# Master Informatics Eng.

2017/18

*A.J.Proença*

## From ILP to Multithreading

*(most slides are borrowed)*



## *Processor arch: beyond Instruction-Level Parallelism*

- When exploiting ILP, goal is to minimize CPI
  - Pipeline CPI (*efficient to exploit loop-level parallelism*) =>
    - Ideal pipeline CPI + ✔
    - Structural stalls + ✔
    - Data hazard stalls + ✔
    - Control stalls + ✔
    - Memory stalls ... *cache techniques* ✔...
  - Multiple issue =>
    - find enough parallelism to keep pipeline(s) occupied
- Multithreading =>
  - find additional ways to keep pipeline(s) occupied
- Insert data parallelism features ...*(next set of slides)*

# Multiple Issue and Static Scheduling



- To achieve CPI < 1, need to complete multiple instructions per clock cycle
- Solutions:
  - statically scheduled superscalar processors
  - VLIW (very long instruction word) processors
  - dynamically scheduled superscalar processors




# Multiple Issue

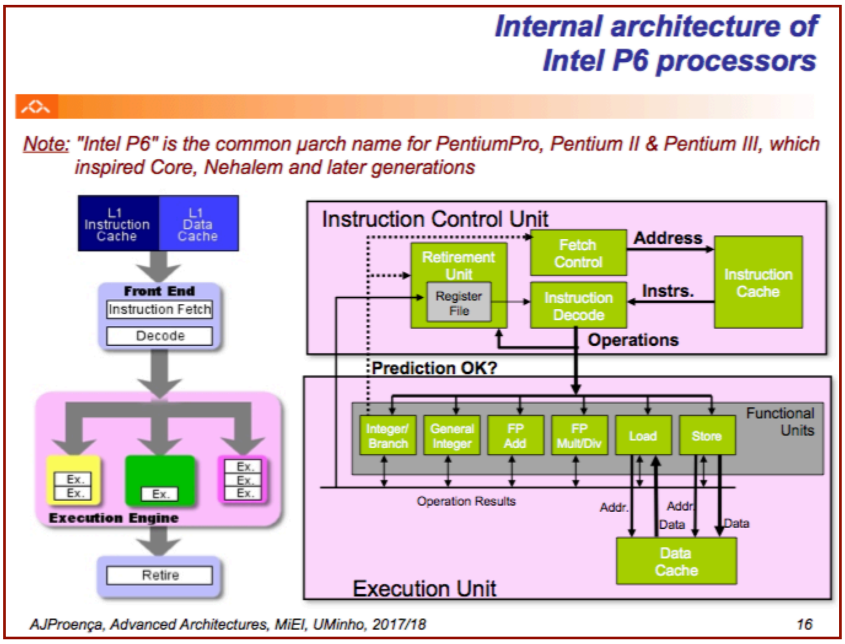


| Common name | Issue structure | Hazard detection | Scheduling | Distinguishing characteristic | Examples |
|---|---|---|---|---|---|
| Superscalar (static) | Dynamic | Hardware | Static | In-order execution | Mostly in the embedded space: MIPS and ARM, including the ARM Cortex A8 , Atom |
| Superscalar (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 |

*EPIC: Explicitly Parallel Instruction Computer*

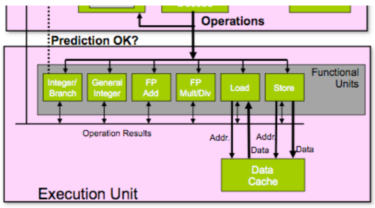
# *The **n**-way superscalar P6: how this architecture supports **n**-way multiple issue*

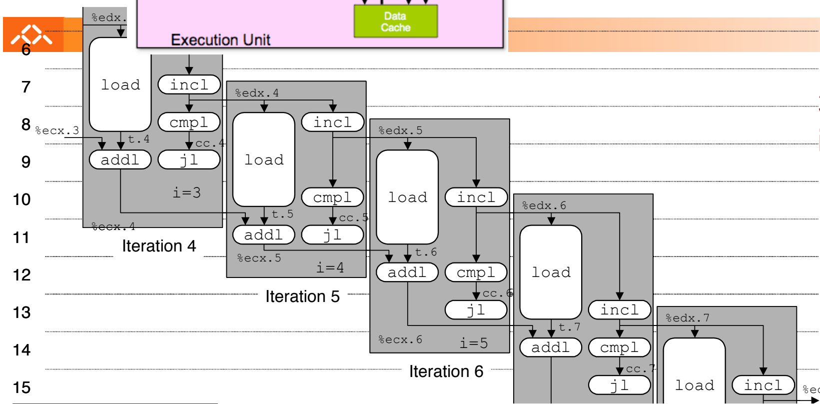
As seen before...



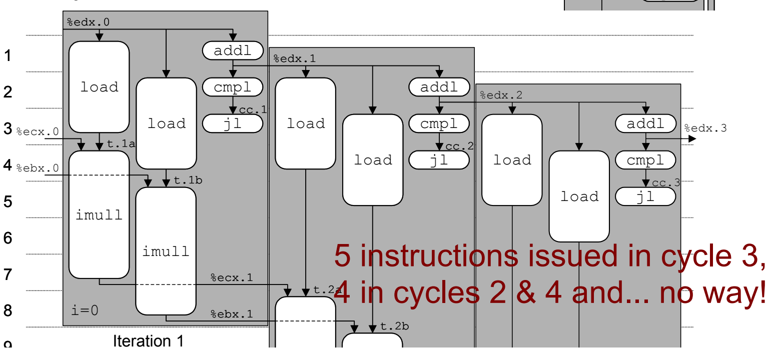



# *Executions views of `combine`: can this architecture be **3**-way?*



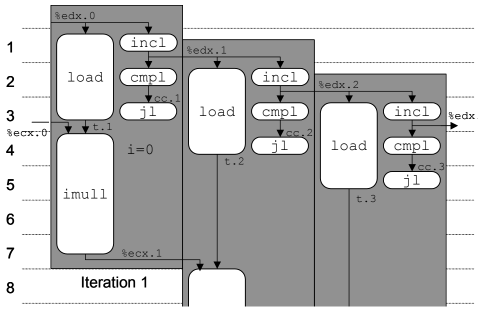




3 instructions issued in cycles 8, 9... ok!




5 instructions issued in cycle 3, 4 in cycles 2 & 4 and... no way!

4 instructions issued in cycle 3... no way!



# Multithreading



- Performing multiple threads of execution in parallel
  - Replicate registers, PC/IP, etc.
  - Fast switching between threads
- Fine-grain multithreading / time-multiplexed MT
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

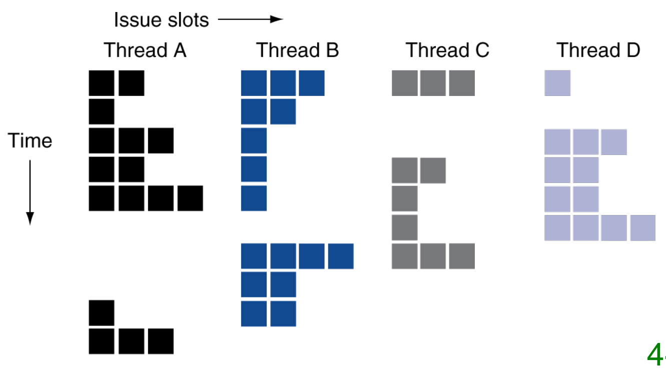


# Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches

*__HT__: Hyper-Threading, Intel trade mark for their SMT implementation*
*MT in Xeon Phi KNC: 4-way SMT with time-mux MT, __not HT__!*

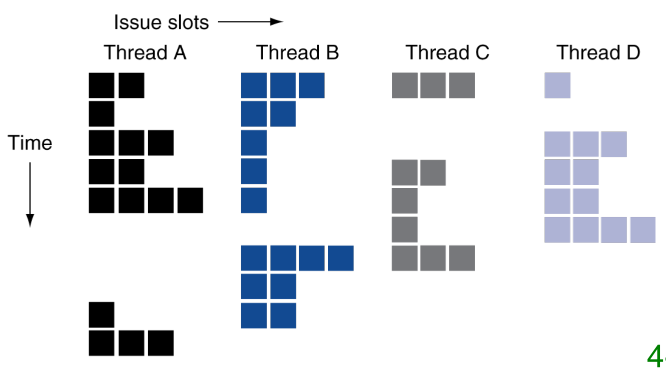
# Multithreading Example



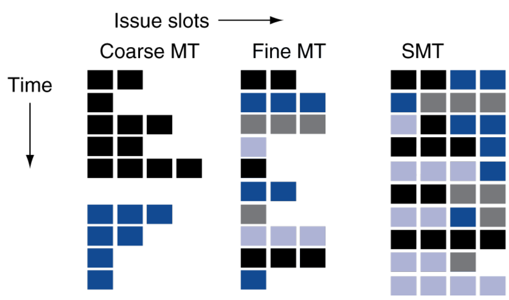

4-way superscalar



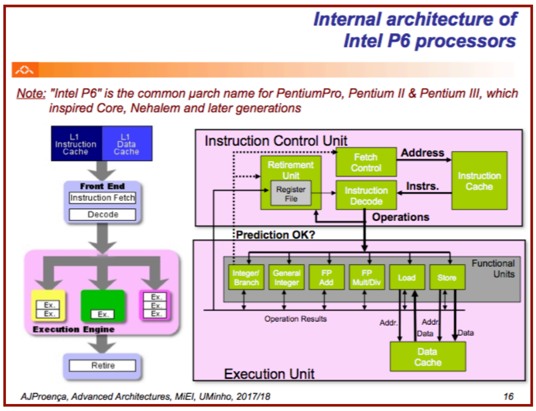
# Multithreading Example

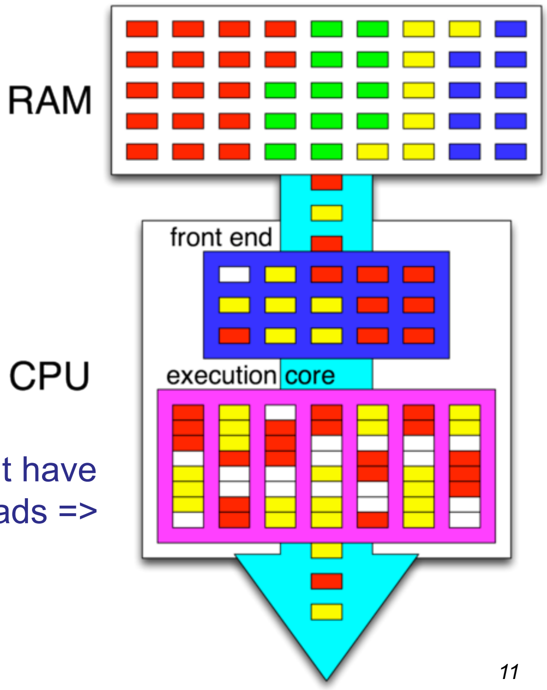



4-way superscalar




## As seen before...




The pipelined functional units might have better use if shared among more threads =>

*Note: white boxes are bubbles...*



## *Reading suggestions* *(from CAQA 5th Ed)*

- Concepts and challenges in ILP: section 3.1
- Exploiting ILP w/ multiple issue & static scheduling: 3.7
- Exploiting ILP w/ dyn sched, multiple issue & specul: 3.8
- Multithread: exploiting TLP on uniprocessors: 3.12
- Multiprocessor cache coherence and snooping coherence protocol with example: 5.2
- Basics on directory-based cache coherence: 5.4
- Models of memory consistency: 5.6
- A tutorial by Sarita Ave & K. Gharachorloo (*see link at website*)