LECTURE - 07

Recall: Data Hazards

- Have to be detected dynamically, and pipeline stalled if necessary
- *Instruction issue:* process of moving the instruction from ID stage to EX
- For DLX, all data hazards can be checked before *instruction issue*
  - Also, control for data forwarding can be determined
  - This is good since instruction is suspended *before* any machine state is updated

Pipeline Interlock for "Load"

Opcode of ID/EX (ID/EX.IR0..5)	Opcode of IF/ID (IF/ID.IR0..5)	Check for interlock
Load	Reg-Reg ALU	ID/EX.IR11.15 == IF/ID.IR6..10
Load	Reg-Reg ALU	ID/EX.IR11.15 == IF/ID.IR11..15
Load	Load, store, ALU immediate, or branch	ID/EX.IR11.15 == IF/ID.IR6..10

Control Logic for Data-Forwarding

- Data forwarding always happens
  - *From* ALU or data-memory output
  - *To* ALU input, data-memory input, or zero-detection unit
- Which registers to compare?
  - Compare the **destination register** field in EX/MEM and MEM/WB latches with the **source register** fields of IR in ID/EX and EX/MEM stages

Control Hazard

- Result of **branch instruction** not known until end of MEM stage
- Naïve solution: stall until result of branch instruction is known
  - That an instruction is a branch is known at the end of its ID cycle
  - Note: "IF" may have to be repeated

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
Branch	IF	ID	EX	MEM	WB				
Branch succ		IF	STALL	STALL	IF	ID	EX	MEM	WB
Branch succ + 1						IF	ID	EX	MEM

Reducing the Branch Delay

- Three clock cycles wasted for every branch ==> significantly bad performance
- Two things to speedup:
  - Determine earlier, if branch is taken
  - Compute PC earlier
- Both can be done one cycle earlier
- But, beware of data hazard

Branch Behaviour of Programs

- Integer programs: 13% forward conditional, 3% backward conditional, 4% unconditional
- FP programs: 7%, 2%, and 1% respectively
- 67% of branches are taken
  - 60% forward branches are taken
  - 85% backward branches are taken

Handling Control Hazards

- **Stall:** Naïve solution
- **Predict untaken or Predict not-taken:**
  - Treat every branch as not taken
  - Only slightly more complex
  - Do not update *machine state* until branch outcome is known
  - Done by clearing the IF/ID register of the fetched instruction

Predict Untaken Scheme

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I (Untaken branch)	IF	ID	EX	MEM	WB			
I + 1		IF	ID	EX	MEM	WB		
I + 2			IF	ID	EX	MEM	WB	
I + 3				IF	ID	EX	MEM	WB

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I (Taken branch)	IF	ID	EX	MEM	WB			
I + 1		IF	Noop	Noop	Noop	Noop		
Target			IF	ID	EX	MEM	WB	
Target + 1				IF	ID	EX	MEM	WB
Target +2					IF	ID	EX	MEM

More Ways to Reduce Control Hazard Delays

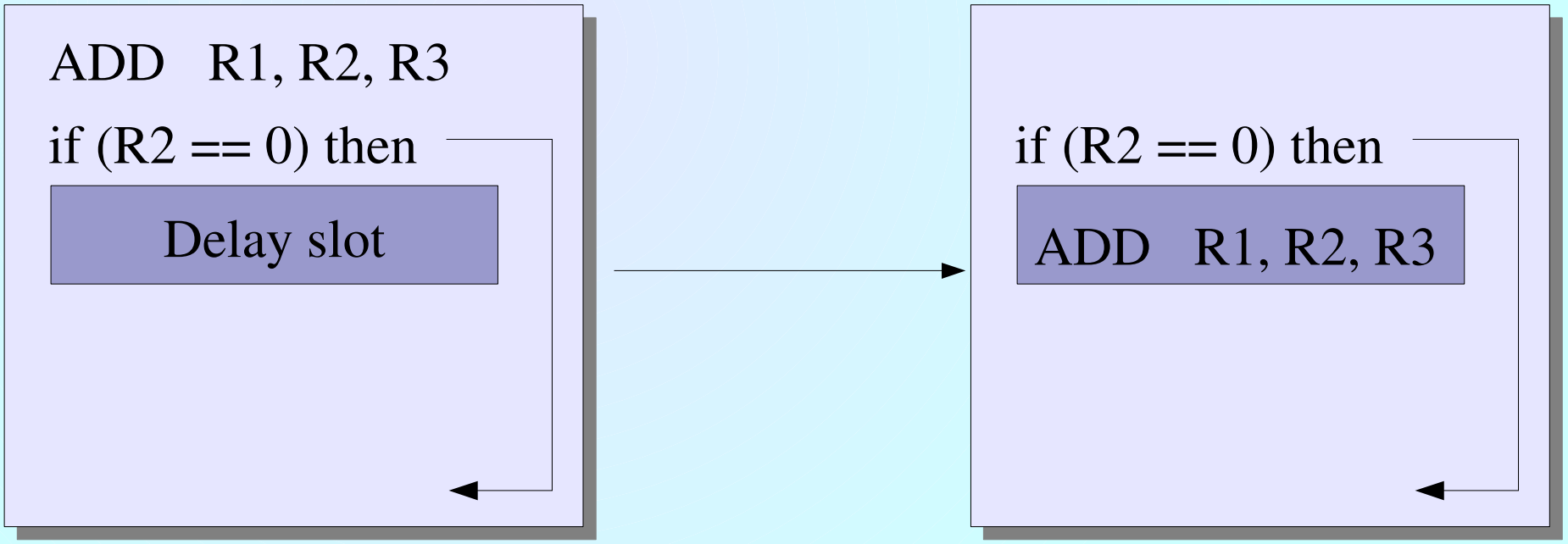
- **Predict taken:**
  - Treat every branch as taken
  - Not of any use in DLX since branch target is not known before branch condition anyway
    - May be of use in other architectures
- **Delayed branch:**
  - Instruction(s) after branch are executed anyway!
  - Sequential successors are called *branch-delay-slots*

Delayed Branch

EITHER	OR	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I (Untaken branch)	I (Taken branch)	IF	ID	EX	MEM	WB			
I + 1 (Branch delay)	I + 1 (Branch delay)		IF	ID	EX	MEM	WB		
I + 2	Target			IF	ID	EX	MEM	WB	
I + 3	Target + 1				IF	ID	EX	MEM	WB
I + 4	Target + 2					IF	ID	EX	MEM

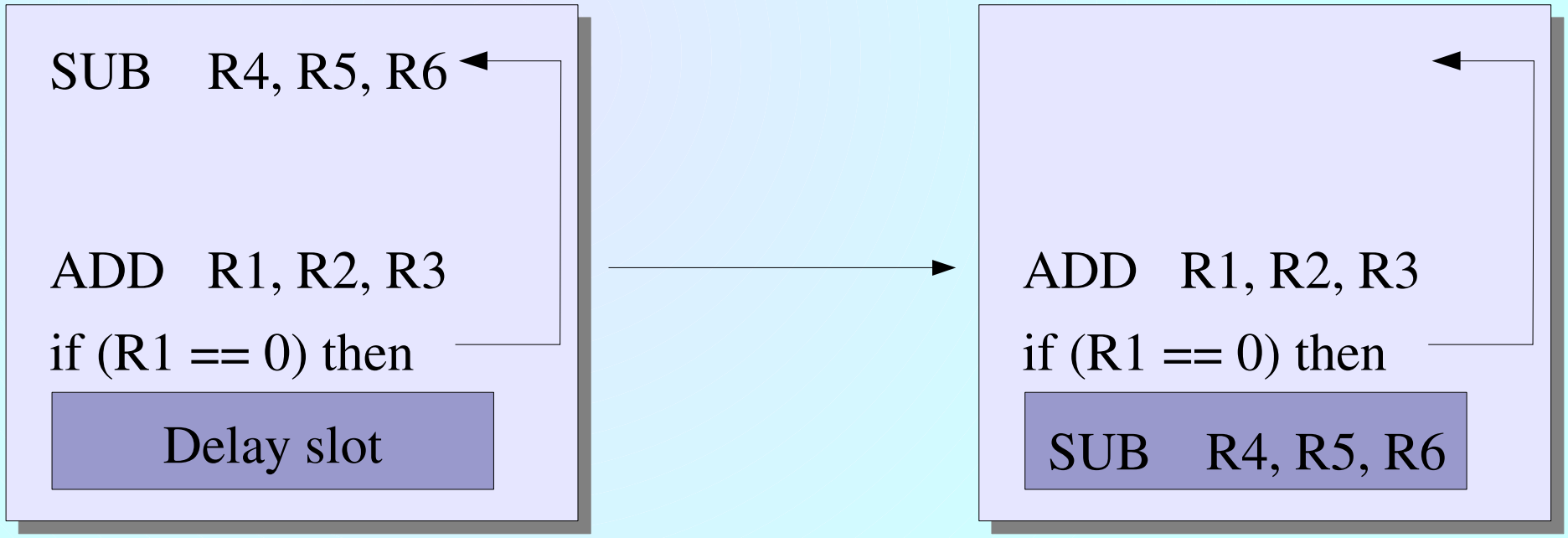
- DLX has one delay-slot
- Note: another branch instruction cannot be put in delay-slot
- Compiler has to fill the delay-slots

Filling the Delay-Slot: Option 1 of 3



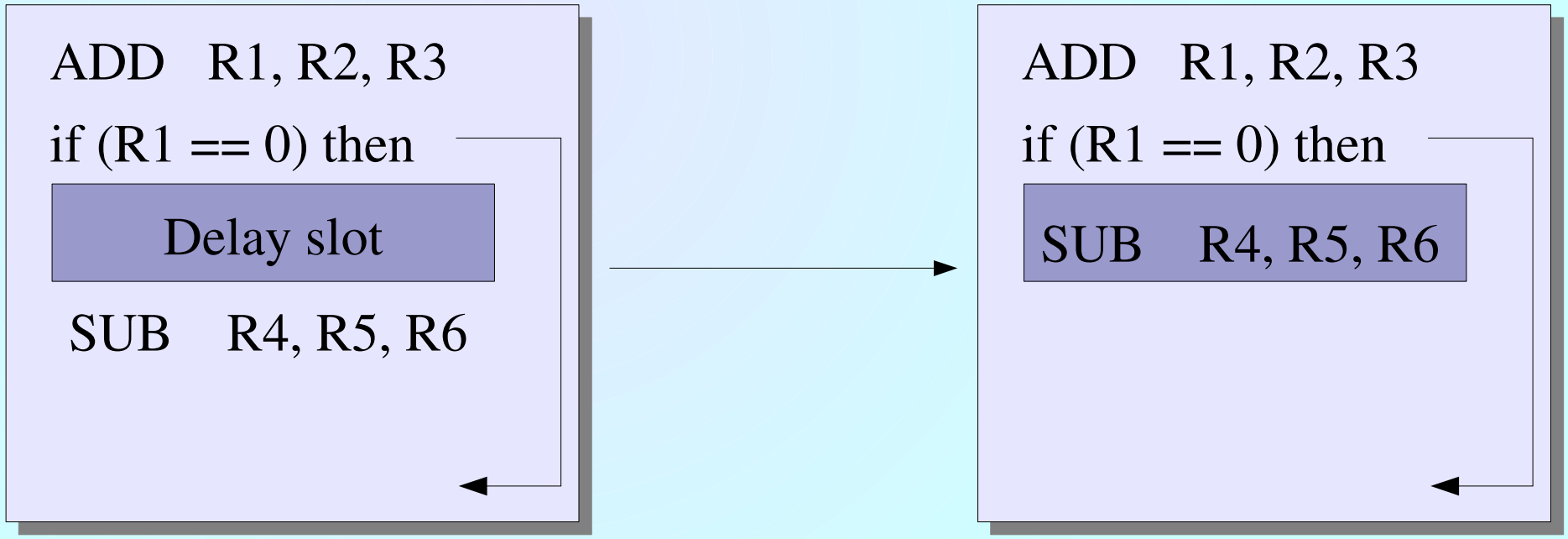
- Fill the slot **from before** the branch instruction
- **Restriction:** branch must not depend on result of the filled instruction
- **Improves performance:** always

Filling the Delay-Slot: Option 2 of 3



- Fill the slot **from the target** of the branch instruction
- **Restriction:** should be OK to execute instruction even if not taken
- **Improves performance:** when branch is taken

Filling the Delay-Slot: Option 3 of 3



- Fill the slot **from fall through** of the branch
- **Restriction:** should be OK to execute instruction even if taken
- **Improves performance:** when branch is not taken

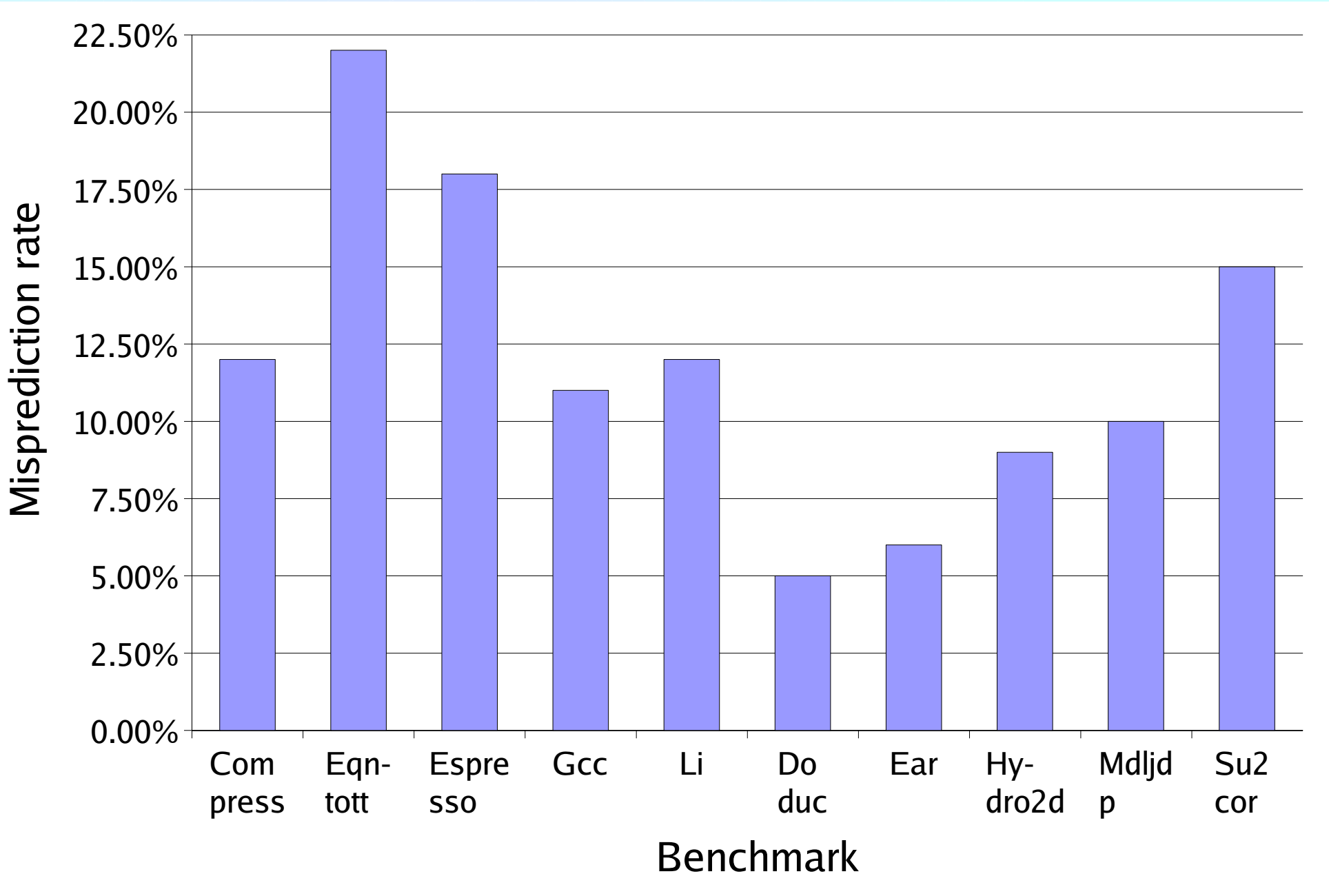
Helping the Compiler

- Encode the *compiler prediction* in the branch instruction
  - CPU knows whether branch was predicted taken or not taken by compiler
  - *Cancel* or *nullify* if prediction incorrect
  - Known as *canceling* or *nullifying* branch
    - Options 2 and 3 can now be used without restrictions

Static Branch Prediction

- Predict-taken
- Predict-untaken
- Prediction based on direction (forward/backward)
- Profile-based prediction

Static Misprediction Rates



Some Remarks

- Delayed branches are architecturally visible
  - Strength as well as weakness
  - Advantage: better performance
  - Disadvantage: what if implementation changes?
- Deeper pipeline ==> more branch delays ==> delay-slots may no longer be useful
  - More powerful dynamic branch prediction
- Note: need to remember extra PC while taking exceptions/interrupts
- Slowdown due to mispredictions:

$$1 + Branch\ frequency \times Misprediction\ rate \times Penalty$$

Further Issues in Pipelining

- Exceptions
- Instruction set issues
- Multi-cycle operations