Advance Computer Architecture and x86 ISA

University of Science and Technology of Hanoi

MS. LE Nhu Chu Hiep

Pipeline Hazard

Pipeline Hazard

- A situations that prevent the next instruction in the instruction stream from executing during its designed clock cycle
- Hazards in pipeline can make it necessary to stall the pipeline
  - Some instructions are allowed to proceed while others are delayed
  - **Cycle loss**
- Reducing the performance of the ideal speedup gained by pipelining
- There are three classes of hazards
  - Structural hazard
  - Data hazard
  - Control hazard

Hazard types

- **Structural hazards**
  - Caused from the resource conflicts when the hardware can not support all instructions simultaneously
- **Data hazards**
  - An instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline
- **Control hazards**
  - The pipelining of branches and other instructions that change the PC

Pipeline performance with Stalls

- Speedup equation from pipelining

$$\begin{aligned}\text{Speedup from pipelining} &= \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}} \\ &= \frac{\text{CPI unpipelined} \times \text{Clock cycle unpipelined}}{\text{CPI pipelined} \times \text{Clock cycle pipelined}} \\ &= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}\end{aligned}$$

- Speedup equation encounting the stall

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}}$$

Structural Hazards

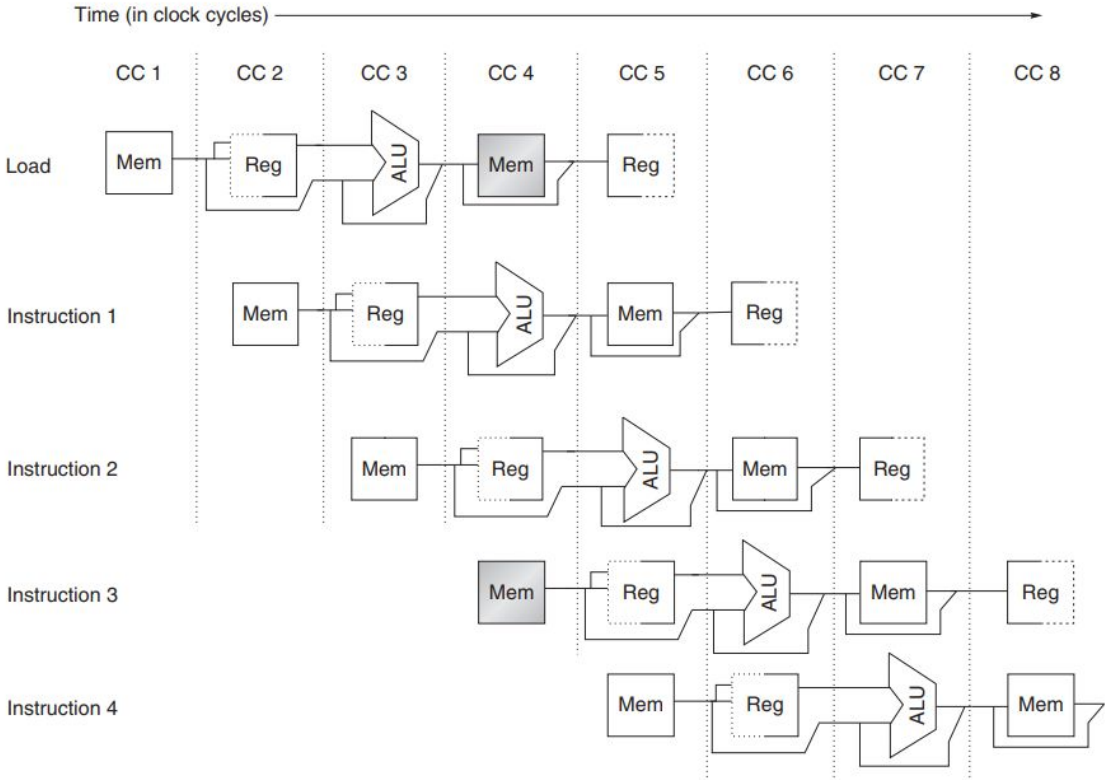
Structural Hazards

- A structural hazard: pipeline cannot be accommodated because of resource conflicts
  - Not fully pipelined functional unit => sequence of instructions cannot proceed at the rate of one per clock cycle
  - Not enough duplicated resources => not allow all combinations of instructions in the pipeline to execute
- Examples:
  - A processor have only one register-file write port but the pipeline need two writes in a clock cycle
  - The pipeline will stall one of the instructions until the required unit is available
  - The stalls increase the CPI from its ideal value of 1

Structural hazard example

- Pipelined processors
  - Shared a single-memory pipeline for data and instructions
  - When an instruction contains a data memory reference
  - And the processor try to fetch a later instruction
  - There are at least two memory access requested simultaneously => conflict !!!
- Resolve hazard
  - When the data memory access occurs
  - Stall the pipeline for 1 clock cycle
  - A pipeline **bubble** - taking space but carrying no useful work

Structural hazard example (cont.)



Structural hazard example (cont.)

	Clock cycle number									
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				Stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Structural hazard (Quest.)

- How much might the load structural hazard cost ?
  - The data references constitute 40% of the mix
  - The ideal CPI of the pipelined processor is 1
  - The clock rate with structural hazard = 1.05 * clock rate without the hazard

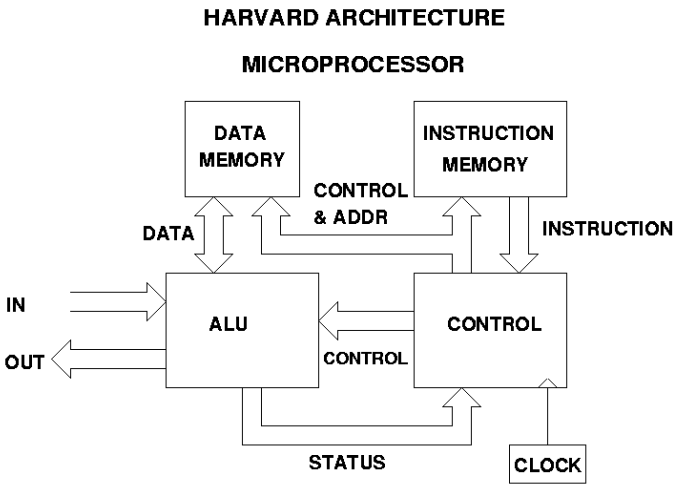
Structural hazard (Answer.)

- How much might the load structural hazard cost ?
  - The data references constitute 40% of the mix
  - The ideal CPI of the pipelined processor is 1
  - The clock rate with structural hazard = 1.05 * clock rate without the hazard
- The simplest method is to compare average instruction time with and without hazard:

$$\begin{aligned}\text{Average instruction time} &= \text{CPI} \times \text{Clock cycle time} \\ &= (1 + 0.4 \times 1) \times \frac{\text{Clock cycle time}_{\text{ideal}}}{1.05} \\ &= 1.3 \times \text{Clock cycle time}_{\text{ideal}}\end{aligned}$$

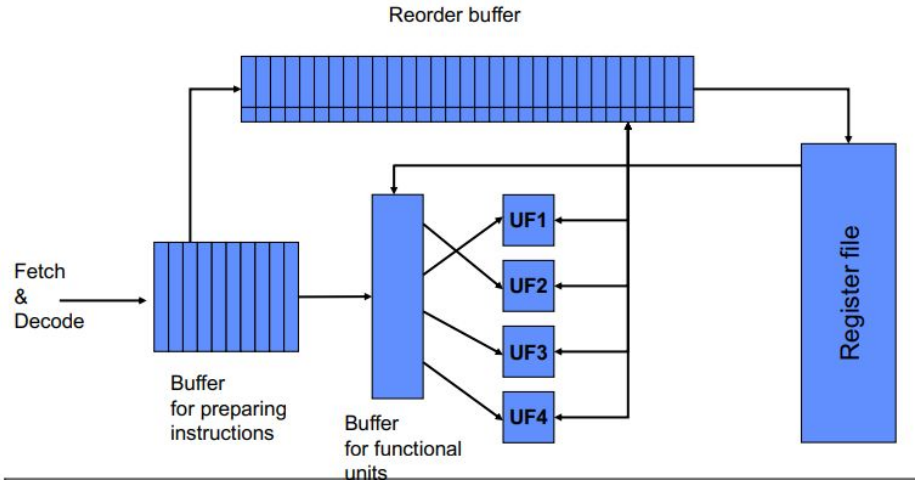
Structural hazard alternative solution

- Harvard model
  - A memory access for data
  - A memory access for instruction
  - No conflict between data reference instruction and next instruction fetching



Structural hazard alternative solution

- Applying set of buffers
  - Reorder buffer
  - **Instruction buffer**
  - Functional buffer

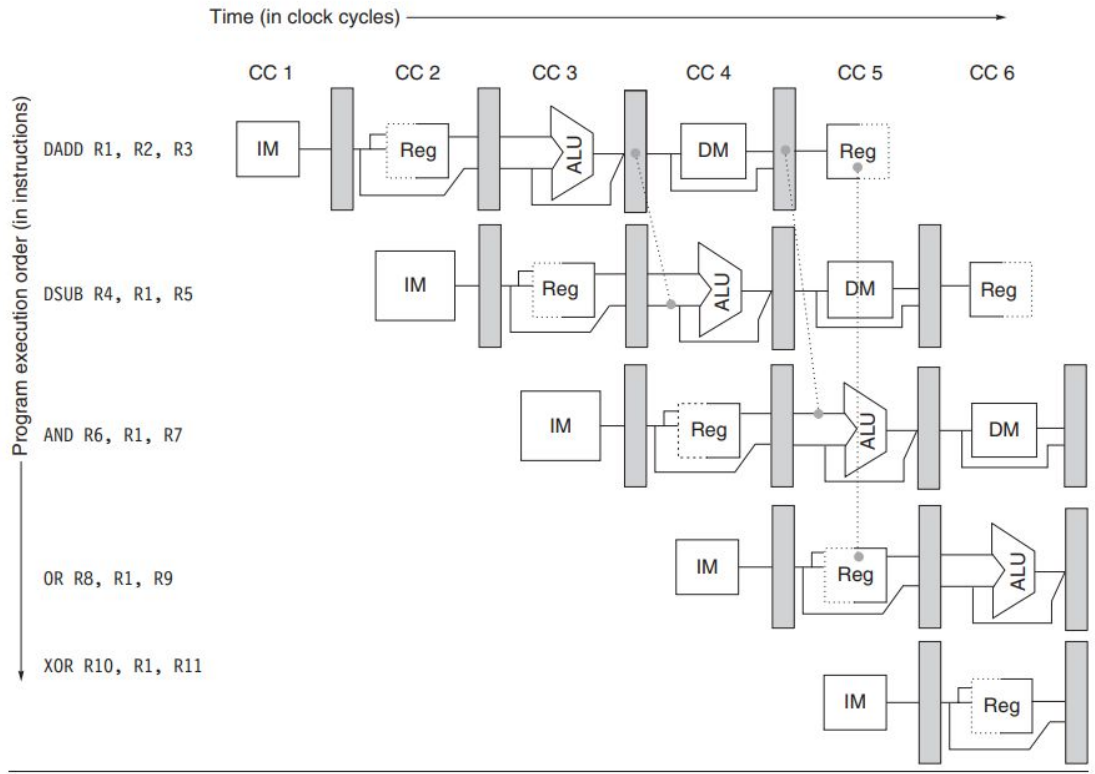


Data Hazards

Data hazard

- Changing the order of read/write access to operands so that order differs from the order seen by sequentially executing
  - The operand values are different between pipelined and non-pipelined processing
  - The stall needed to avoid old operands loaded
- Example
  - DADD R1,R2,R3
  - DSUB R4,R1,R5
  - AND R6,R1,R7
  - OR R8,R1,R9
  - XOR R10,R1,R11

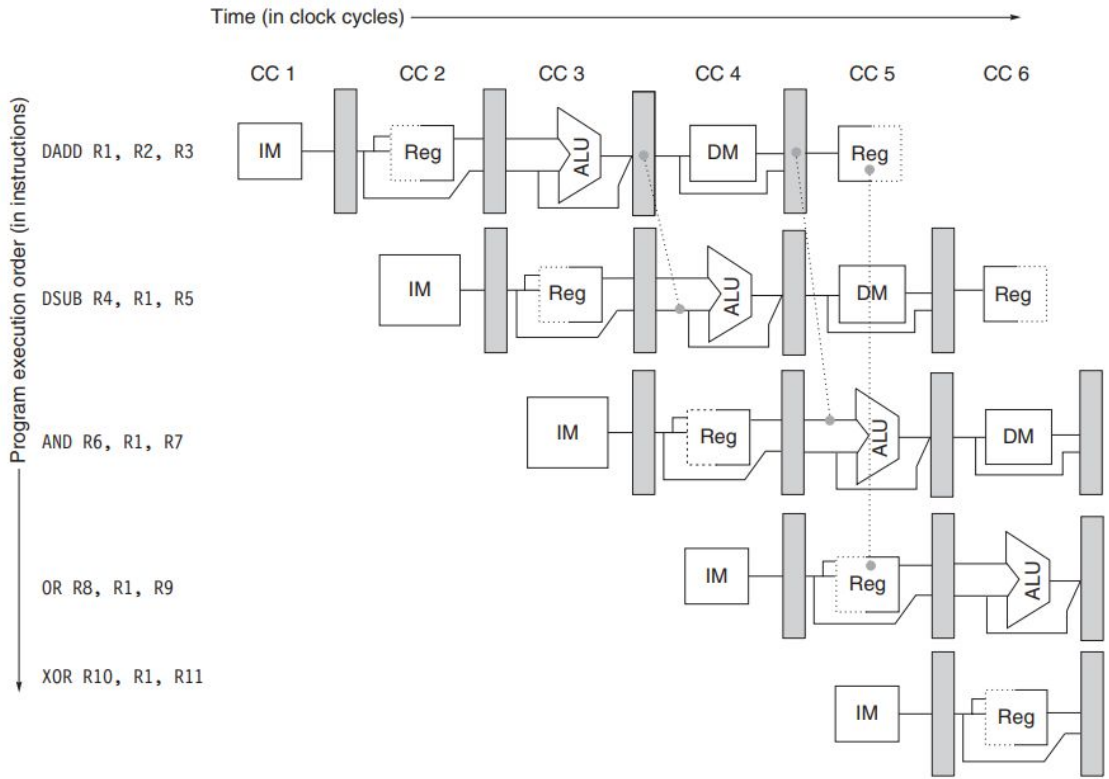
Data hazard (cont.)



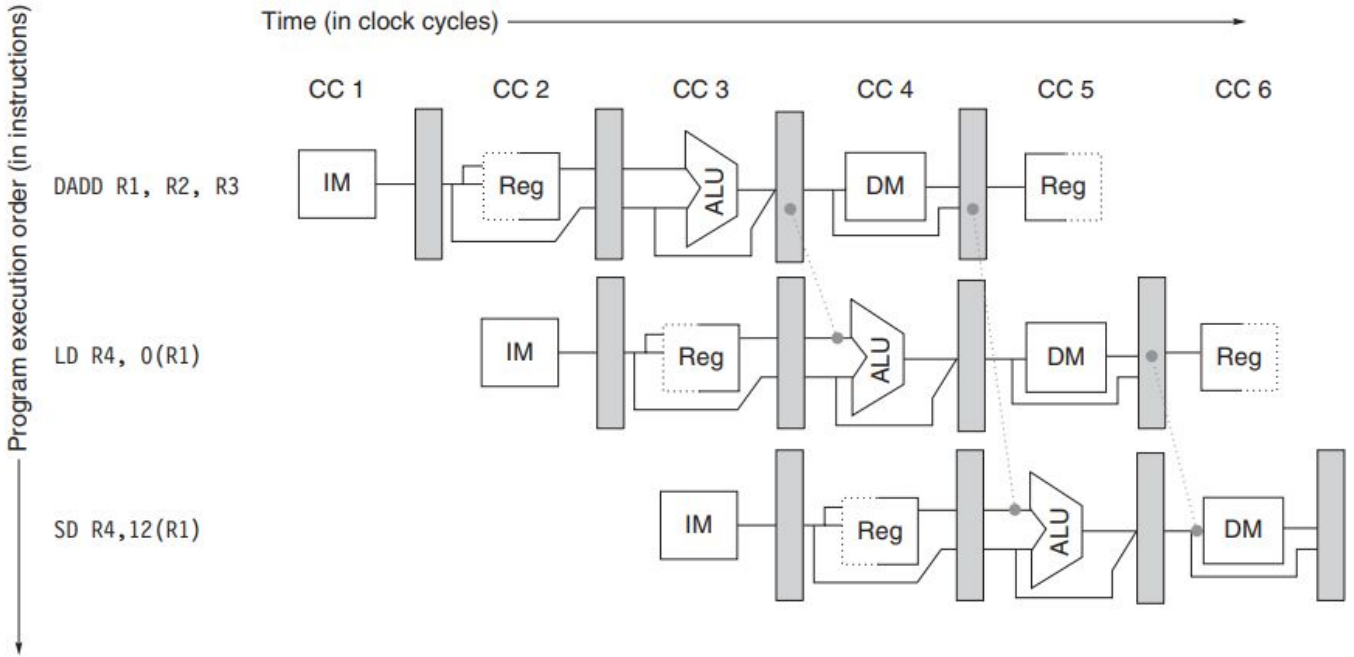
Minimizing data hazard stalls by bypassing

- Bypassing / short-circuiting
  - The result is passing directly to the functional unit that requires it
  - A result is forwarded from the pipeline register corresponding to the output of one unit to the input of same / another unit
  - Then the stall could be avoid

Minimizing data hazard stalls by bypassing



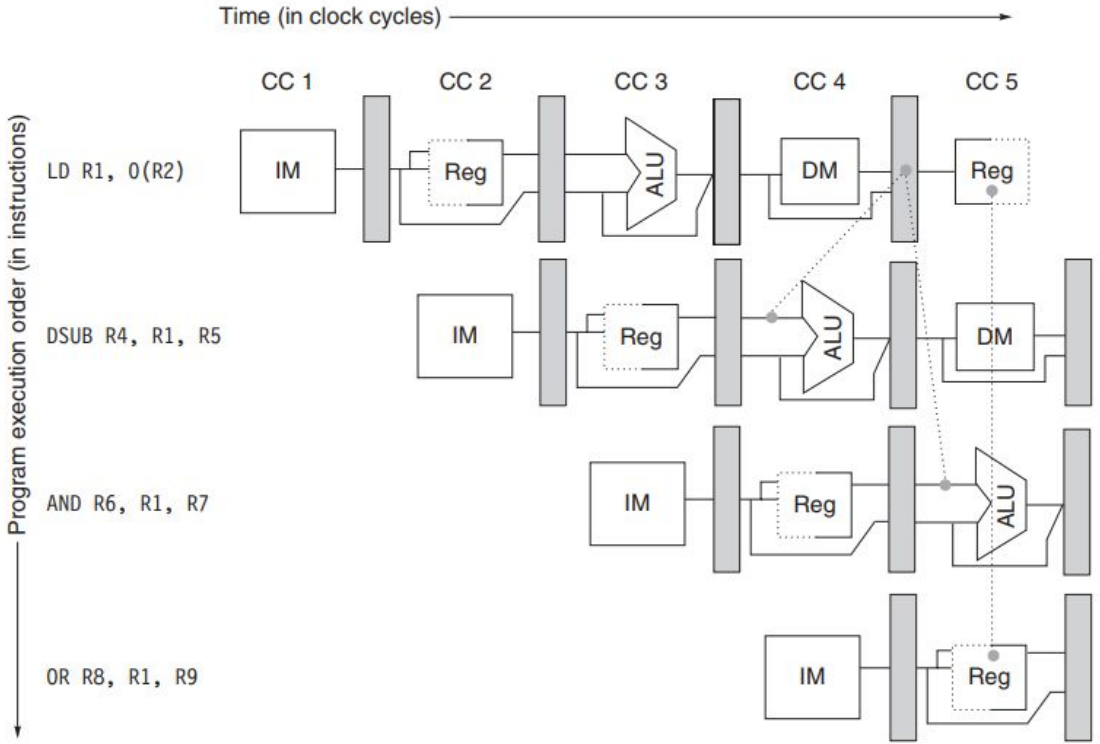
Minimizing data hazard stalls by bypassing (cont.)



Data hazards requiring Stalls

- Not all data hazards can be handled by bypassing
- Example
  - LD R1,0(R2)
  - DSUB R4,R1,R5
  - AND R6,R1,R7
  - OR R8,R1,R9

Data hazards requiring Stalls (cont.)



Data hazards requiring Stalls solution

- The hazard need to be detected
- The stall need to happen in hardware level to delay some cycle
- Invented of hardware **interlock**
  - Detect a hazard
  - Stall / bubble the pipeline until the hazard is cleared
  - CPI for the stalled instruction increases by the length of the stall

Data hazards requiring Stalls solution (cont.)

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB			
AND	R6,R1,R7			IF	ID	EX	MEM	WB		
OR	R8,R1,R9				IF	ID	EX	MEM	WB	

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB		
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB	
OR	R8,R1,R9				stall	IF	ID	EX	MEM	WB

Control Hazards

Control hazards

- The pipelining of branches and other instructions that possibly change the PC
- Recall of branches
  - If the branch changes PC to its target address => taken branch
  - If it falls through => not taken
- The taken branch leads to current loaded instructions in pipeline to be useless and cause great performance loss

Branch hazard stall example

- The branch instruction could not be detected until ID stages
- When the branch is detected, the first IF cycle of branch successor is essentially a stall
- Then a new instruction needs to be fetched following new PC
- **However**, if the branch is untaken, then first IF cycle still has meaning and should not be ignored => **could take advantage**

Branch instruction	IF	ID	EX	MEM	WB		
Branch successor		IF	IF	ID	EX	MEM	WB
Branch successor + 1				IF	ID	EX	MEM
Branch successor + 2					IF	ID	EX

Reducing pipeline branch penalties

- Four compile time schemes to deal with branch hazard
  - **Stall the pipeline** until the branch destination is known
  - **Treat every branch as not taken**, allowing the hardware to continue until the branch outcome is definitely known
  - **Treat every branch as taken**, begin fetching and executing the target branch as soon as the branch is decoded and the target address is computed
  - **Delayed branch**

Treat branch as taken / not taken

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

Delayed Branch

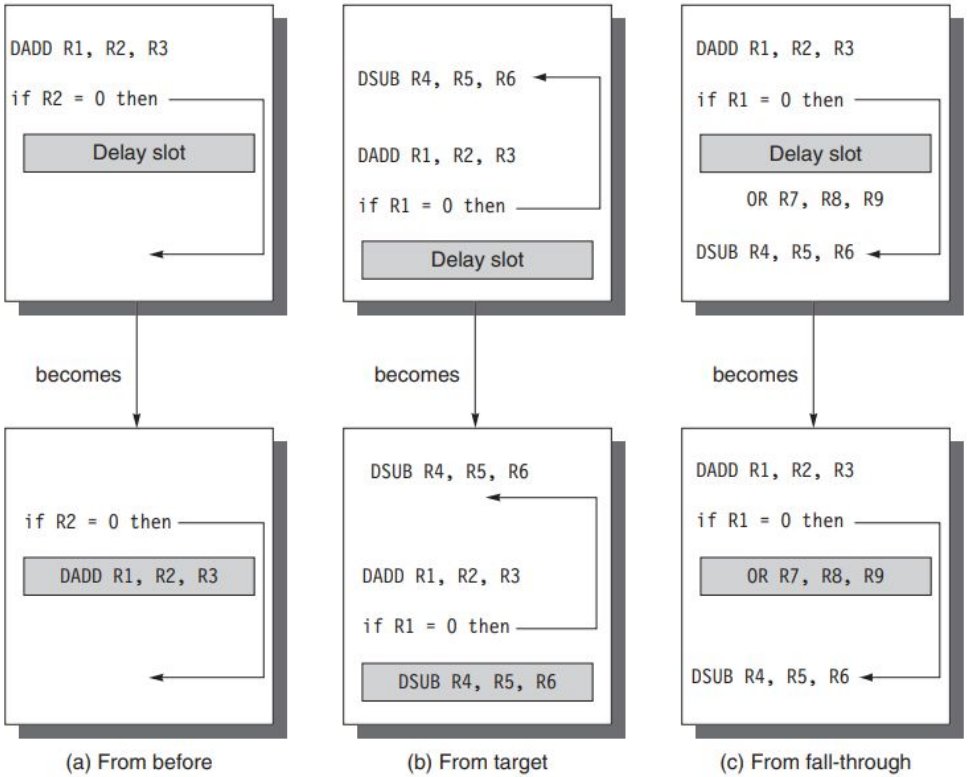
- A technique that intentionally execute instructions in a “branch delay slot” while waiting for the branch outcome is definitely known
- Example: the execution cycle with a branch delay of one is
  - Branch instruction
  - Sequential successor
  - Branch target if taken
- The sequential successor is in the branch delay slot
- This instruction is executed whether or not the branch is taken

Delayed Branch (cont.)

Untaken branch instruction	IF	ID	EX	MEM	WB				
Branch delay 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				
Branch delay instruction (i + 1)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Scheduling the branch delay slot

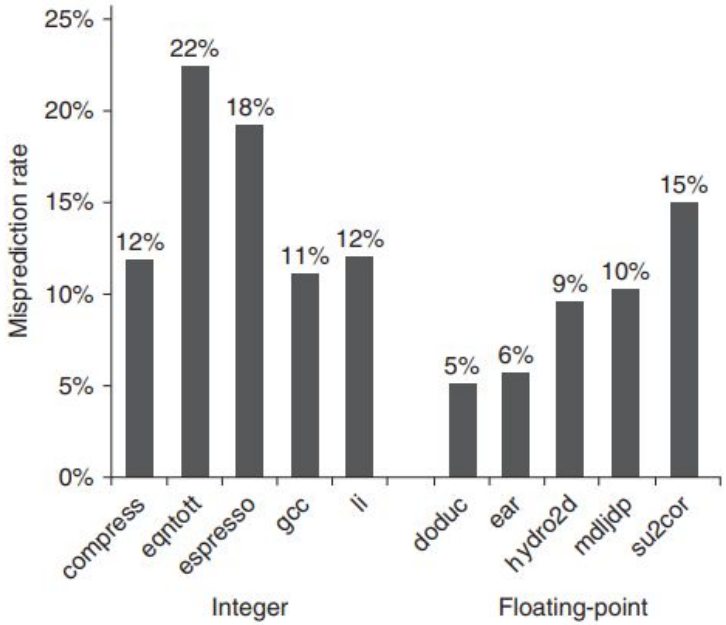


Reducing Cost of Branch through Prediction

Static branch prediction

- Rely on information available at compile time (profile information collected from earlier runs) to propose a branch prediction
- Based on key observation
  - The behavior of branches is often bimodally distributed
  - An individual branch is often highly biased toward taken or untaken
- Effectiveness of scheme depends both on
  - The accuracy of the scheme
  - The frequency of conditional branches

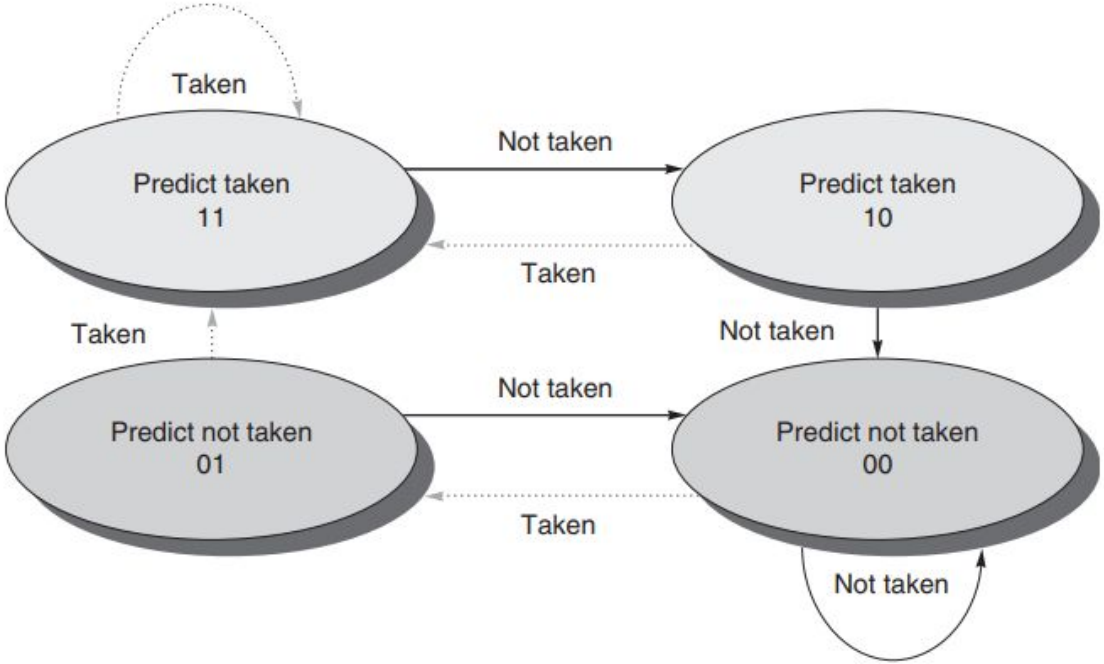
Static branch prediction (cont.)



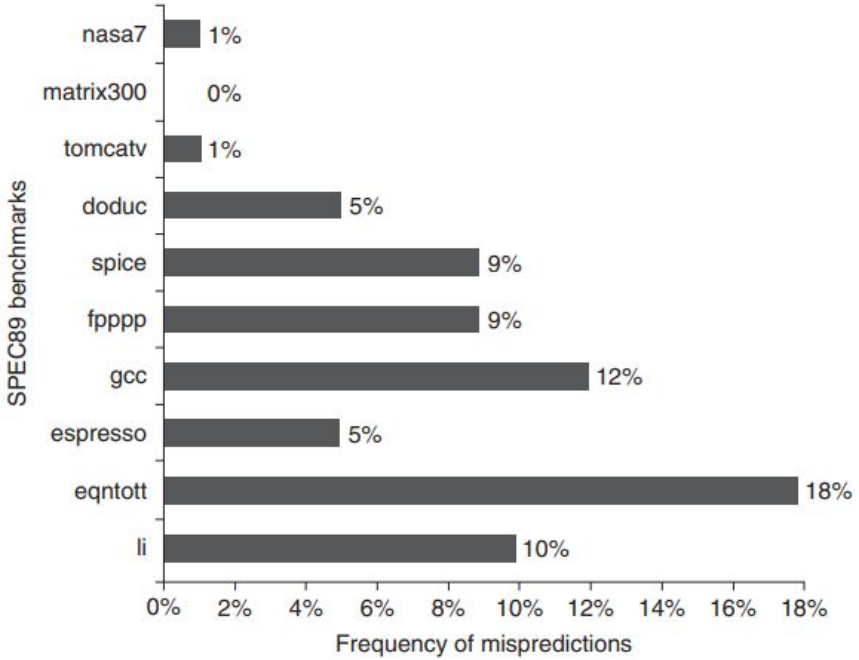
Dynamic branch prediction

- Branch-prediction buffer / Branch history table
  - A small memory indexed by the lower portion of the address of the branch instruction
  - Contains 1 bit that says whether the branch was recently taken or not
  - Simplest sort of buffer to be a hint for the branch prediction direction
- The performance of the buffer depends on
  - How often the prediction is for the branch of interest
  - How accurate the prediction is when it matches
- Improvement of branch-prediction buffer
  - Using 2-bit prediction schemes instead of 1-bit schemes
  - The prediction must miss twice before it is changed between taken and untaken
  - Based on a finite-state processor to provide prediction

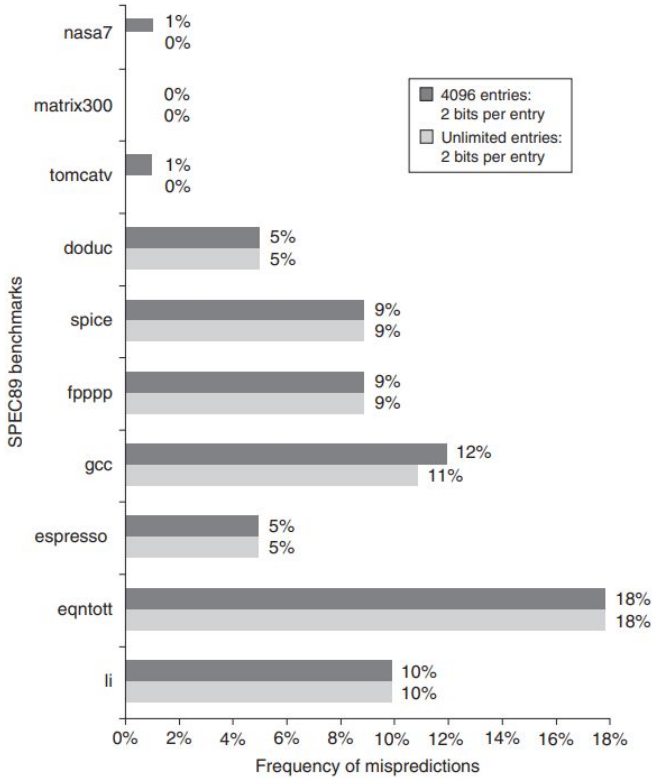
Dynamic branch prediction state machine



Dynamic branch prediction performance



Dynamic branch prediction performance (cont.)



Thank you for you listening