Hardware Multithreading

Coarse-grained
Fine-grained
Simultaneous
Dynamically Scheduled CPU

Instruction fetch and decode unit

Reservation station
Reservation station
... Reservation station
Reservation station

Instruction fetch and decode unit

Instruction issue

Preserves dependencies

Hold pending operands

Out-of-order execute

Results also sent to any waiting reservation stations

Commit unit

Reorders buffer for register writes

In-order commit

Can supply operands for issued instructions
Does Multiple Issue Work?

» Yes, but not as much as we’d like
» Programs have real dependencies that limit ILP
» Some dependencies are hard to eliminate
  – e.g., pointer aliasing
» Some parallelism is hard to expose
  – Limited window size during instruction issue
» Memory delays and limited bandwidth
  – Hard to keep pipelines full
» Speculation can help if done well

Chapter 4 — The Processor — 3
Figure 4.11.2: The ARM A8 pipeline (COD Figure 4.75). The first three stages fetch instructions into a 12-entry instruction fetch buffer. The Address Generation Unit (AGU) uses a Branch Target Buffer (BTB), Global History Buffer (GHB), and a Return Stack (RS) to predict branches to try to keep the fetch queue full. Instruction decode is five stages and instruction execution is six stages.
Figure 4.11.3: CPI on ARM Cortex A8 for the Minnespec benchmarks, which are small versions of the SPEC2000 benchmarks (COD Figure 4.76)
How can we make better use of the execution units?

» Have two or more separate threads of control running, alternately using the pipeline

» This is called multi-threading
Coarse- grained multithreading

- Single thread runs until a costly stall
  - E.g. 2nd level cache miss
- Another thread starts during stall for first
  - Pipeline fill time requires several cycles!
- Does not cover short stalls
- Less likely to slow execution of a single thread (smaller latency)
- Needs hardware support
  - PC and register file for each thread
  - Little other hardware
Fine-grained multithreading

• Two or more threads interleave instructions
  – Round-robin fashion
  – Skip stalled threads

• Needs hardware support
  – Separate PC and register file for each thread
  – Hardware to control alternating pattern

• Naturally hides delays
  – Data hazards, Cache misses
  – Pipeline runs with rare stalls

• Does not make full use of multi-issue architecture
Simultaneous Multithreading (SMT)

- Instructions from multiple threads issued on same cycle
  - Uses register renaming and dynamic scheduling facility of multi-issue architecture
- Needs more hardware support
  - Register files, PC’s for each thread
  - Temporary result registers before commit
  - Support to sort out which threads get results from which instructions
- Maximizes utilization of execution units
Conceptual Diagram

(Similar to fig 7.5 in text)
Coarse Multithreading

Stalls for A and C would be longer than indicated in previous slide
Assume long stalls at end of each thread indicated in previous slide
Fine Multithreading

----------Skip A
Simultaneous Multithreading

Skip A

Skip C

Skip A

Skip C