ECSE 425 Lecture 8: Branch Hazards; Pipeline Implementation

H&P Appendix A

Last Time

- Limits of pipeline performance
- Pipeline hazards
- Hazard mitigation

Today

- Hazards
  - Branch Hazards
- Implementing Pipelining
  - Pipeline Control
  - Managing Branches

Branch Hazards

- Caused by branch instructions
  - Now the consuming stage is always IF
  - Can cause greater performance loss than data hazards
- If branch is taken
  - The value of the PC is determined later in the pipeline
- If the branch is not taken
  - PC is the next value (fall-through instruction)
- Performance losses due to branches
  - We don't determine an instruction is a branch until ID
  - We don't determine where the branch goes until EX

Reducing Branch Penalties

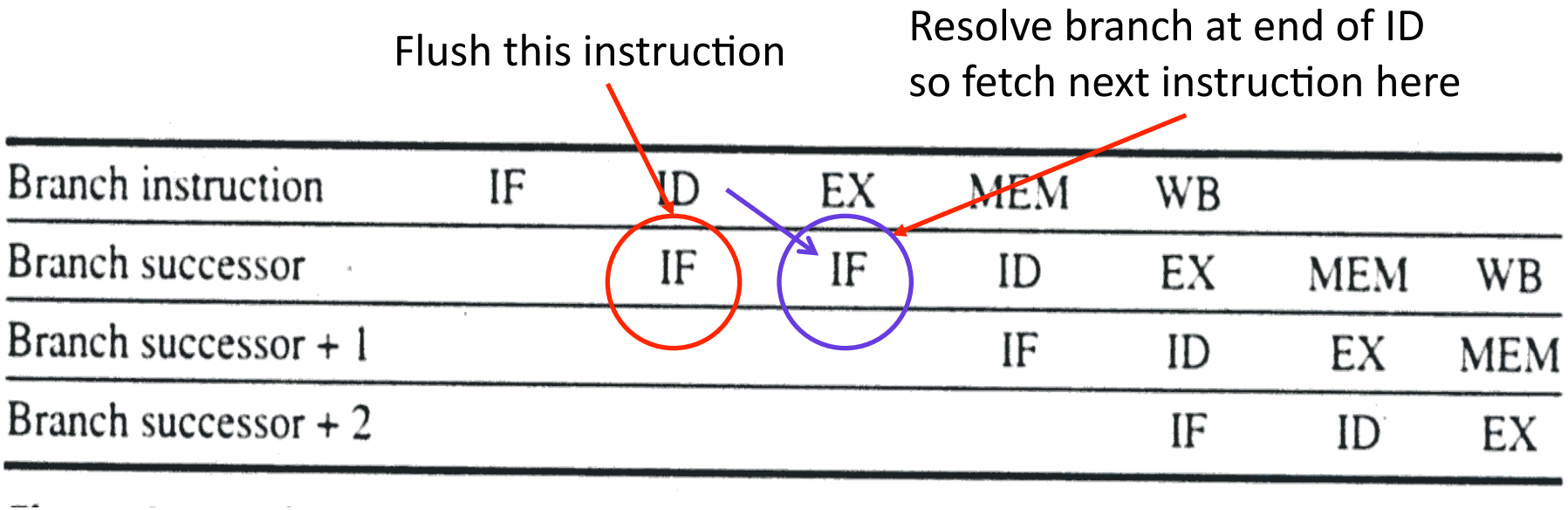
- Waiting until EX to determine branch target
  - Two cycle delay imposed by each branch instruction!
  - What is the resulting performance loss?
- Assuming an ideal CPI of 1, we can measure speedup with branch penalties as:

$$SpeedUpPipe = \frac{PipeDepth}{1 + BranchFrequency \times BranchPenalty}$$

- To reduce the penalty
  - Move branch resolution to ID stage
  - Branch prediction

Flush (or freeze) the pipeline

- Once branch is detected and resolved in ID, re-fetch destination instructions
- Fixed branch penalty (1 cycle per branch)
  - What is the resulting performance loss?



Predicted-not-taken

- Only slightly more complicated than flushing
- Proceed as if branch is not taken
  - Re-fetch instruction only if the branch is taken
- No state changes until branch outcome is known

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

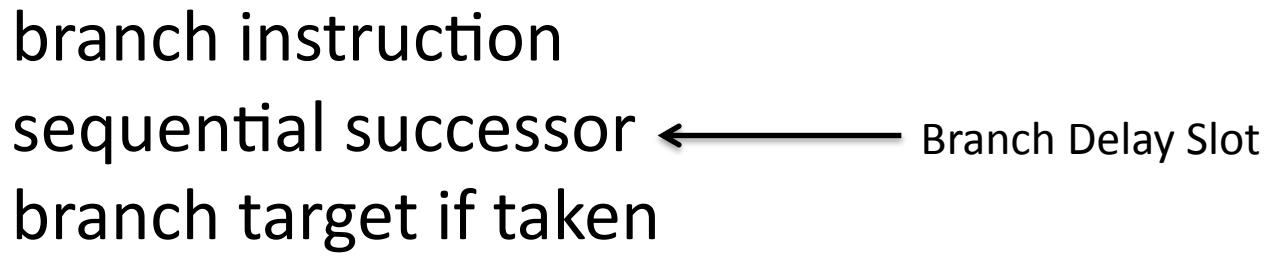
Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	**idle**	**idle**	**idle**	**idle**			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Predicted-taken

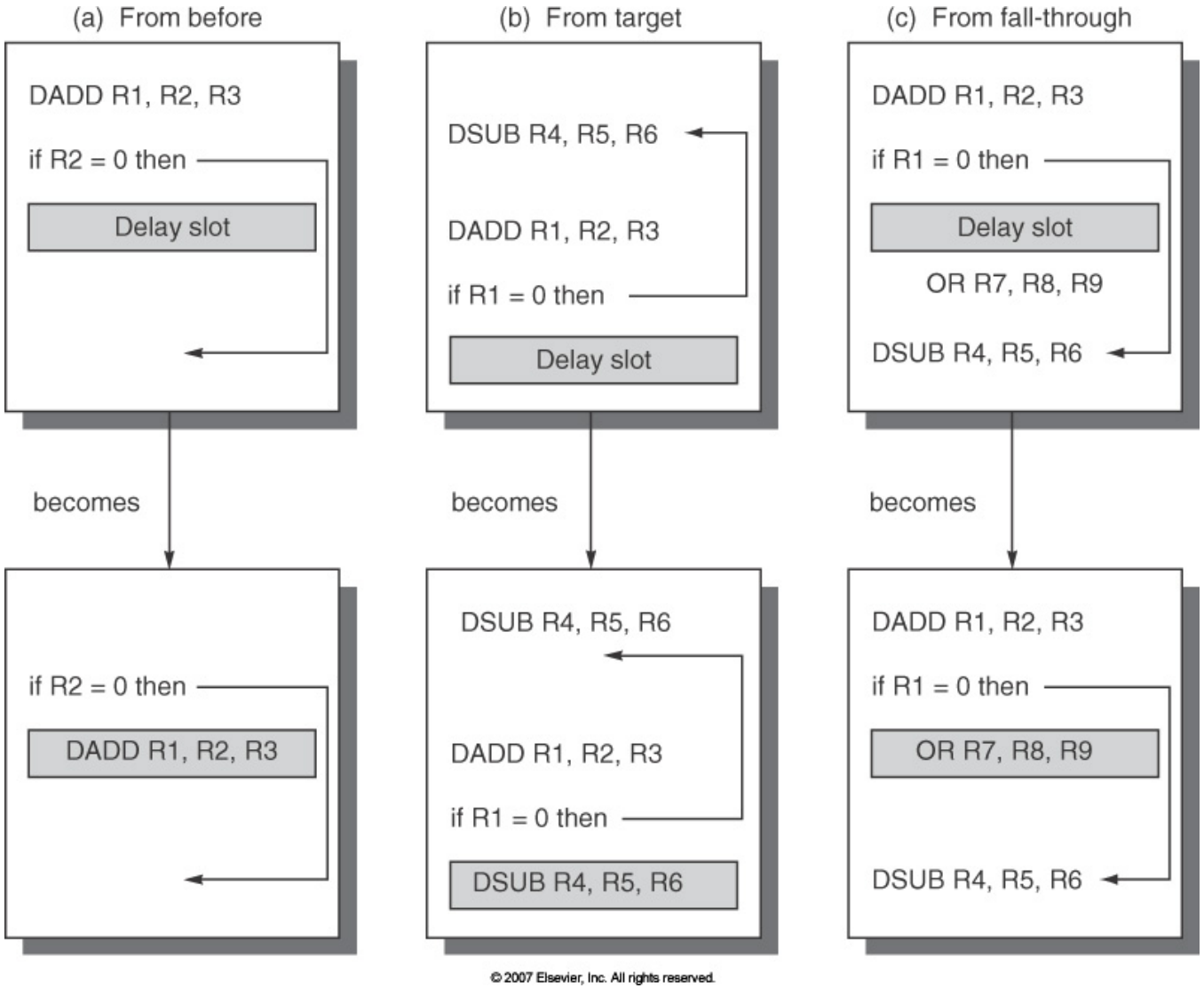
- Fetch and execute target instruction as soon as branch is decoded and target address is known
  - In processors with complex branch conditions, branch target may be known before branch outcome
- Compiler profiling can help
  - Make the common case fast
  - Organize code to so the most frequent execution path benefits from hardware branch prediction schemes

Delayed branch

- Allow one or more instructions following the branch to execute even if the branch is taken
- Rely on the compiler to schedule code
  - Compiler finds instructions to fill in after branch
  - In a five-stage pipeline, find one independent instr.



Scheduling Delayed Branches



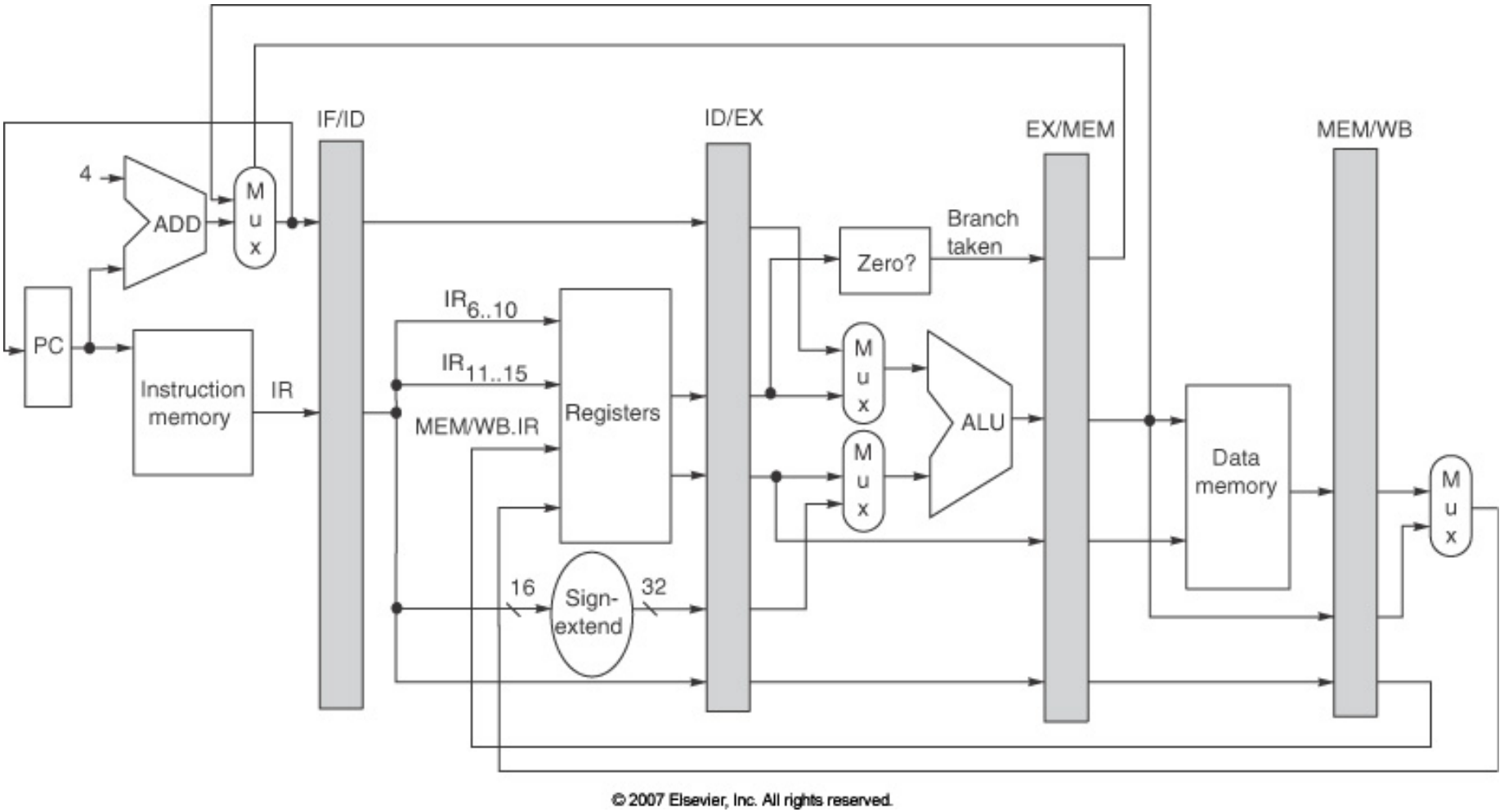
Branch prediction performance

- Example: MIPS R4000, an 8-stage integer pipeline
  - 3 pipeline stages before branch-target address known
  - 1 additional cycle before branch condition is evaluated (if there is no stall waiting on the register for comparison)

Scheme	Penalties		
	Unconditional	Untaken	Taken
Flush	2	3	3
Predicted-taken	2	3	2
Predicted-untaken	2	0	3

- Find the CPI degradation due to branches assuming
  - Unconditional branch 4%
  - Conditional branch, untaken 6%
  - Conditional branch, taken 10%

Basic MIPS pipeline implementation

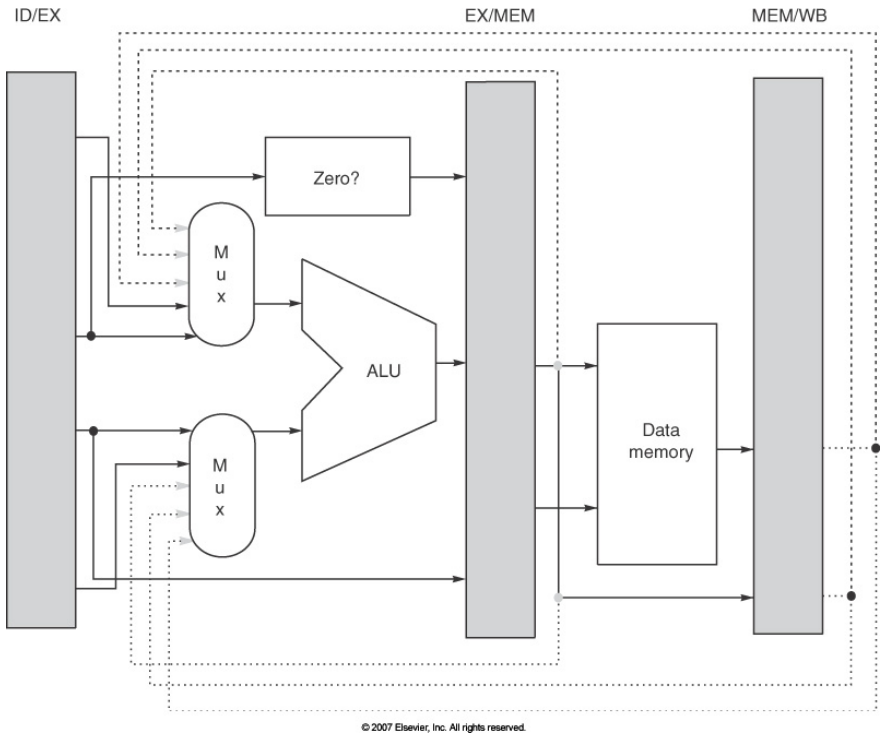


Implementing Hazard Detection

- Detect data hazards during ID
  - Stall the instruction before it is issued
  - Insert pipeline bubbles (no-ops) by changing control fields to 0s (DADD R0, R0, R0)
- Early detection of interlocks (e.g., due to a load) reduces complexity
- Detect load interlock by comparing:
  - Source registers in IF/ID (consumers) with
  - Destination register in ID/EX (producer)

Implementing Forwarding

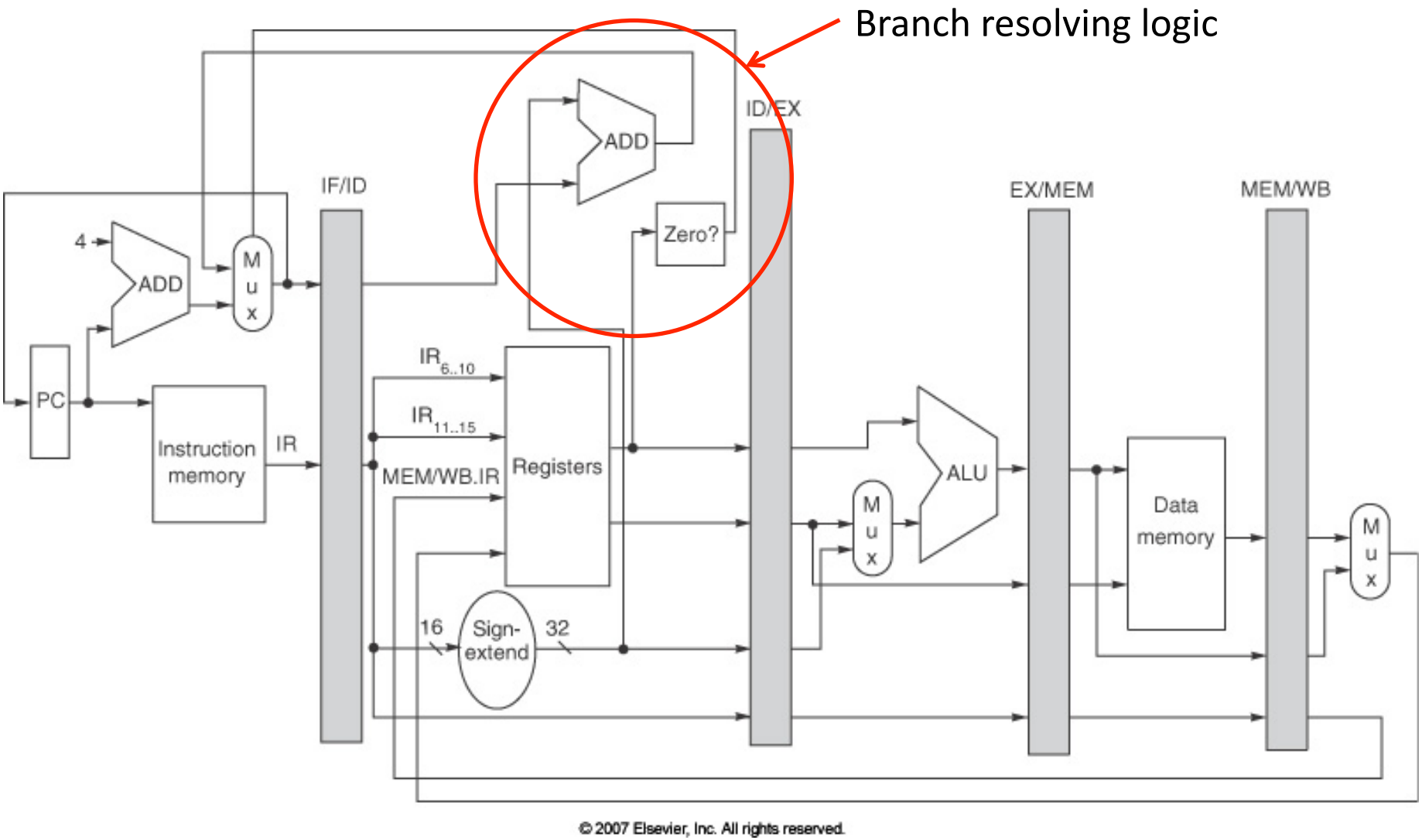
- Determine forwarding at the start of EX, MEM stages
- Compare
  - Destination registers in EX/MEM and MEM/WB with
  - Source registers in ID/EX and EX/MEM
- Forwarding
  - from EX/MEM, MEM/WB
  - to ALU, data memory, zero detection units
- Additional logic is needed to select among inputs



Branches in the Pipeline

- Resolve branches during ID
- Consider the cases BEQZ and BNEZ
  - Move zero test to ID cycle
  - Compute the branch-target address during ID (adder)
  - One clock-cycle stall on branches (instead of two!)
- Doesn't work for other branches
  - BEQ and BNE test two registers
  - Branch requires more cycles in these cases

Dealing with branches in the pipeline



Summary

- Branch Hazards
  - Branch resolution stalls the pipeline
- Mitigate branch delays by
  - Resolving during ID—requires additional hardware
  - Branch prediction
- Implementing Pipeline Control
  - Detect hazards by comparing register fields in pipeline registers
  - Forwarding requires additional hardware

Next Time

- Exceptions
- Multi-cycle operations
- Superpipelining