ECSE 425 Lecture 16: Hardware Speculation

H&P Chapter 2

Last Time

- Dynamic Scheduling (Chapter 2.4 and 2.5)
  - In-order issue
  - Out-of-order execution
  - Out-of-order completion

Today

- Limitations of Dynamic Scheduling
  - Limited overlapping of basic blocks
  - Imprecise exceptions
- Hardware Speculation (Chapter 2.6)
  - In-order issue
  - Out-of-order execution
  - *In-order completion*

Dynamic Scheduling

- In-order
  - Fetch of instructions
  - Issue of instructions to reservation stations
- Out-of-order
  - Dispatch to execution units
  - Write-back
- When a branch is encountered
  - Make a prediction
  - Fetch and issue instructions
  - Don’t dispatch until branch is resolved

Limitations of Dynamic Scheduling

- During branch resolution, can fetch and issue instructions, but can't execute them
- If we allow instructions to execute, we risk
  – Modifying processor state with instructions that should not execute (violating data flow)
  – Raising exceptions that would not be encountered (violating exception behavior)
- So predict branches, but verify before continuing
- Branch prediction exposes some ILP, hides some latency, but we can do better!

Dynamic Scheduling with Speculation

- Predict branches
  - Often, make a series of predictions
- *Assume* the predictions are correct
  - And allow instructions to *speculatively* execute
  - Use speculative results to allow further speculation
- Misprediction?
  - Identify instructions that shouldn't have executed
  - Preserve data flow and exception behavior by undoing their execution

Requirements of Hardware Speculation

- Preserve data flow
  - Violation means the program gets the wrong result
  - Prevent state update from until branches are resolved
- Preserve exception behavior
  - Violation means we raise exceptions that wouldn't otherwise occur
  - Prevent exceptions until branches are resolved
- Bonus: precise exceptions
  - If additionally exceptions aren't raised until the proper time, they are *precise*!

Requirements, Continued

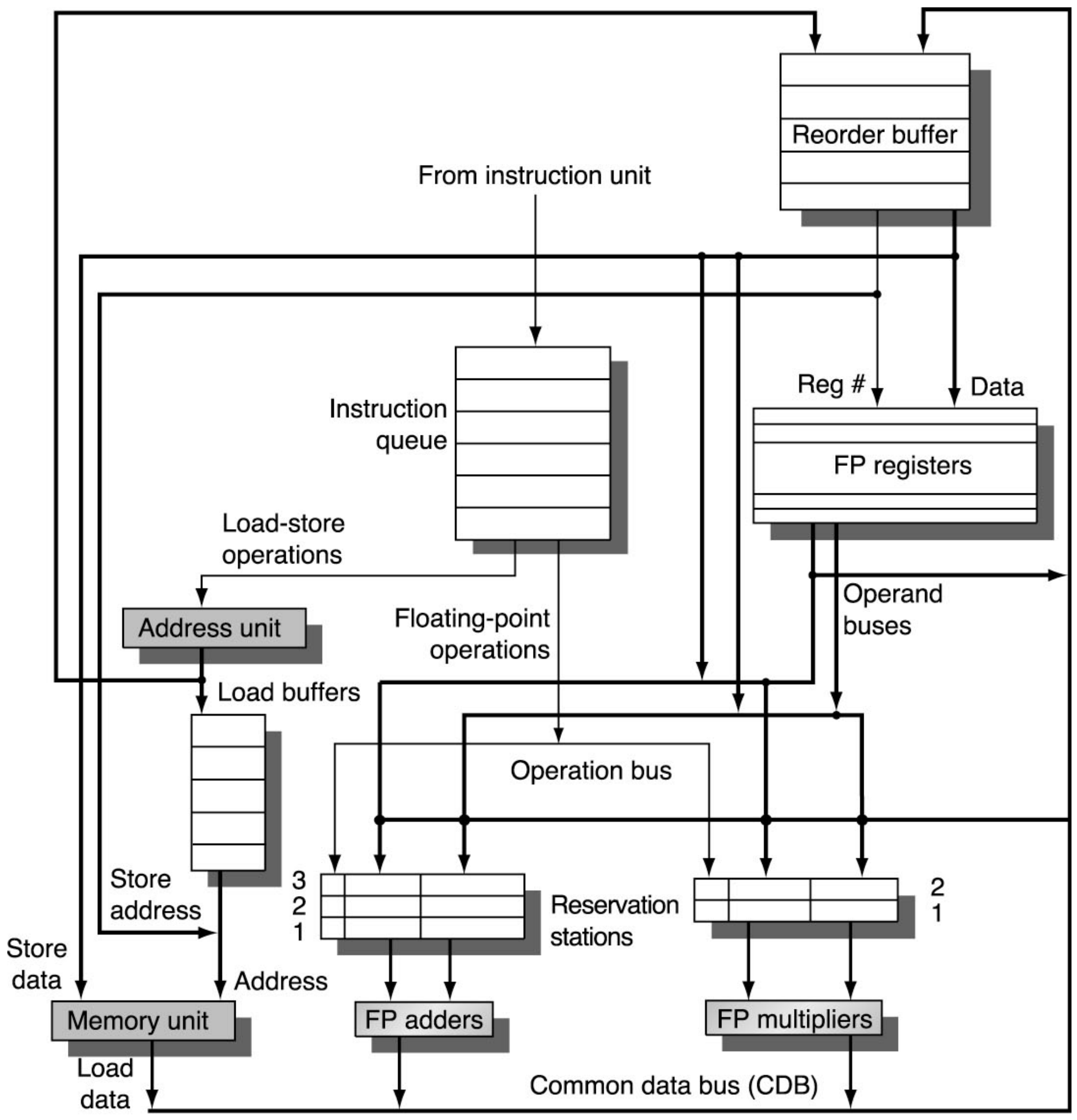
- We need to be able to isolate speculative state from *committed* state (which can't be undone)
  - Only commit state changes when we know they definitely occur
- We want the results of speculative execution to be available for further speculation
  - To expose as much ILP as possible
- Solution: a new stage, and a new structure
  - Speculative instructions wait to be *committed, in-order* in the *re-order buffer*, which bypasses the RF

New Stage: Instruction Commit

- Execute *out-of-order* but commit *in-order*
  - Prevents any irrecoverable action (state update, or exception) until branches are resolved
- When a branch is resolved, dependent instructions are no longer speculative
  - Correct prediction? Instructions can write regs/mem
  - Misprediction? Flush instructions, re-start instruction fetch at the correct target instruction
- Instructions may finish execution considerably before they are ready to commit
- Commit when the result is ready, and all earlier instructions have committed

New Structure: Re-order Buffer (ROB)

- Re-order buffer holds uncommitted results
  - CBD writes to RS and ROB, not RF
  - RF is updated only when the instruction commits
  - ROB also replaces the store buffers
  - Memory is updated only when stores commit
- The ROB forwards to speculative instructions
  - Takes over the role of register renaming from the reservation stations (RS)
- RS still buffers instr. between issue and execution



© 2003 Elsevier Science (USA). All rights reserved.

Speculative Tomasulo Algorithm

1. Issue—get instruction from Op Queue
   - Checks for structural hazards
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination
2. Execution—operate on operands (EX)
   - Checks for data hazards
   - When both operands are ready, execute
   - Not ready? Watch CDB for result
3. Write result—finish execution (WB)
   - Write to CDB, to all waiting FUs & reorder buffer
   - Release the reservation station
4. Commit—update register with reorder result
   - When instr. at head of reorder buffer & result present, update RF (or store to memory) and release reorder buffer entry
   - Mispredicted branch flush reorder buffer

Reorder Buffer

ROB

Entry	Busy	Instruction	State	Dest	Value
1	no	L.D. F6,34(R2)	Commit	F6	Mem[34+Regs[R2]]
2	yes	MUL.D F0,F6,F4	Write result	F0	#1 x Regs[F4]
3	yes	DIV.D F10,F0,F6	Execute	F10	

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Mult1	no	MUL.D	Mem[34+Regs[R2]]	Regs[F4]			#2	
Mult2	yes	DIV.D		Mem[34+Regs[R2]]	#2		#3	

FP Register Status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Reorder #	2										3
Busy	yes	no	no	no	no	no	no	no	no	no	yes

What about Precise Interrupts?

- Tomasulo's Algorithm
  - In-order issue
  - Out-of-order execution
  - Out-of-order completion
  - Imprecise exceptions
- ROB gives us a mechanism for providing precise exceptions

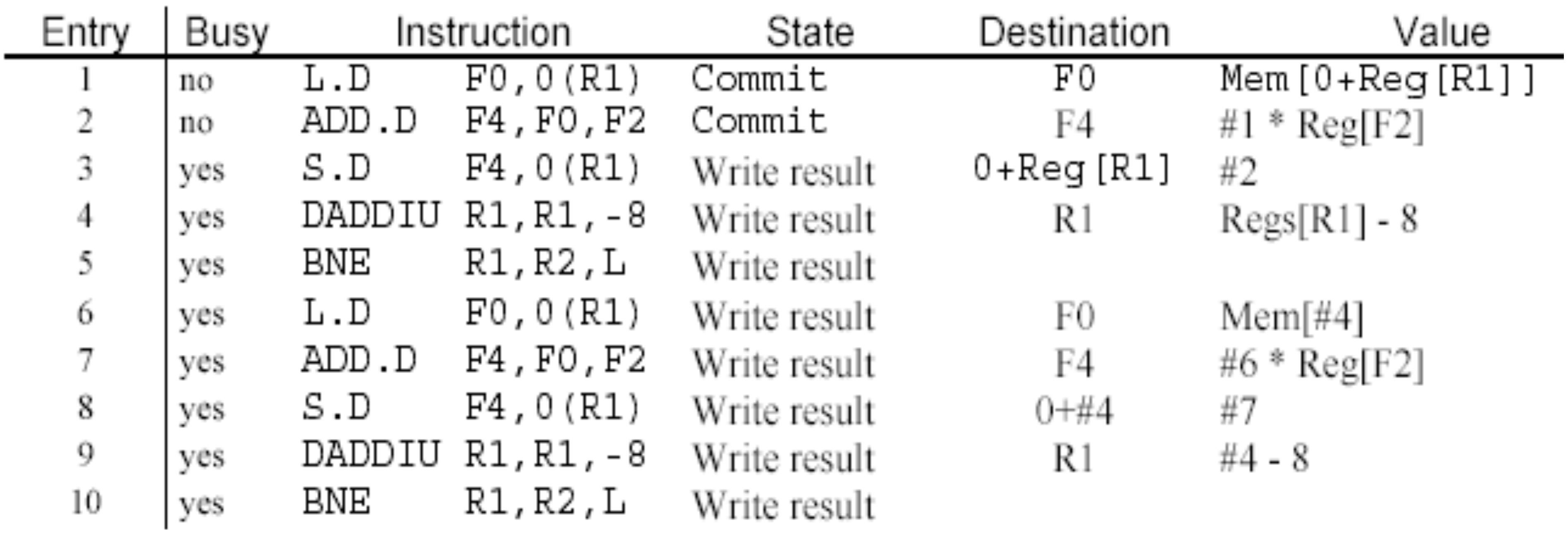
When Speculation is Wrong

- HW speculation guesses which branch to take
- Guess right?
  - Commit the instructions that follow the branch
  - Until the next speculative branch is encountered
- Guess wrong?
  - Don't commit the instructions that follow—
- Instead, free all later ROB entries
  - And then re-start execution from correct branch
- What does this mean for exceptions?

Precise Exceptions and Speculation

- When an instruction requires exception handling:
  - Modify a status register in the ROB entry
  - Wait until the instruction is to commit to give control to the exception handler
- Unspeculative instruction?
  - Exception is raised at commit and only earlier instructions have committed: precise exception
- Speculative instruction?
  - Instruction never commits, exception is flushed with the instruction: correct exception behavior

Add-scalar-to-vector example



Entry	Busy	Instruction		State	Destination	Value
1	no	L.D	F0,0(R1)	Commit	F0	Mem[0+Reg[R1]]
2	no	ADD.D	F4,F0,F2	Commit	F4	#1 * Reg[F2]
3	yes	S.D	F4,0(R1)	Write result	0+Reg[R1]	#2
4	yes	DADDIU	R1,R1,-8	Write result	R1	Regs[R1] - 8
5	yes	BNE	R1,R2,L	Write result		
6	yes	L.D	F0,0(R1)	Write result	F0	Mem[#4]
7	yes	ADD.D	F4,F0,F2	Write result	F4	#6 * Reg[F2]
8	yes	S.D	F4,0(R1)	Write result	0+#4	#7
9	yes	DADDIU	R1,R1,-8	Write result	R1	#4 - 8
10	yes	BNE	R1,R2,L	Write result		

- Two complete loops issued
  - First two instructions have committed, freeing ROB
- If BNE is mispredicted, following instructions never commit
- In essence, ROB executes in-order a simplified version of original codes
  - At this point, all results are ready
  - Actual computation was done speculatively

Summary

- Limitations of Dynamic Scheduling
  - Limited overlapping of adjacent basic blocks
  - Imprecise exceptions
- Dynamic Scheduling with Hardware Speculation
  - Not only predict branches, assume correct prediction
- New pipeline stage, new hardware structure
  - ROB: takes over renaming, holds results until it is safe to modify processor state
  - Instruction commit: results are committed in order, but forwarded to speculative instructions
- Speculation ⇒ greater ILP, and precise exceptions
  - CPI approaching the ideal, 1!

Next Time

- Multiple-Issue Processors
  - Chapters 2.7 and 2.8