ECSE 425 Lecture 18: Advanced Techniques for Instruction Delivery and Speculation

H&P Chapter 2

Last Time

- Multiple-issue Processors
  - Static Superscalar
  - VLIW
  - Dynamic Superscalar

Today

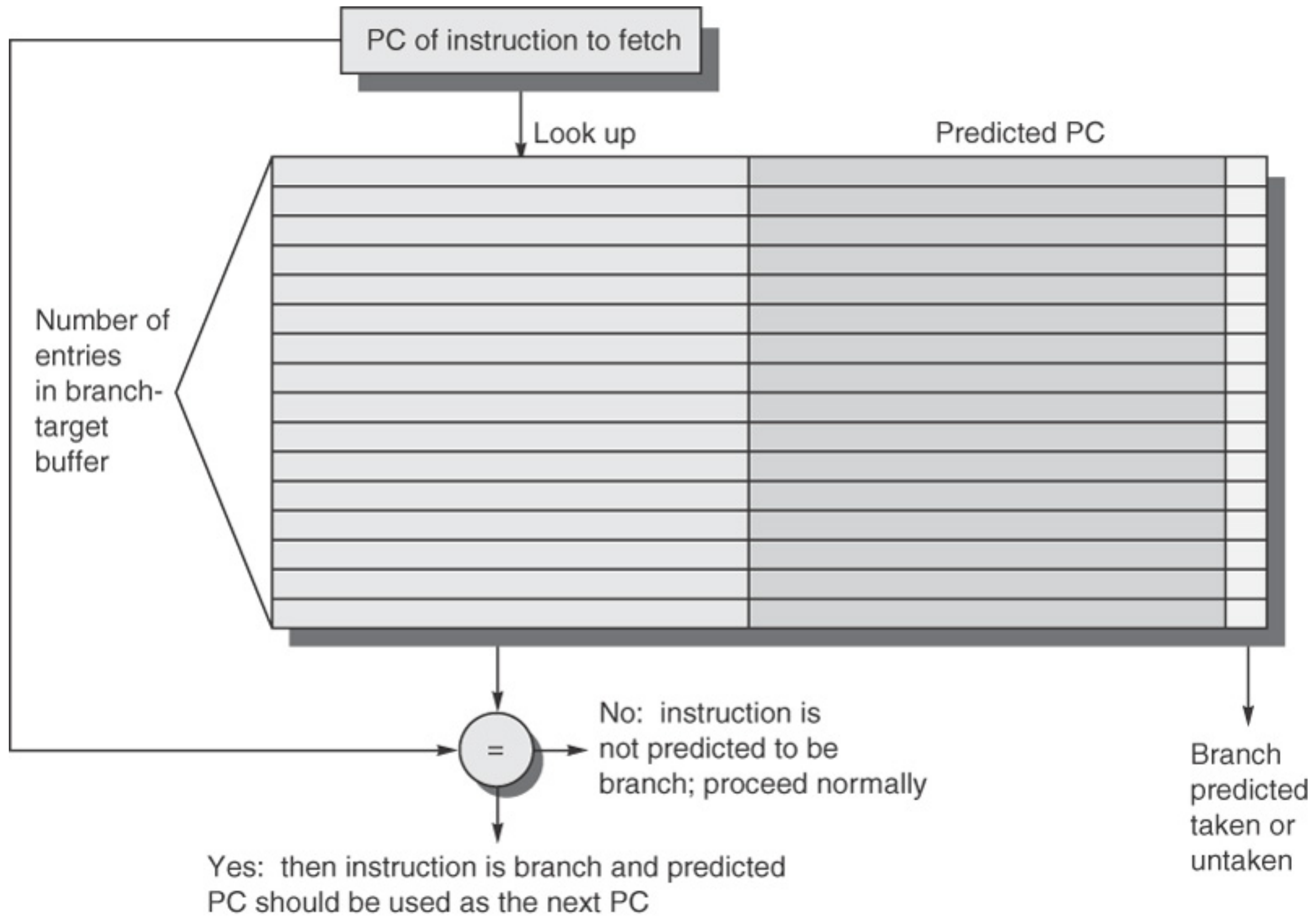
- Advanced Techniques
  - Instruction Delivery
  - Speculation
  - Chapter 2.9

High Performance Instruction Delivery

- Delivering instructions becomes a bottleneck
  - Especially in multiple-issue processors
  - Handling branches is the most difficult
  - Have to go beyond simple branch prediction
- 5-stage MIPS pipeline: branch target address and branch condition (outcome) are known in ID
  - One cycle branch delay
- Predictors don't give much benefit for multiple-issue pipeline unless they can predict in the IF stage
  - Seems impossible: don't even know the instruction yet!

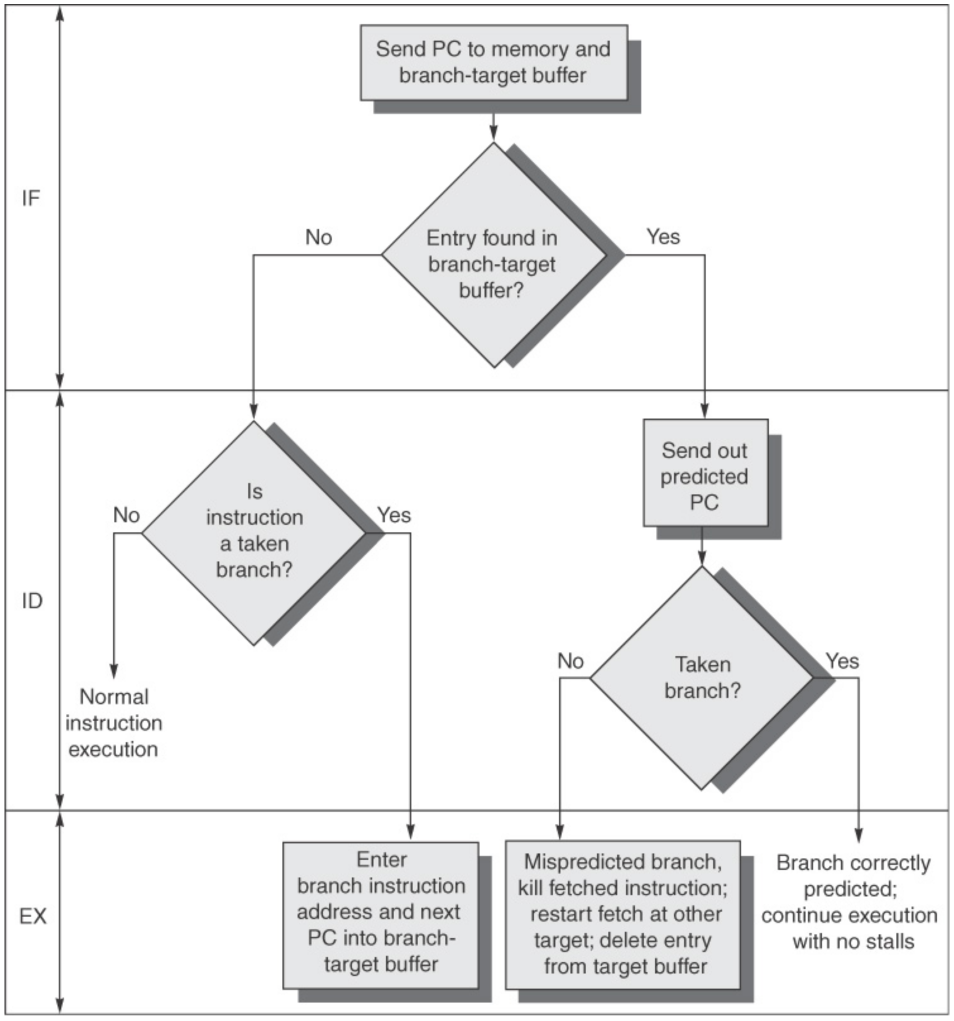
Store Targets in a Branch Target Buffer

- Table look up can be done in hardware for small tables
- Usually, only store predicted taken branches in BTB


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be branch; proceed normally
Yes: then instruction is branch and predicted PC should be used as the next PC
Branch predicted taken or untaken

Branch Target Buffer

1. Hit? Yes. Predicted taken, and taken.
   – No penalty
2. Hit? Yes. Predicted taken, and not taken.
   – 2 cycle penalty; update BTB, restart fetching
3. Hit? No. Taken.
   – 2 cycle penalty; update BTB, restart fetching
4. Hit? No. Not taken.
   – Fall-through, no penalty.


IF
ID
EX
Send PC to memory and branch-target buffer
Entry found in branch-target buffer?
No
Yes
Is instruction a taken branch?
No
Yes
Normal instruction execution
Send out predicted PC
Taken branch?
No
Yes
Enter branch instruction address and next PC into branch-target buffer
Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
Branch correctly predicted; continue execution with no stalls

© 2007 Elsevier, Inc. All rights reserved.

Don’t Store Targets, Store Instructions

- Next logical step: branch folding
  - Buffer the predicted instruction, not its address
- Works perfectly for unconditional branches
  - Eliminates them completely!
  - Replaces the jump instruction directly with the target instruction—a bonus cycle!

Return Address Predictors

- For the procedure call instruction, the return PC varies at run time
  - Consider stdlib in C, or object functions in C++/Java
- Solution: store return addresses in a small buffer
  - Save the 8-16 most recent return addresses
- On a function call, push the return address onto a stack; on a return, pop
  - If the function depth is no more than the size of the buffer, perfect return address prediction

Integrated Instruction Fetch Units

- Separate unit with separate control
  - Runs independently of the pipeline
  - Fetches instructions, predicts branches, provides instructions to the datapath on demand
- Integrated branch prediction
  - Predicts branches as instructions are fetched
- Instruction prefetch
  - Fetch runs ahead of execution
  - Hides delay of cache misses
- Instruction buffering
  - Holds instructions for pipeline, providing them on request

Explicit Renaming Register File vs. ROB

- When using an ROB, distributed architectural state; values can be stored in
  - Registers
  - Reservation stations
  - ROB – implicitly renames registers by ROB entry number
- Instead of an ROB, use a huge register file
  - More physical registers than architectural registers
  - Explicitly rename registers
  - Values only available from the register file
- Simplifies commit
  - Indicate that physical register with temporary value now represents an architectural register
  - Indicate that the physical register holding the old value of the architectural register is now free

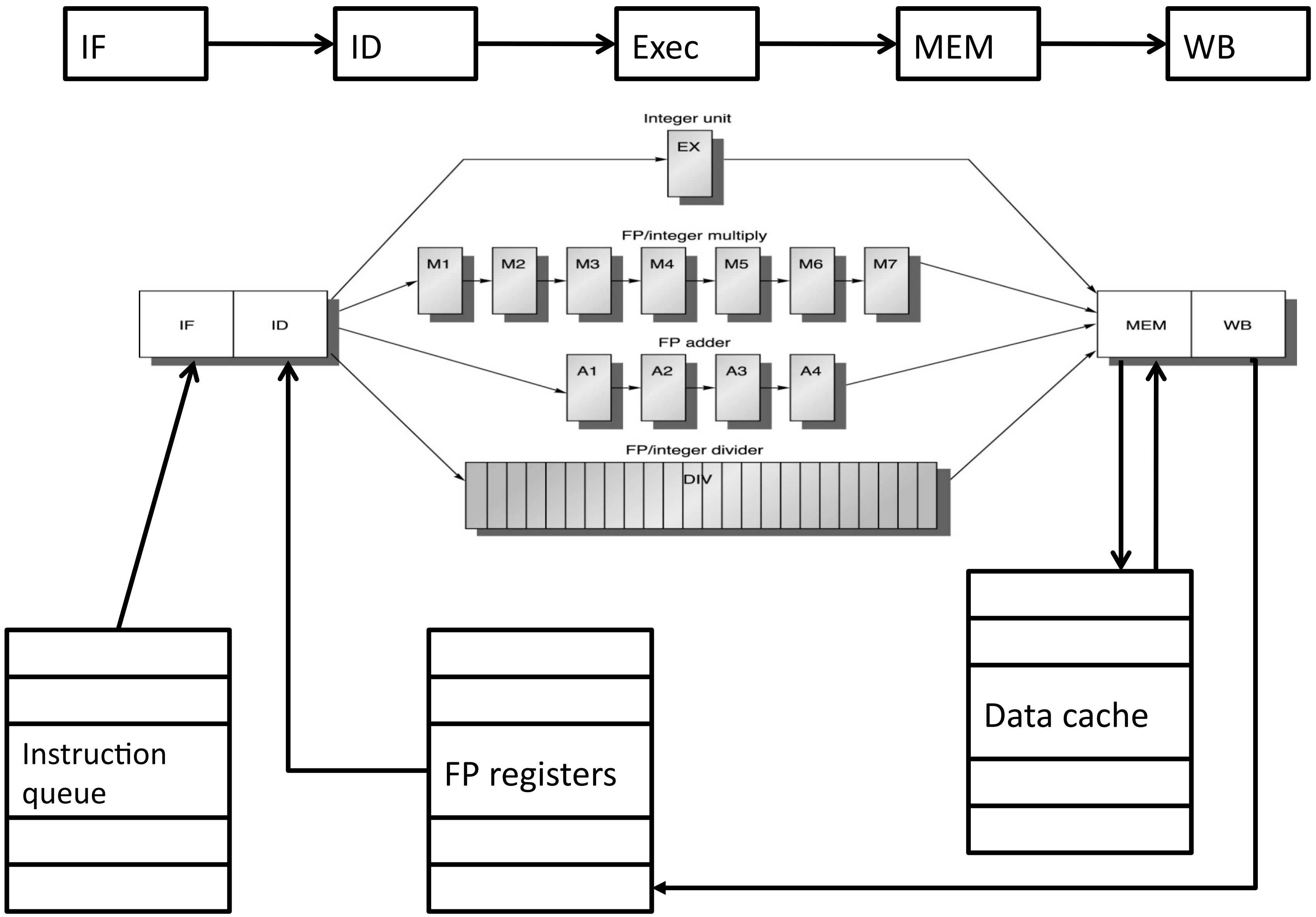
Summary

- Advanced Predicted Techniques
  - Branch Target Buffer—saves predicted PC, too
  - Branch folding—saves the instruction instead
  - Return address predictor—enables perfect prediction when returning from function calls
- Advanced Instruction Delivery
  - Integrated Instruction Fetch Units
- Advanced Speculation Techniques
  - Renaming Register File

Review

- Standard FP Pipeline
- Tomasulo's Algorithm
- Tomasulo's Algorithm with Speculation
- Putting it all together: Intel P4 vs AMD Opteron

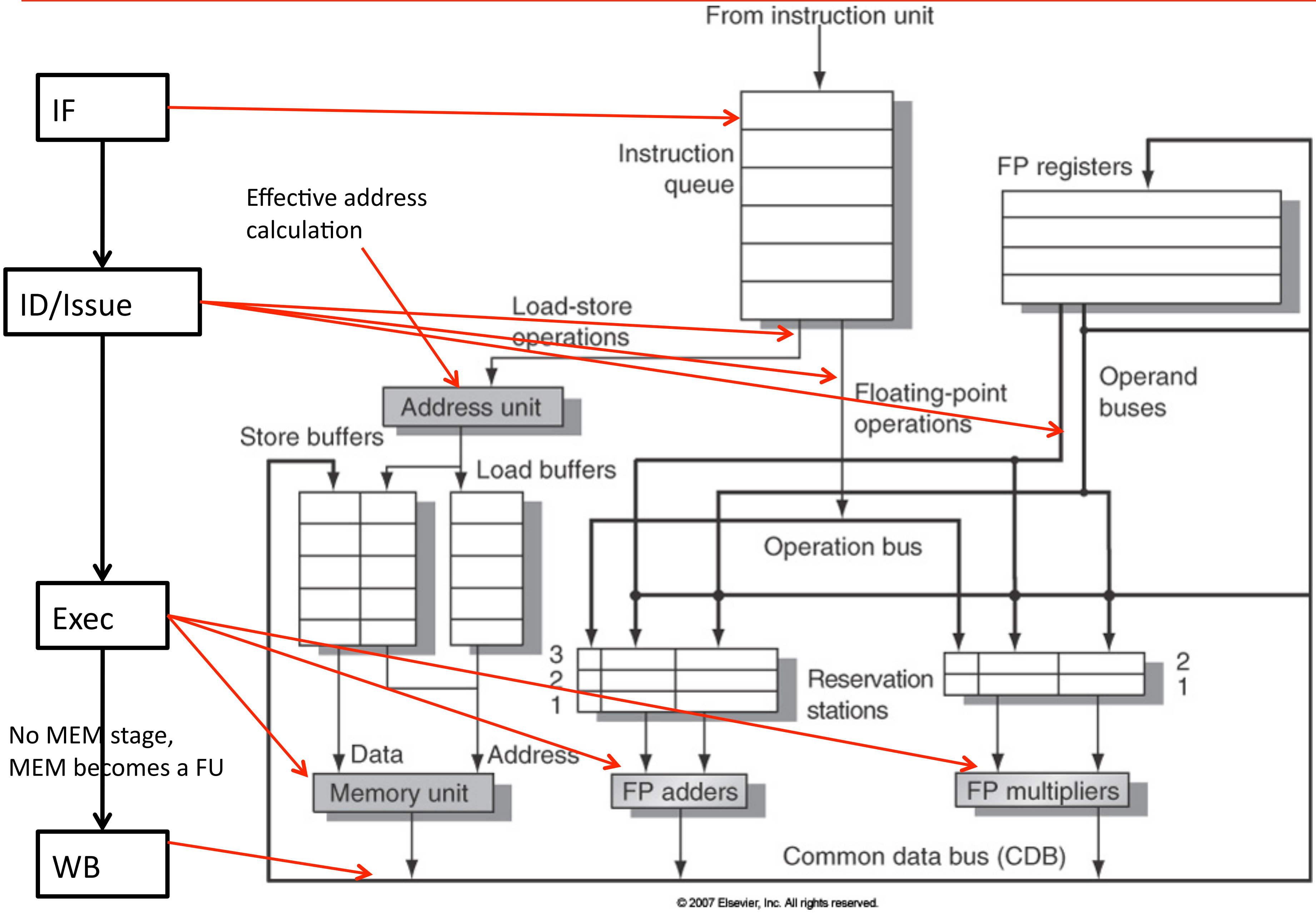
Standard FP Pipeline (horizontal)



Exposing ILP in the Standard FP Pipeline

- In-order issue, out-of-order execution and completion
- Static approaches only
  - Fetch instructions based on branch prediction
  - Loop unrolling
  - Code scheduling

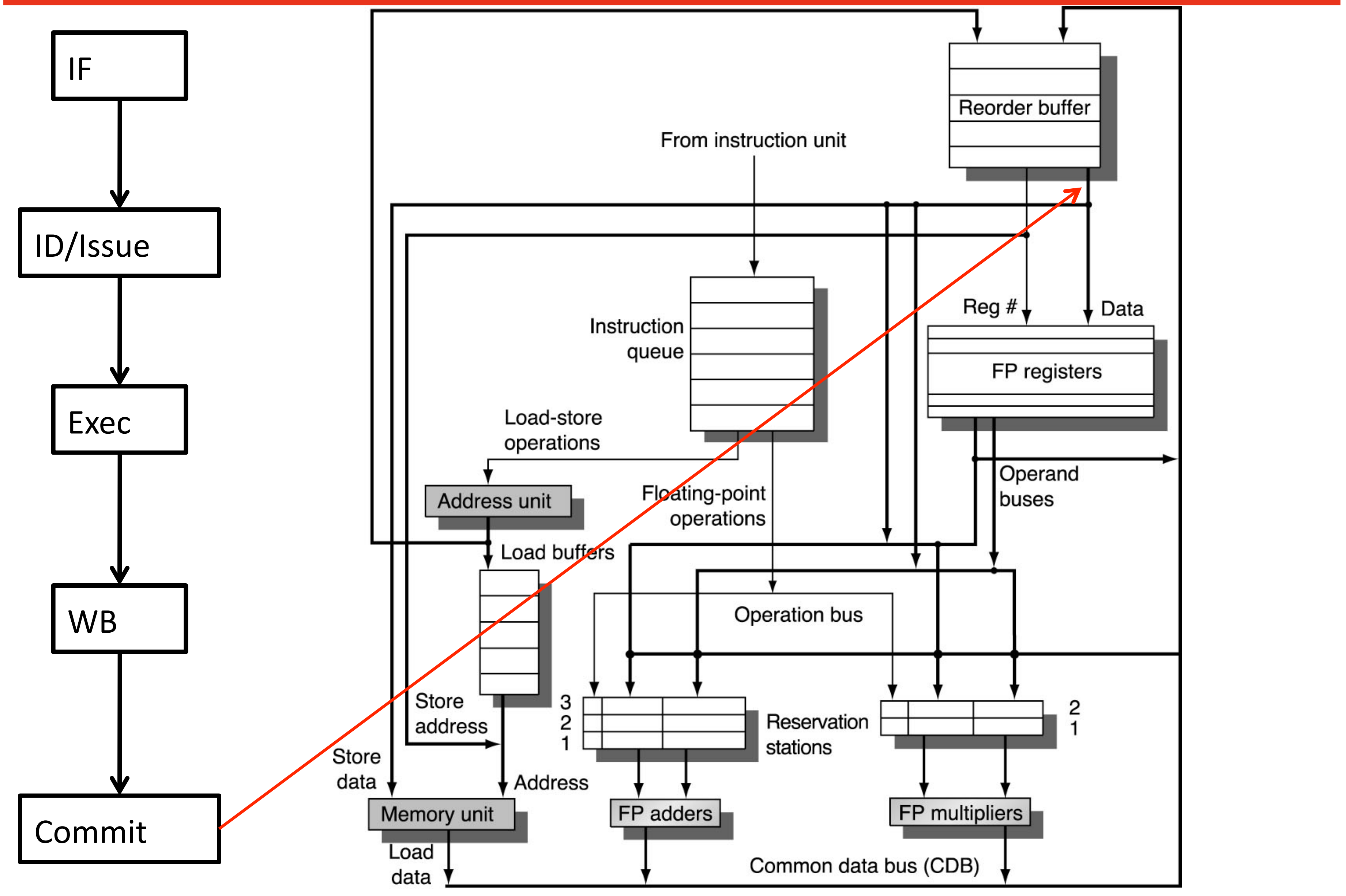
Tomasulo's Dynamic Pipeline (vertical)



Exposing ILP for Dynamic Scheduling

- In-order issue, out-of-order execution and completion
- Static Techniques can still be applied
  - E.g., loop unrolling is important for any architecture
- Dynamic scheduling removes hazards
  - Register renaming eliminates name dependencies

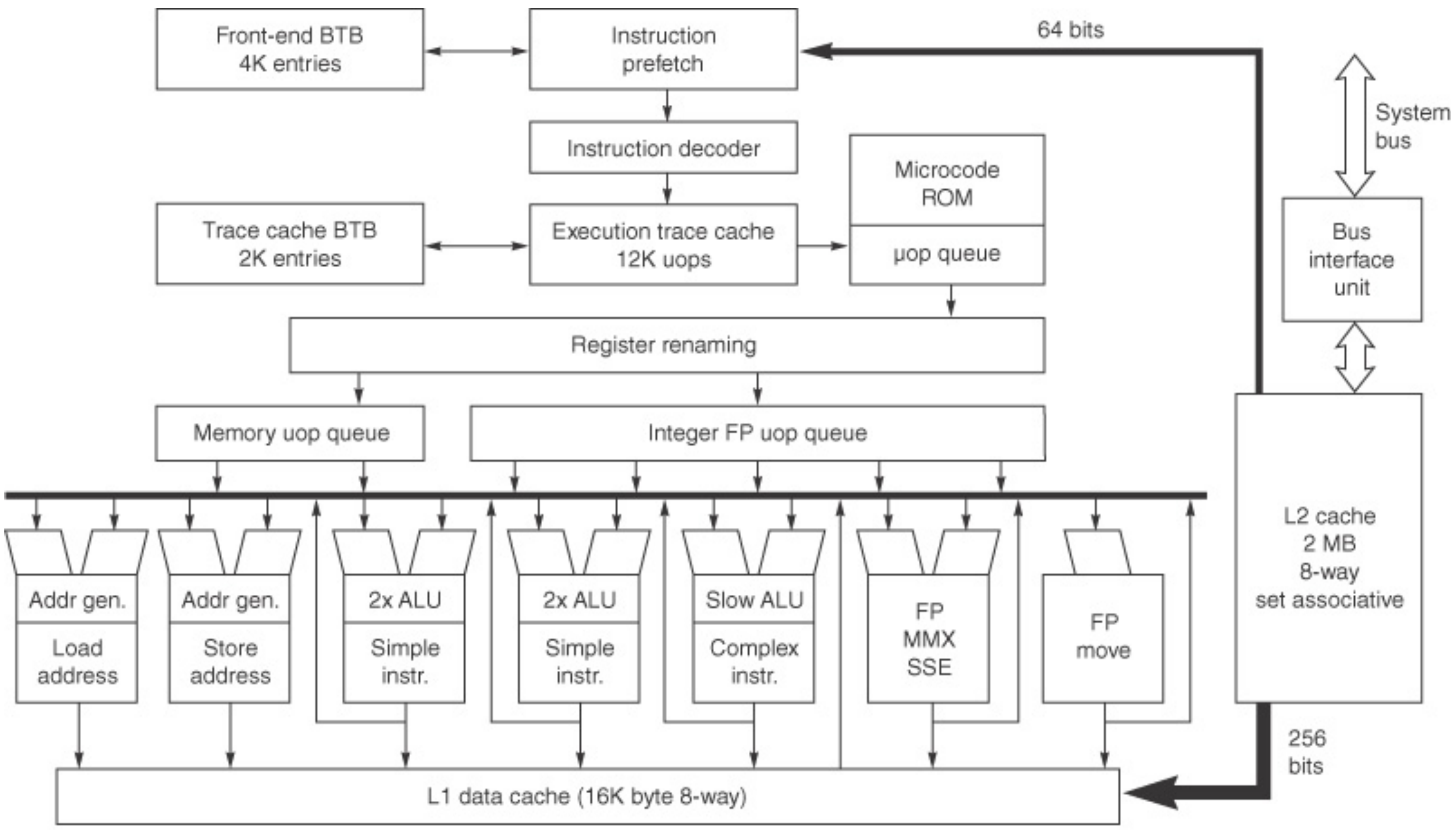
Speculative Pipeline (vertical)



Exposing ILP for Speculation

- In-order issue, out-of-order execution, in-order completion
- Speculation enables additional overlapping
  - Execute instructions based on branch prediction
  - Overlap basic blocks before branch resolution

Intel Pentium 4

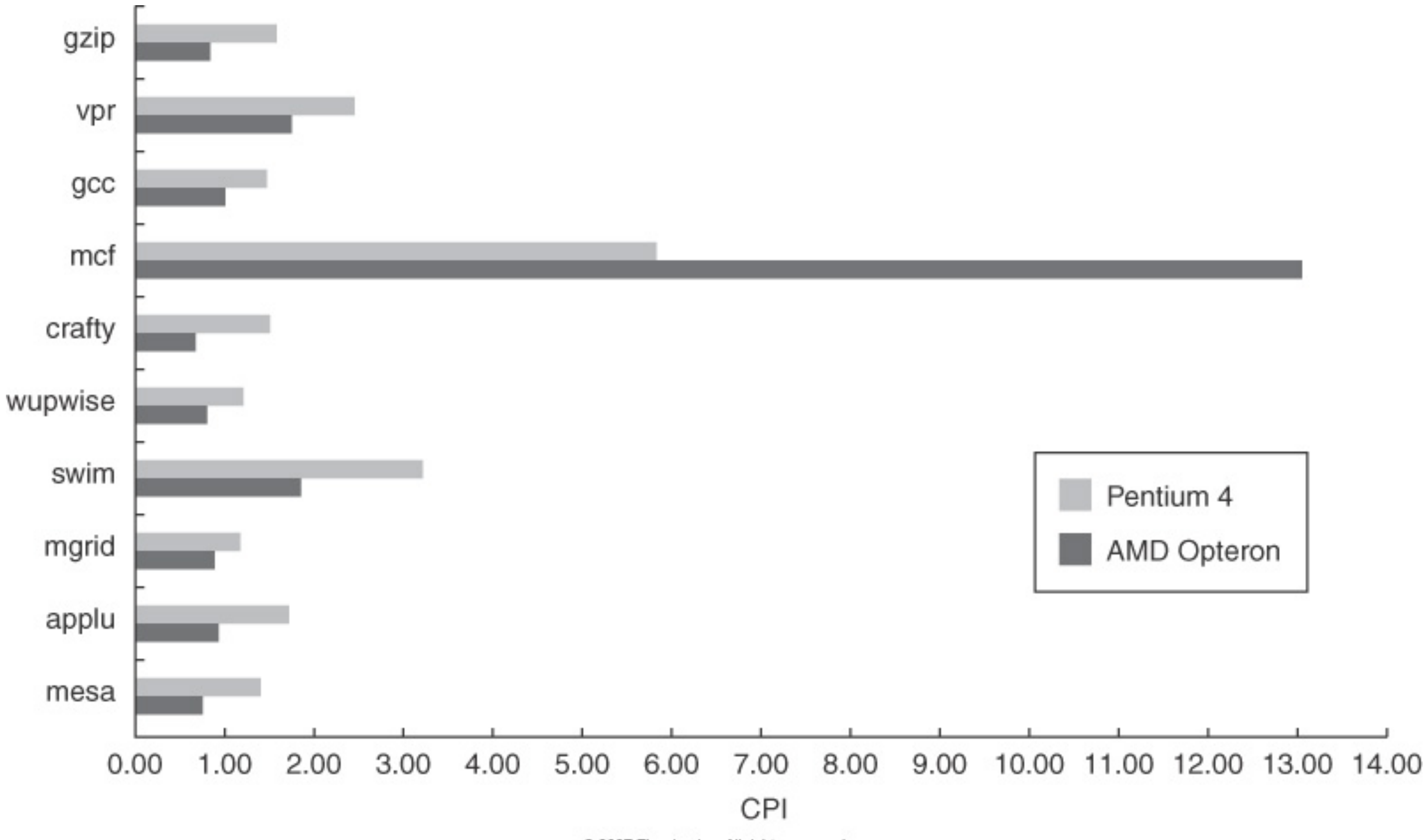


© 2007 Elsevier, Inc. All rights reserved.

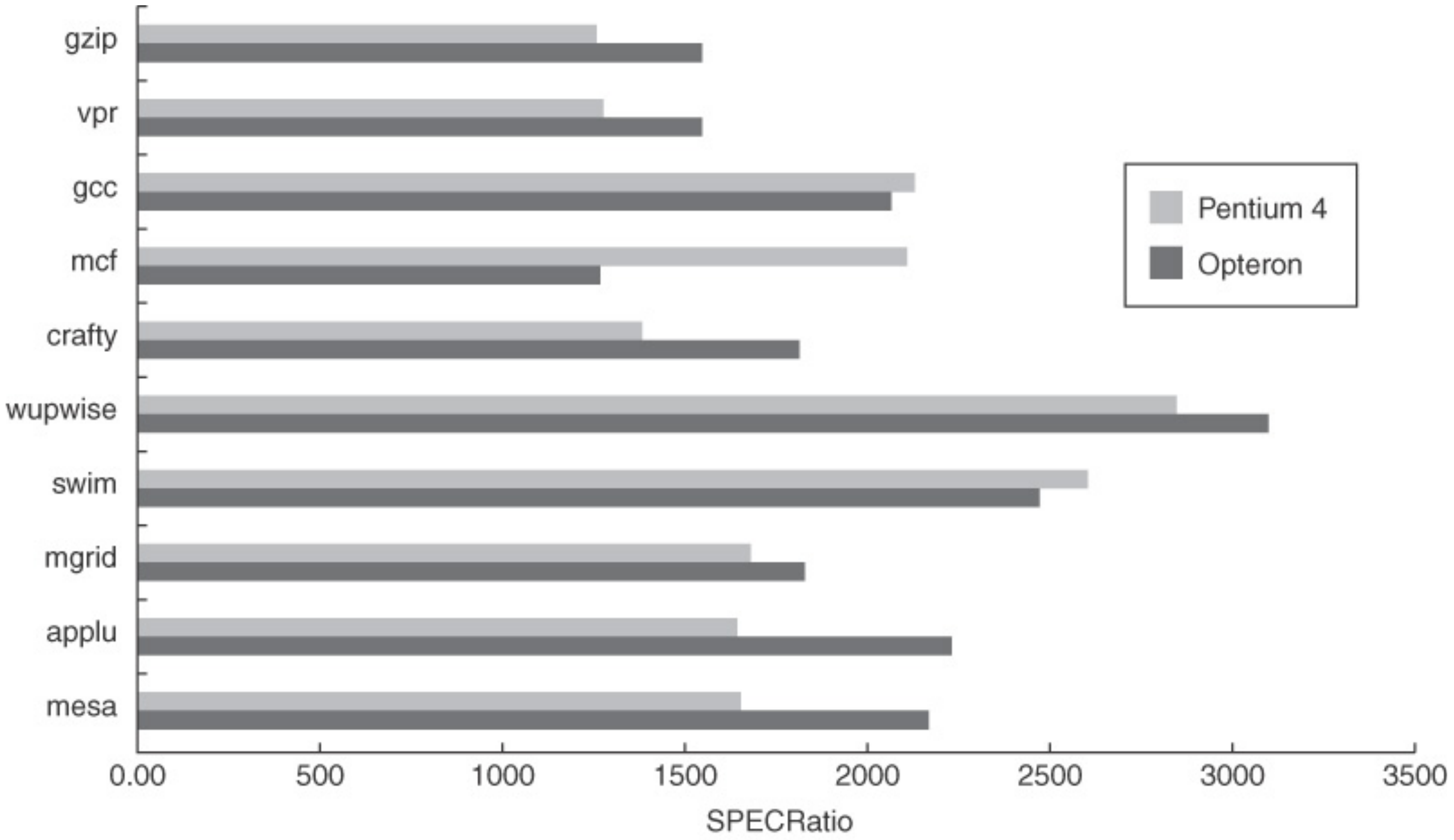
Pentium 4 Features

- Deep pipeline: 31 cycles from fetch to retire
- 4K-entry Front-end BTB
  - Predicts the next IA-32 instruction to fetch
  - Only used when trace cache misses
- 12K-uop Trace Cache
- 128-entry Trace Cache BTB
- 128 registers
  - Supporting 128 uops, including 48 loads, 32 stores
- 7 functional units
- 16 KB L1 data cache
- 2 MB L2 cache

3.2 GHz P4, 2.6 GHz Opteron: CPI



3.2 GHz P4, 2.6 GHz Opteron: SPECRatio



Next Time

- Memory Hierarchy
  - Read Appendix C.1