16.650 Advanced Computer Architecture

Multithreading

Instructor: Prof. Yan Luo

Multithreading

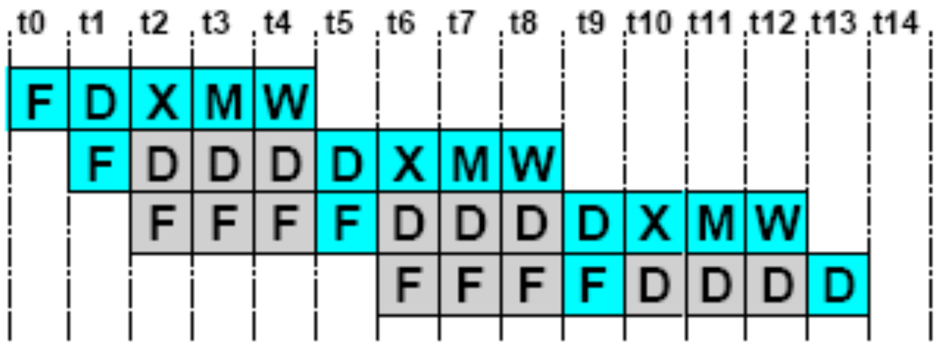
Consider the following sequence of instructions through a pipeline

LW r1, 0(r2)

LW r5, 12(r1)

ADDI r5, r5, #12

SW 12(r1), r5



t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
F	D	X	M	W										
	F	D	D	D	D	X	M	W						
		F	F	F	F	D	D	D	D	X	M	W		
						F	F	F	F	D	D	D	D	

Multithreading

- **How can we guarantee no dependencies between instructions in a pipeline?**
  - **One way is to interleave execution of instructions from different program threads on same pipeline – Micro context switching**

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

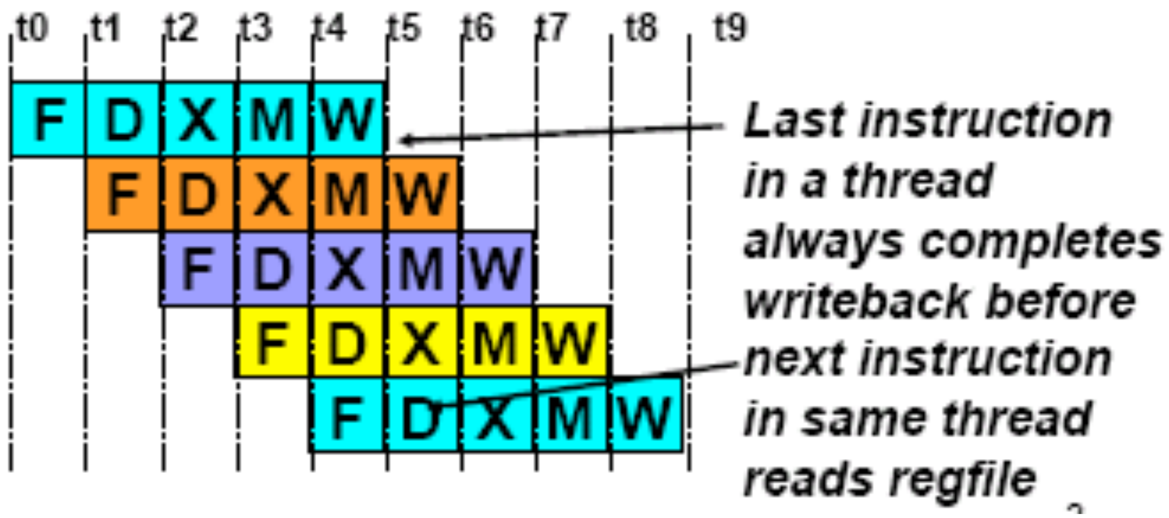
T1: LW r1, 0(r2)

T2: ADD r7, r1, r4

T3: XORI r5, r4, #12

T4: SW 0(r7), r5

T1: LW r5, 12(r1)

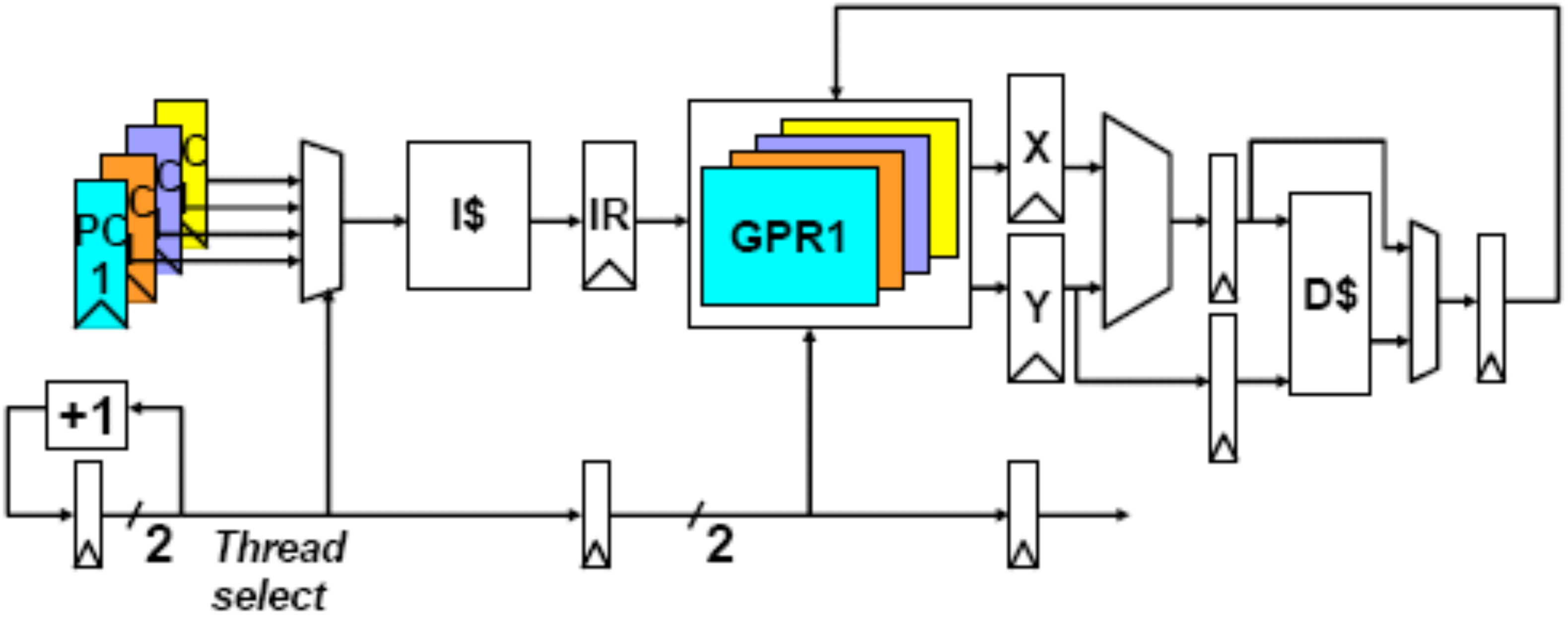


Last instruction in a thread always completes writeback before next instruction in same thread reads regfile

Avoiding Memory Latency

- General processors switch to another context on I/O operation => Multithreading, Multiprogramming, etc. An O/S function. Large overhead! Why?
- Why not context switch on a cache miss? => Hardware multithreading.
- Can we afford that overhead now? => Need changes in architecture to avoid stack operations. How to achieve it?
- Have many contexts CPU resident (not memory resident) by having separate PCs and registers for each thread. No need to store them in stack on context switching.

Simple Multithreaded Pipeline



- **Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage**

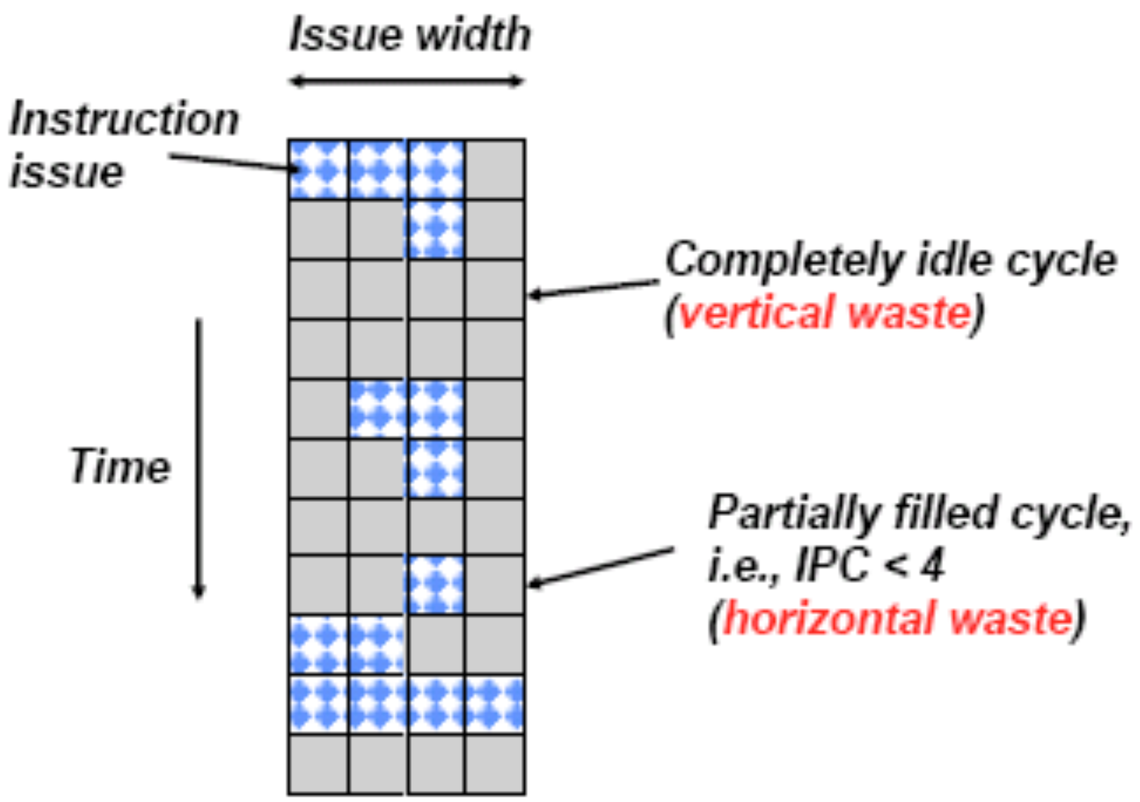
Multithreading Costs

- Appears to software (including OS) as multiple slower CPUs
- Each thread requires its own user state
  - GPRs
  - PC
- Also, needs own OS control state
  - virtual memory page table base register
  - exception handling registers
- *Other costs?*

What "Grain" Multithreading?

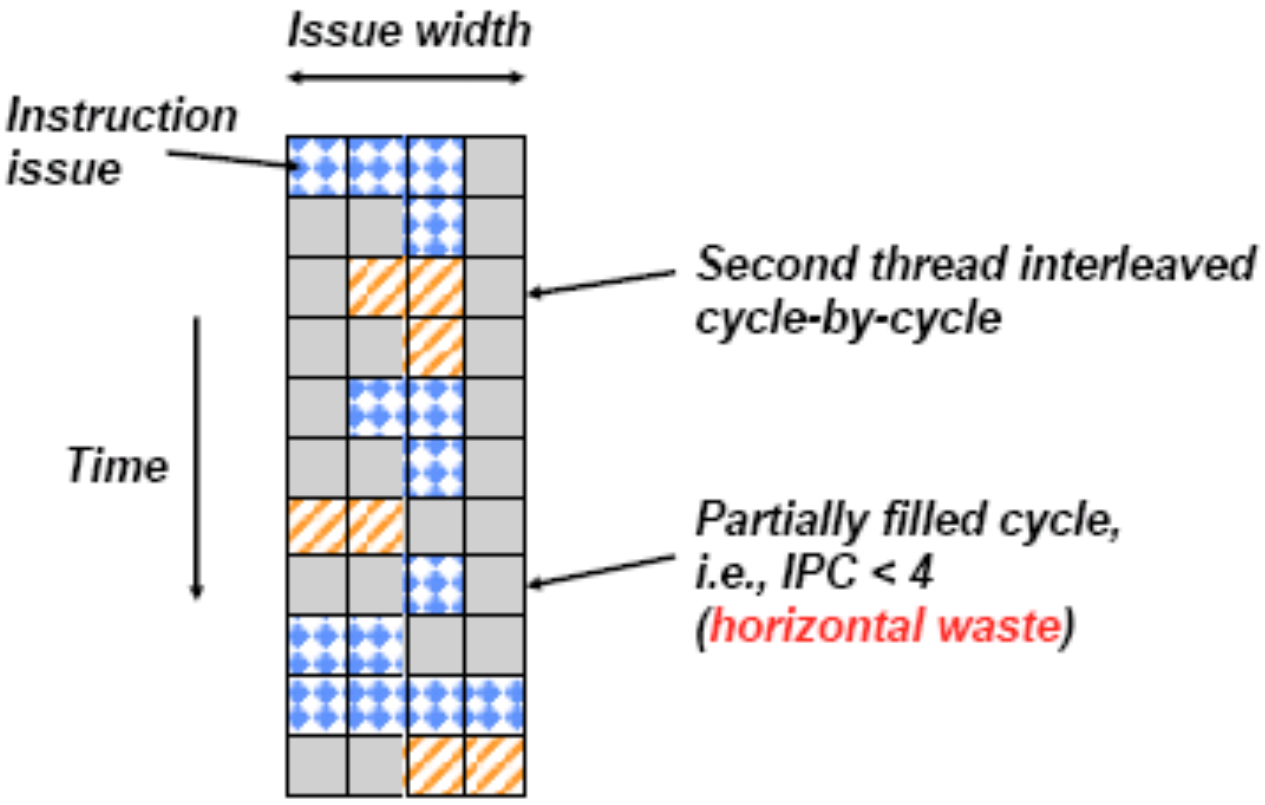
- **So far assumed fine-grained multithreading**
  - **CPU switches every cycle to a different thread**
  - ***When does this make sense?***
- **Coarse-grained multithreading**
  - **CPU switches every few cycles to a different thread**
  - ***When does this make sense (Ex - Memory Access? – NPs)?***

Superscalar Machine Efficiency


Issue width
Instruction issue
Time
Completely idle cycle (vertical waste)
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

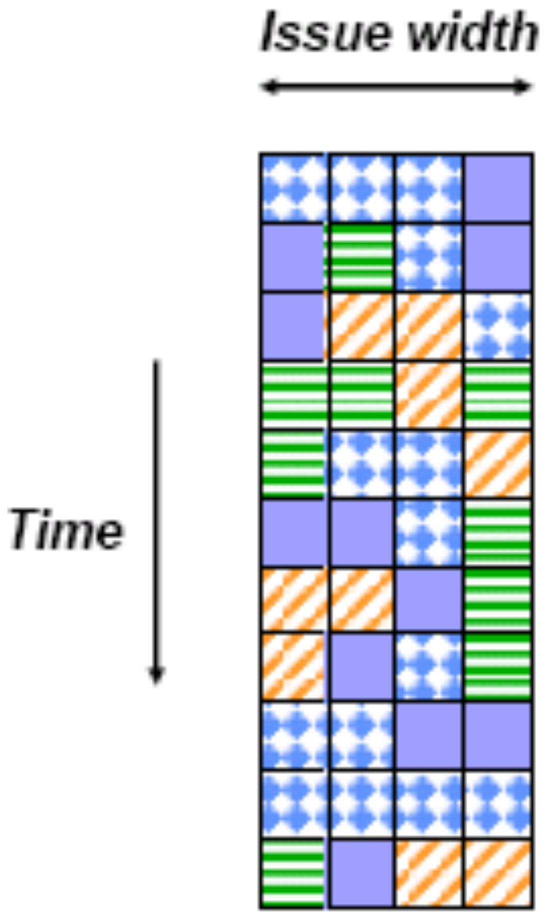
- *Why horizontal waste?*
- *Why vertical waste?*

Vertical Multithreading


Issue width
Instruction issue
Time
Second thread interleaved cycle-by-cycle
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

- **Cycle-by-cycle interleaving of second thread removes vertical waste**

Ideal Multithreading for Superscalar



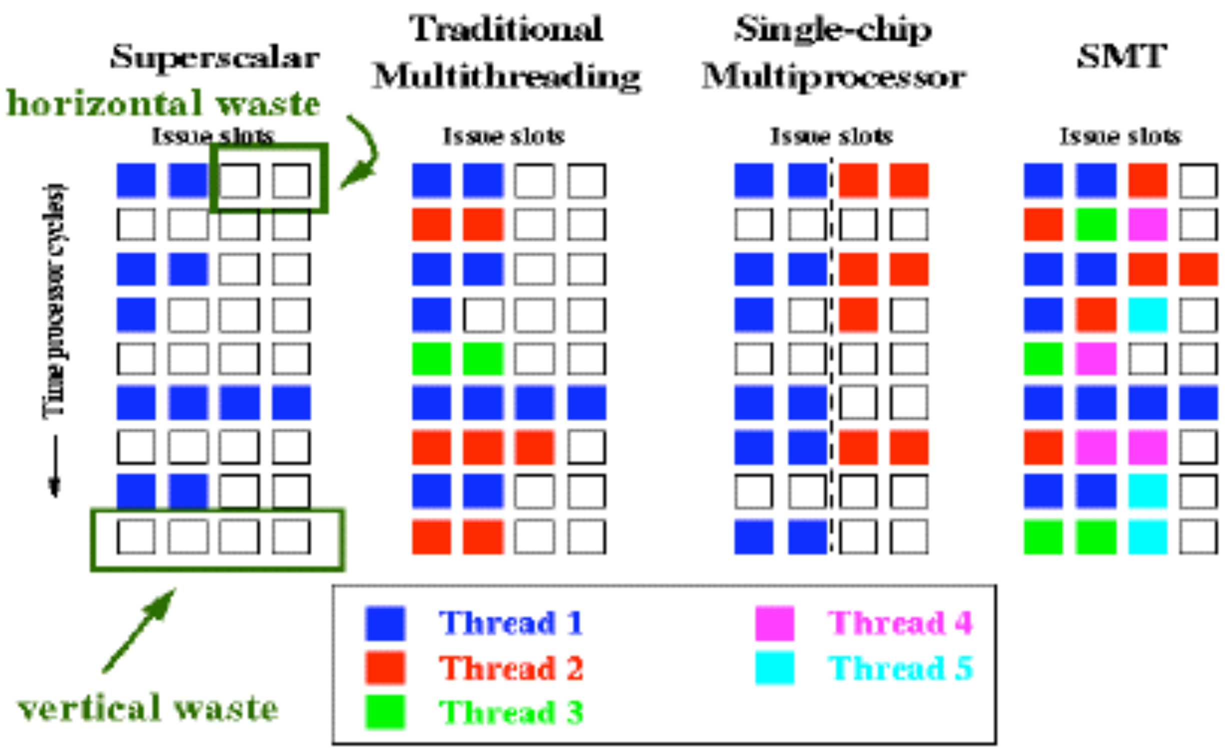
- **Interleave multiple threads to multiple issue slots with no restrictions**

Simultaneous Multithreading

- **Add multiple contexts and fetch engines to wide out-of-order superscalar processor**
  - **[Tullsen, Eggers, Levy, UW, 1995]**
- **OOO instruction window already has most of the circuitry required to schedule from multiple threads**
- **Any single thread can utilize whole machine**

Comparison of Issue Capabilities

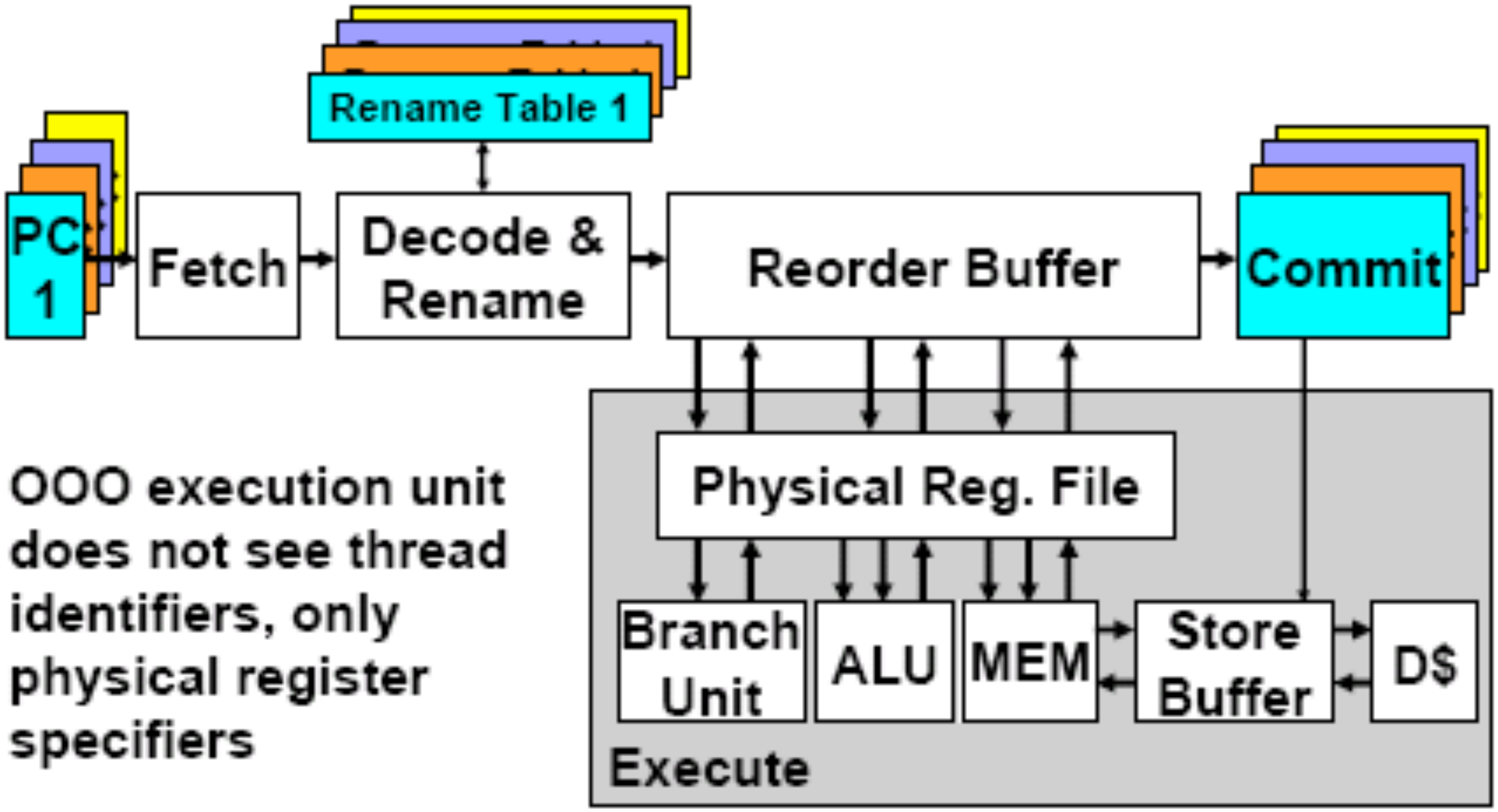
Courtesy of Susan Eggers; Used with Permission



From Superscalar to SMT

- **Small items**
  - **per-thread program counters**
  - **per-thread return stacks**
  - **per-thread bookkeeping for instruction retirement, trap & instruction dispatch queue flush**
  - **thread identifiers, e.g., with BTB & TLB entries**

Simultaneous Multithreaded Processor



OOO execution unit does not see thread identifiers, only physical register specifiers

Intel Xeon Processor

- **Dual core: two cores on chip**
- **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**
- **A processor running only one active software thread to run at the same speed with or without hyperthreading**