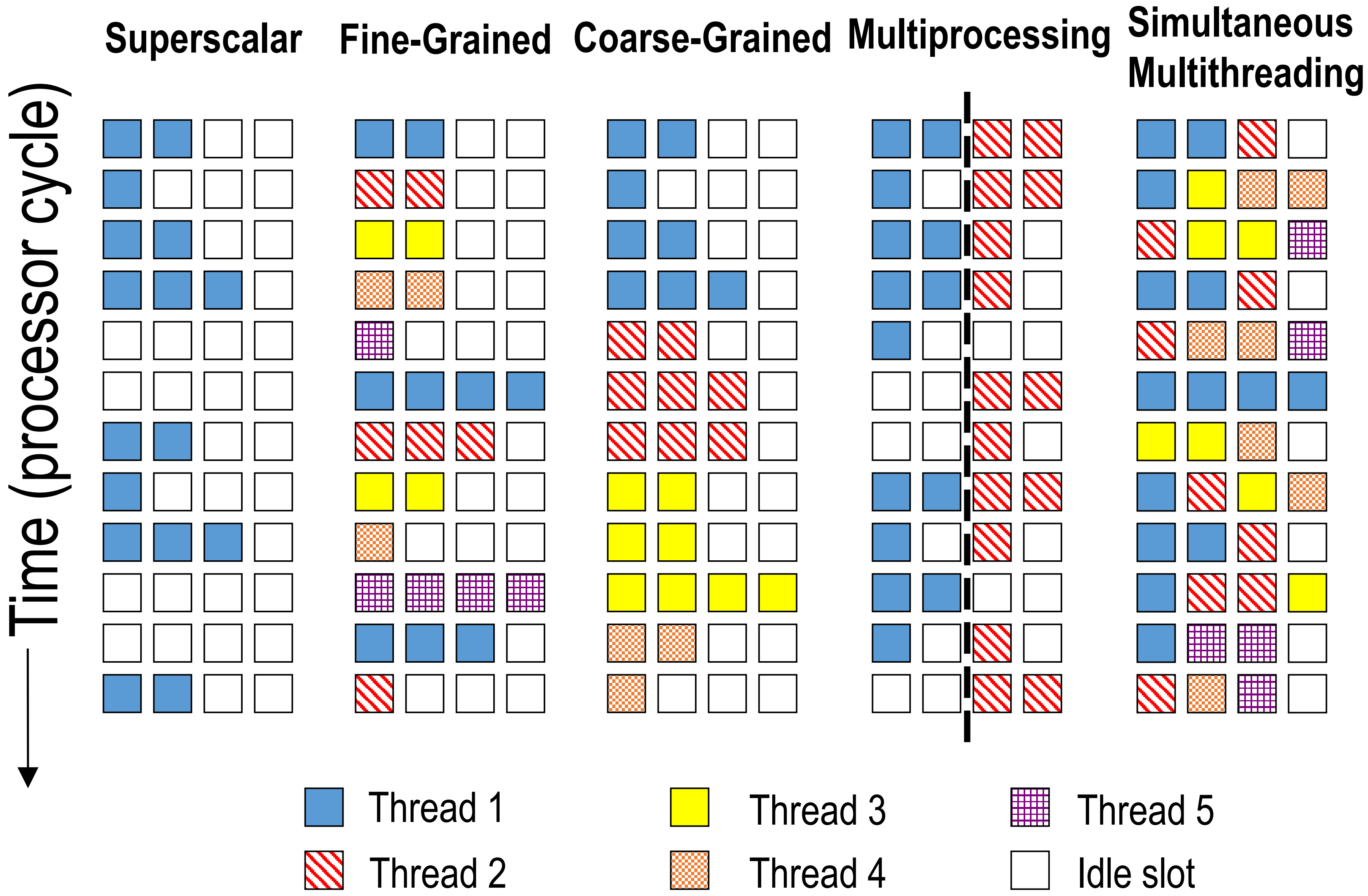
For regions with high thread level parallelism (TLP) entire machine width is shared by all threads

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





Summary: Multithreaded Categories



Superscalar

Fine-Grained

Coarse-Grained

Multiprocessing

Simultaneous Multithreading

Time (processor cycle)

Thread 1

Thread 2

Thread 3

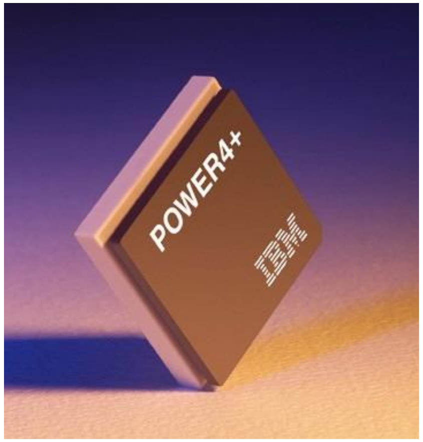
Thread 4

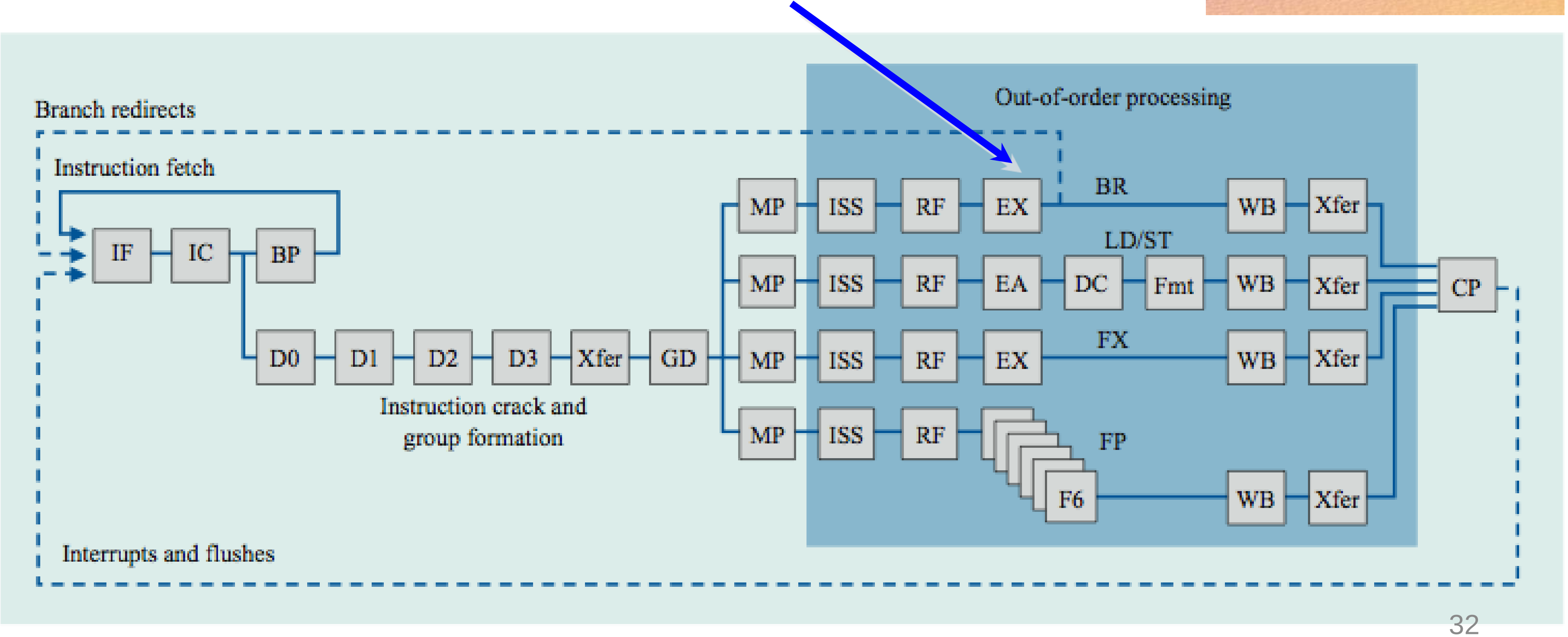
Thread 5

Idle slot

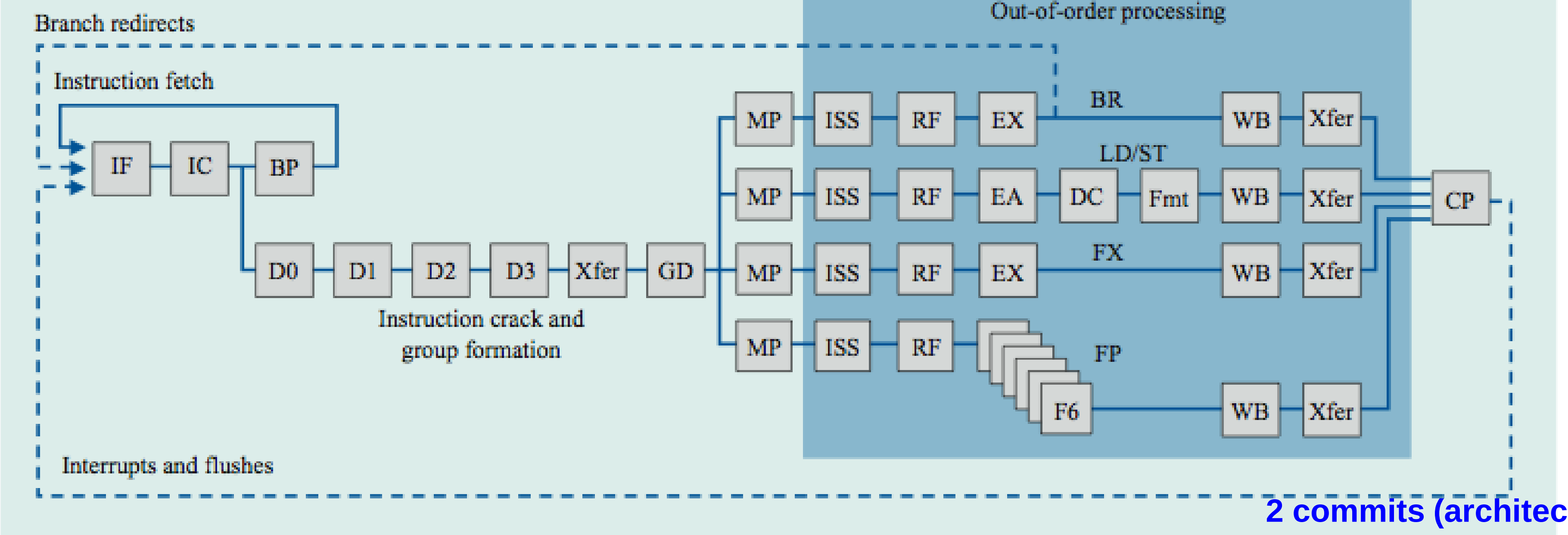
IBM Power 4

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



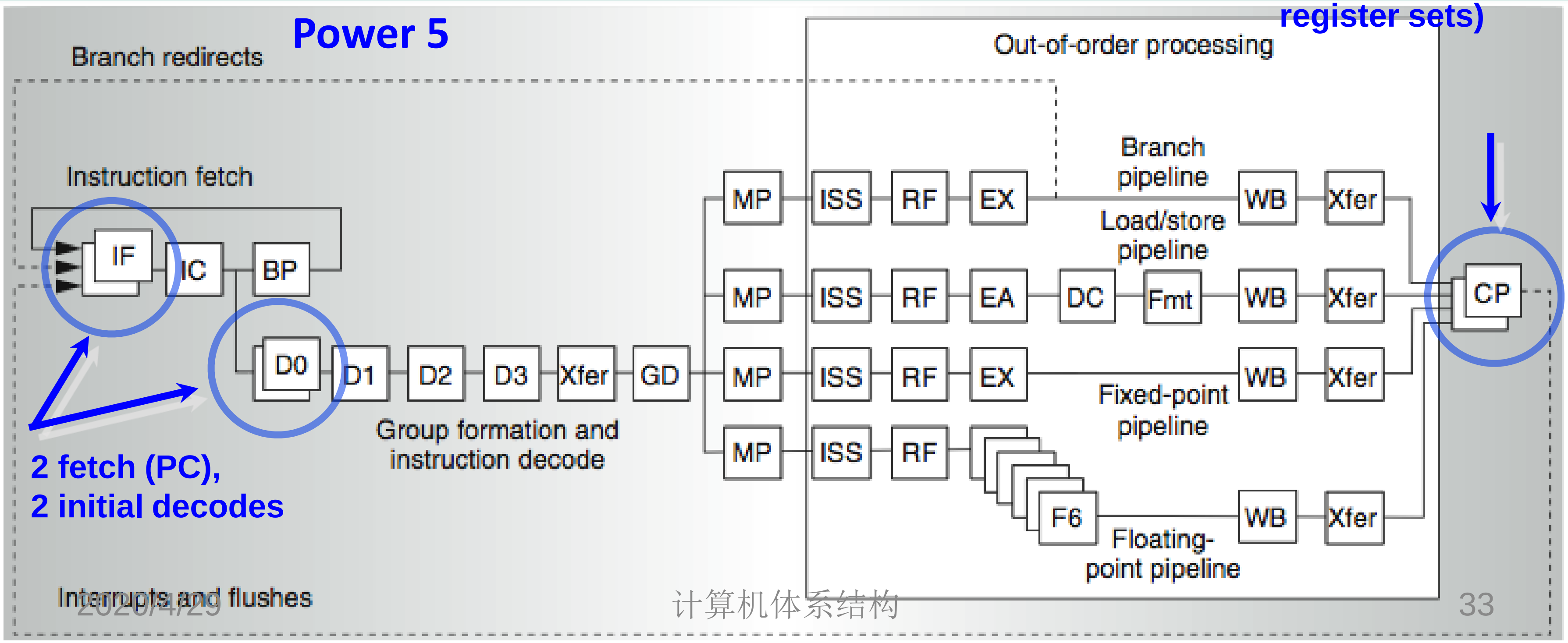


Power 4



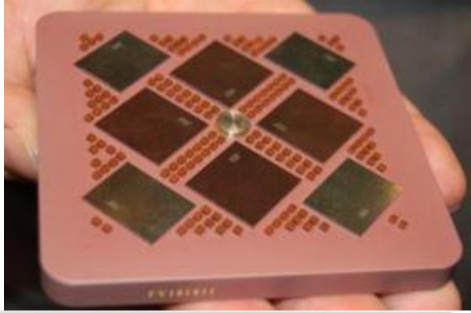
2 commits (architected register sets)

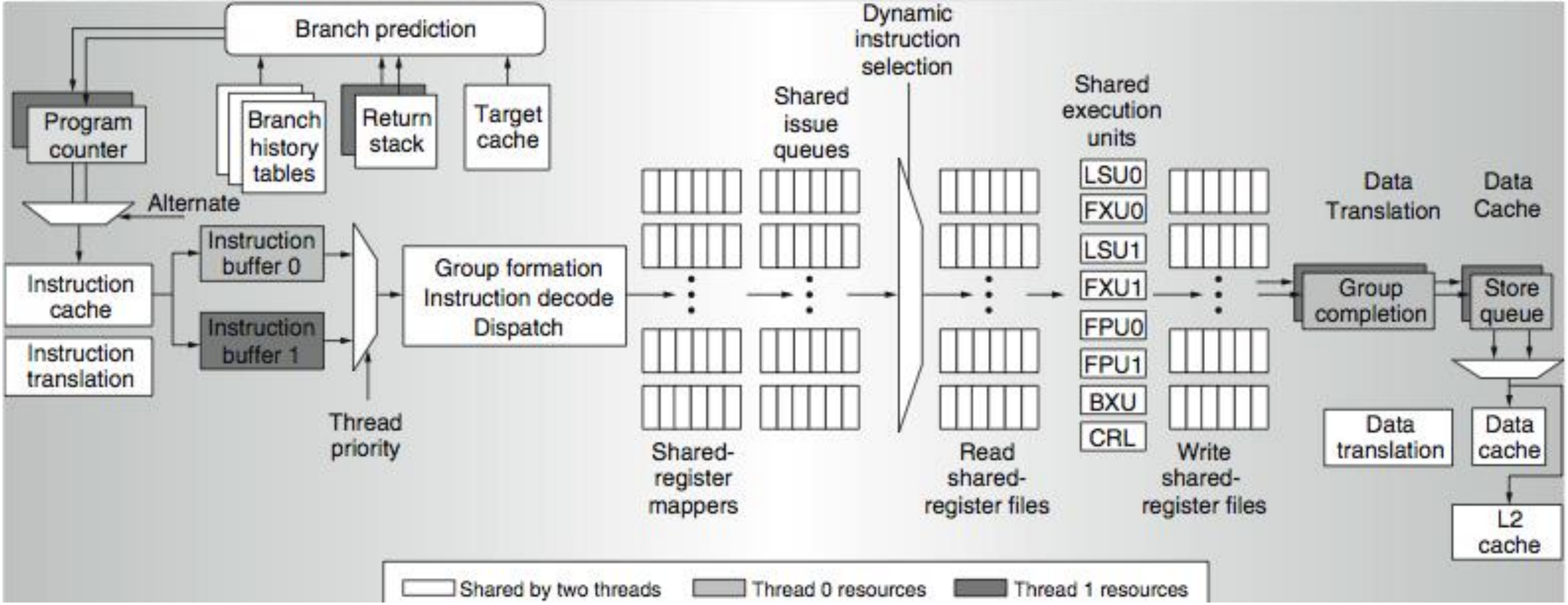
Power 5



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

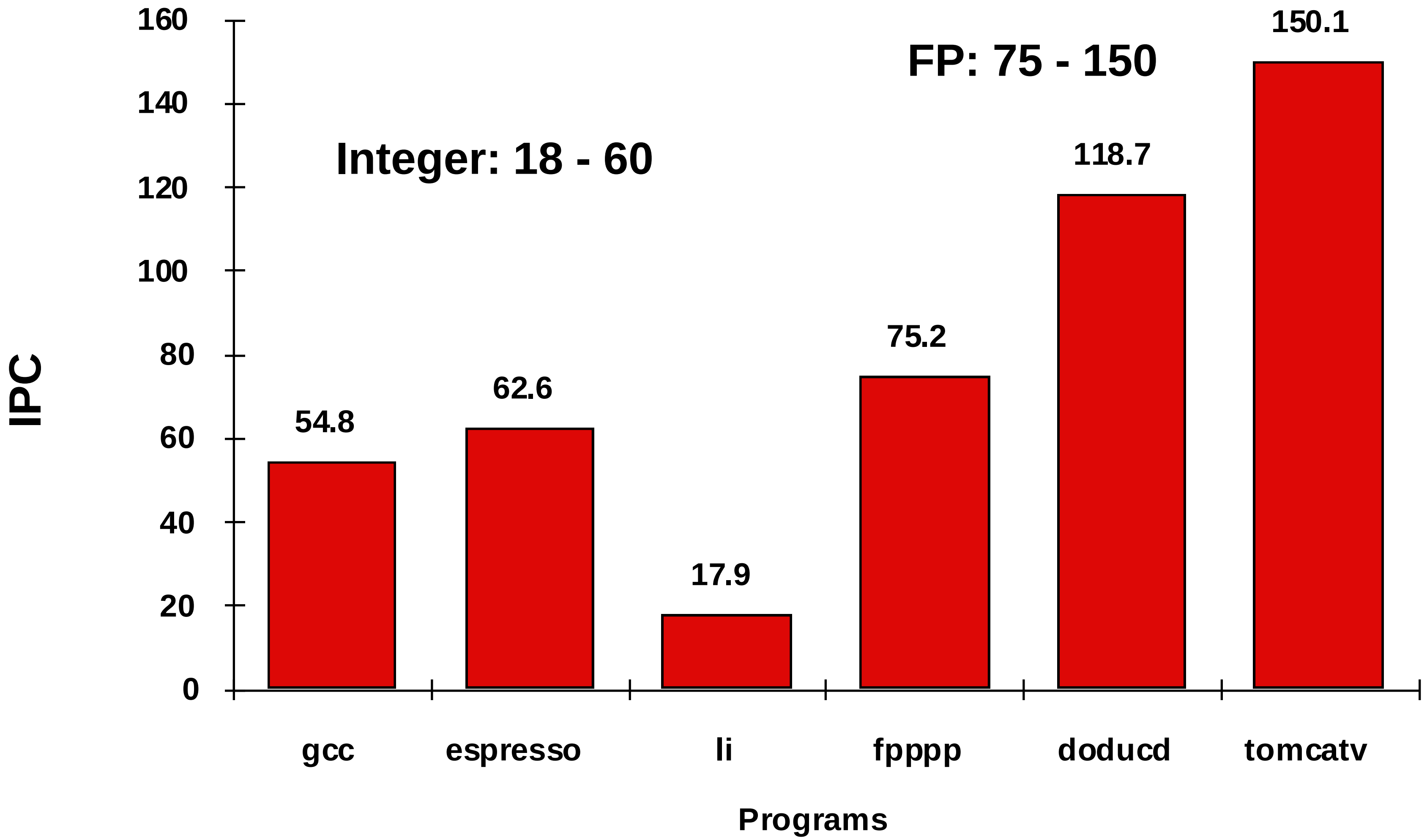
Changes in Power 5 to support SMT

- 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

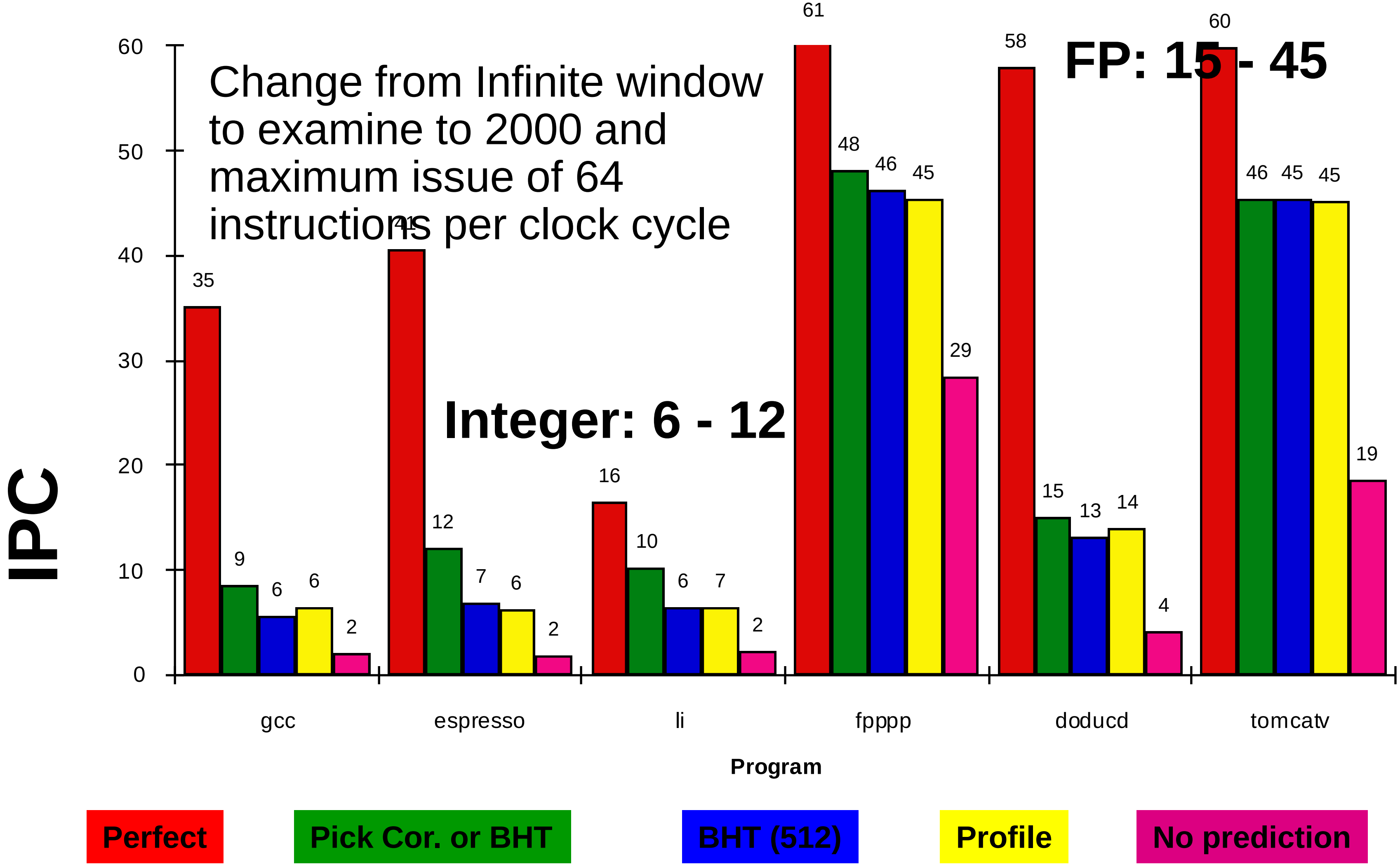
Pentium-4 Hyperthreading (2002)

- 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

Upper Limit to ILP: Ideal Machine



More Realistic HW: Branch Impact

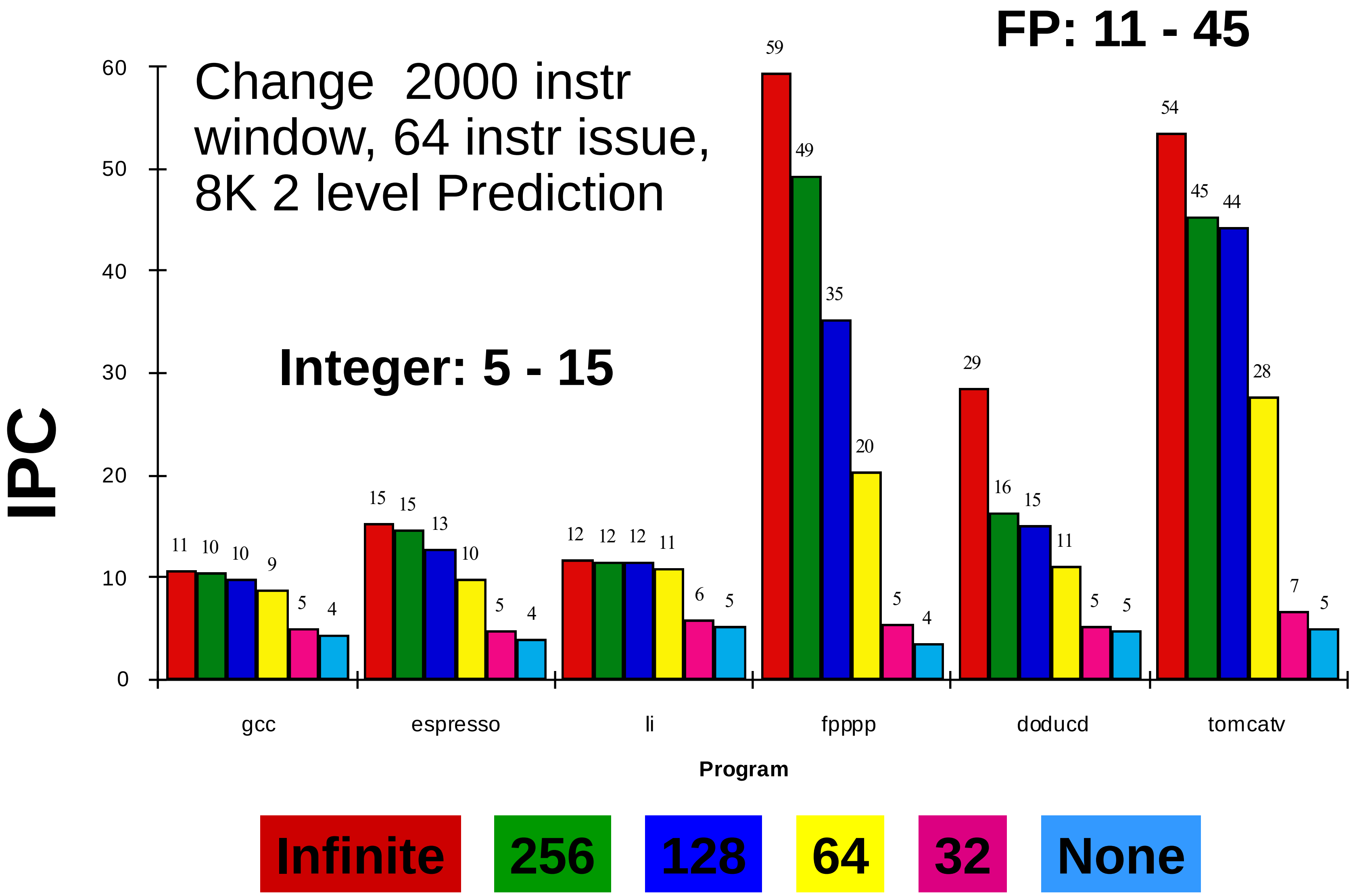


Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

Integer: 6 - 12

FP: 15 - 45

More Realistic HW: Register Impact (rename regs)

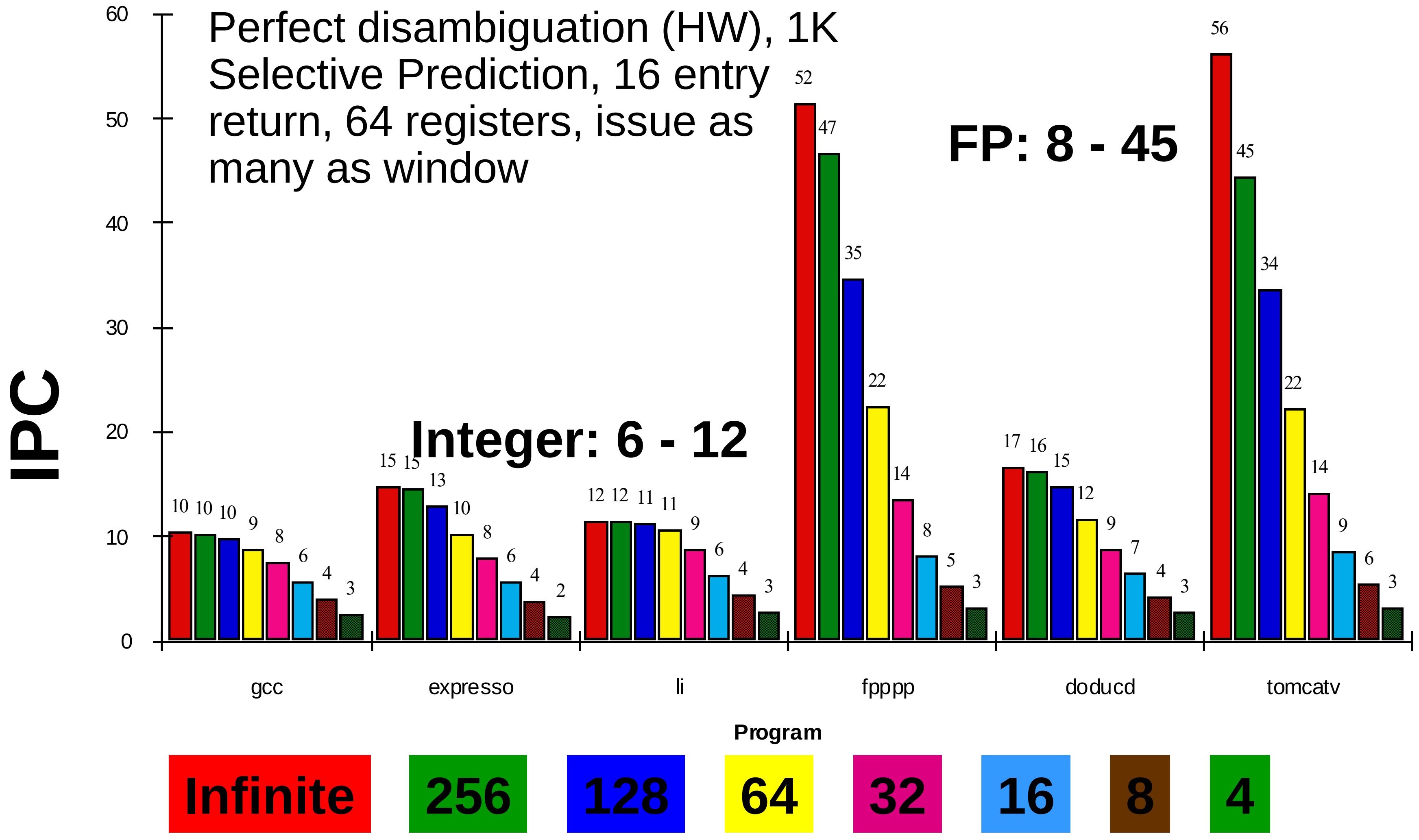


FP: 11 - 45

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction

Integer: 5 - 15

Realistic HW for ‘9X: Window Impact



Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window

Integer: 6 - 12

FP: 8 - 45

下一节内容

- 数据级并行性

From : H&P Computer Architecture: A Quantitative Approach,
Fifth Edition, （5th edition)

谢　谢！