EE309(S2): Microprocessors

Spring 2025

[Week#10 Slides]

Instructor: Shalabh Gupta

Handling Data Hazards

- Read-After-Write (Read-After-Write): Can be overcome by data forwarding
  - Requires extra hardware for the datapath
  - Performance is not affected
- Load-Use Data Hazard: Generally overcome by pipeline interlocking, i.e.
  - Stalling or adding a bubble in the instruction pipeline of currently being executed instructions (i.e. the cycle is repeated or not executed)
  - Affects processor performance, extensive hazard detection unit required
- Original MIPS (Microprocessor without Interlocked Pipeline Stages) had no interlocking

  Compiler can overcome Load-Use or RAW data hazard or by rearranging the sequence of instructions (may not be possible always).

Pipelining: Data Hazard Example

(RAW: Read-After-Write)

Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

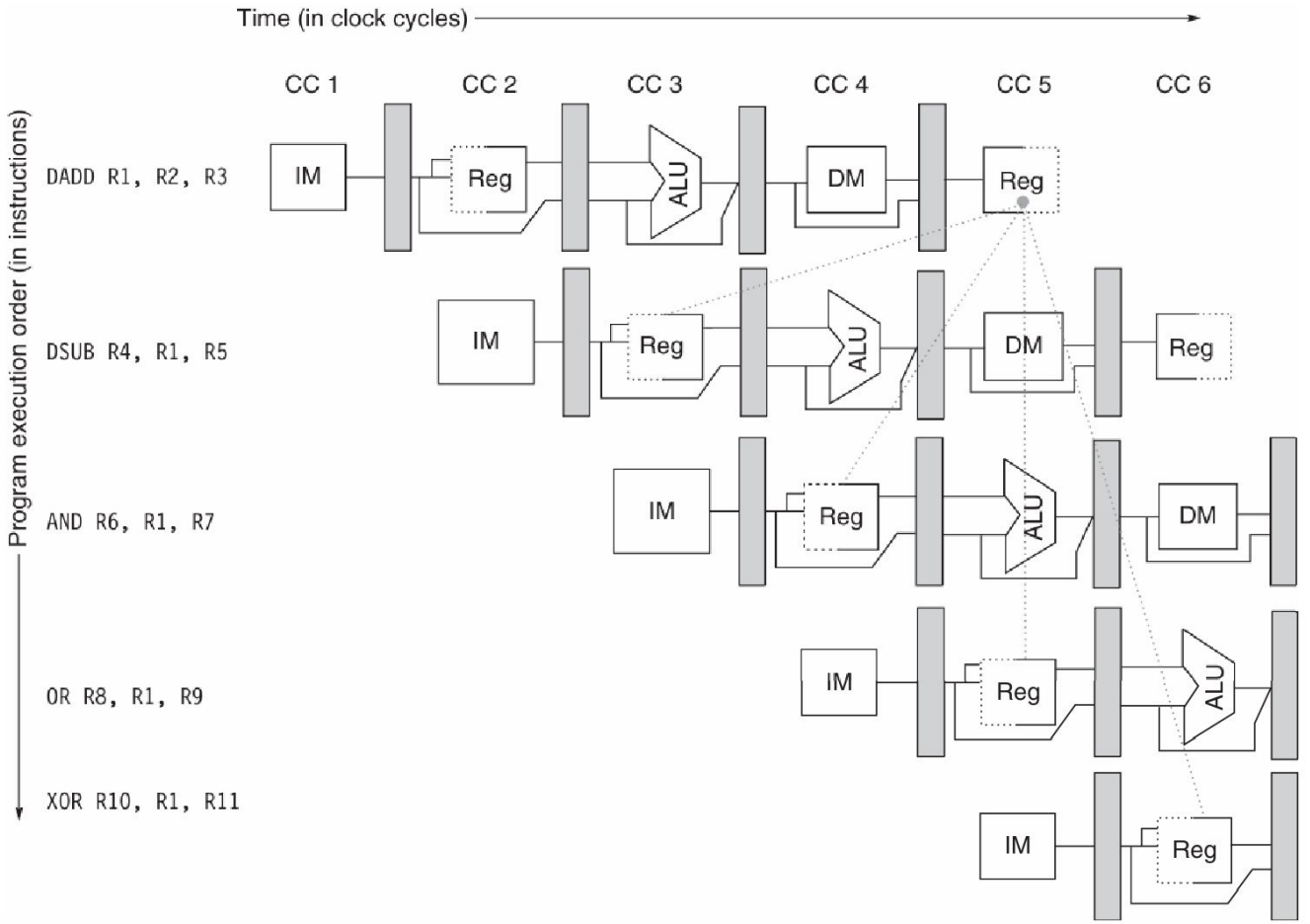


Figure C.6 The use of the result of the DADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

Pipelining: Overcoming Data Hazard by Forwarding

(RAW: Read-After-Write)

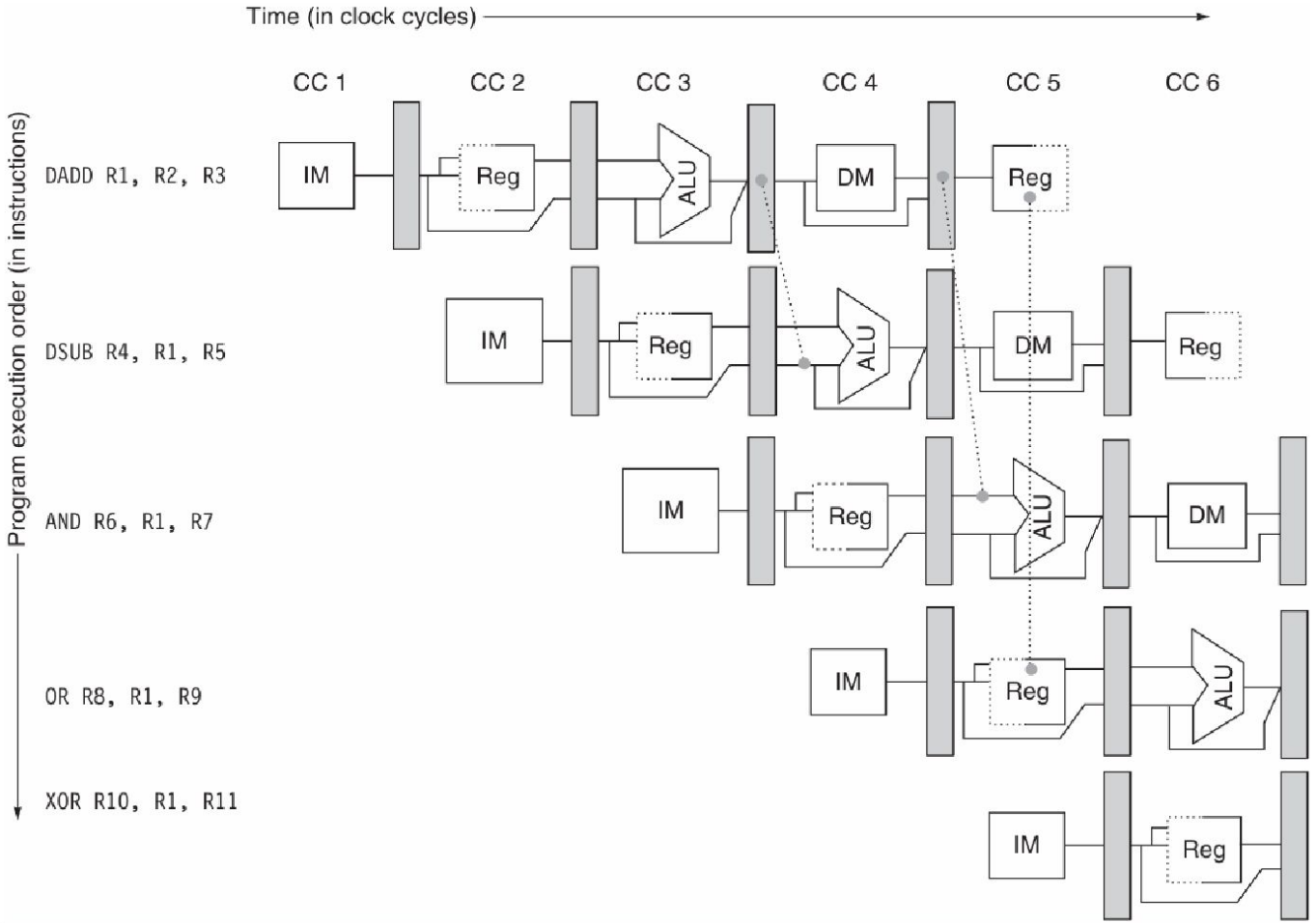


Figure C.7 A set of instructions that depends on the DADD result uses forwarding paths to avoid the data hazard.

Handling Data Hazards: Reordering

- Example: A := B + C; D := E+ F

  Usual code will be

```
LW Rb, B
LW Rc, C
ADD Ra, Rb, Rc
SW Ra, C

LW Rd, D
LW Re, E
ADD Rf, Rd, Re
SW Re, C
```

Pipelining: Data Hazard

Another Example of Forwarding

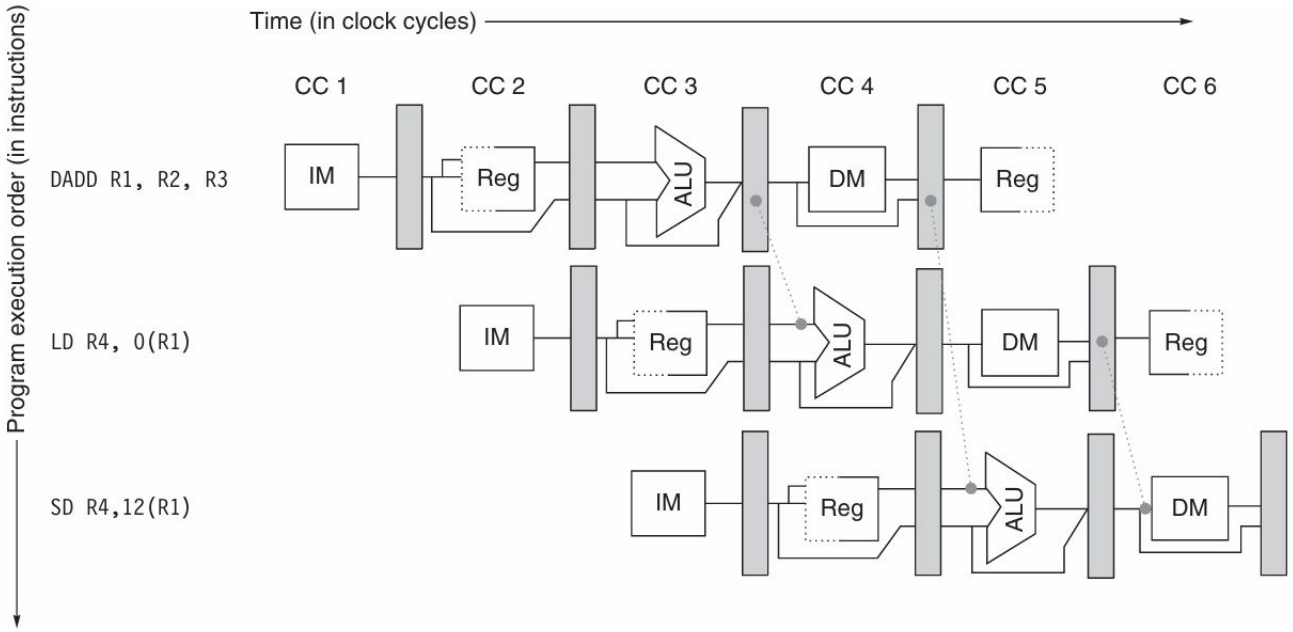


Figure C.8 Forwarding of operand required by stores during MEM. The result of the load is forwarded from the memory output to the memory input to be stored. In addition, the ALU output is forwarded to the ALU input for the address calculation of both the load and the store (this is no different than forwarding to another ALU operation). If the store depended on an immediately preceding ALU operation (not shown above), the result would need to be forwarded to prevent a stall.

Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

Pipelining: Structural Hazard Example

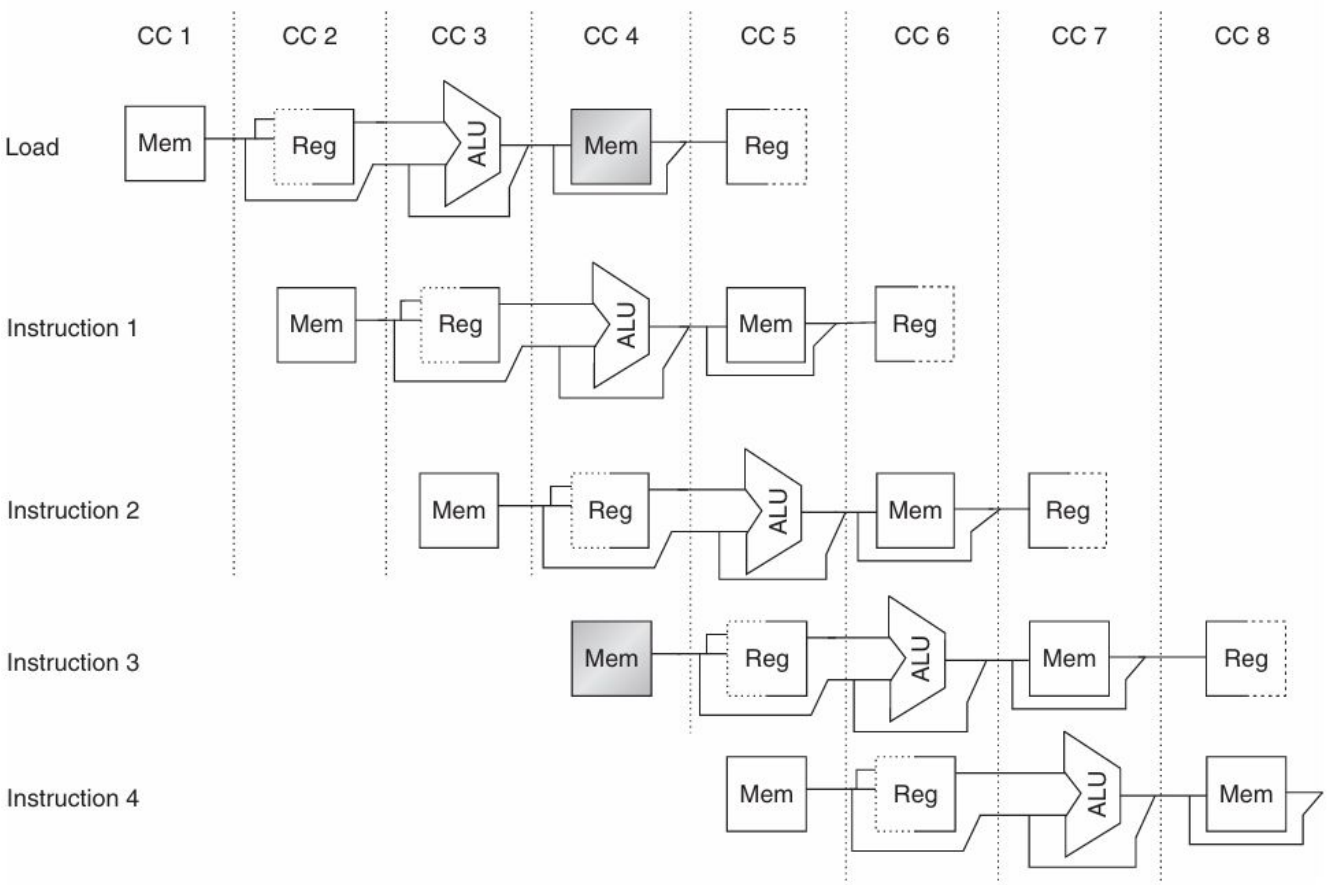


Figure C.4 A processor with only one memory port will generate a conflict whenever a memory reference occurs. In this example the load instruction uses the memory for a data access at the same time instruction 3 wants to fetch an instruction from memory.

Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

Pipelining: Structural Hazard Example

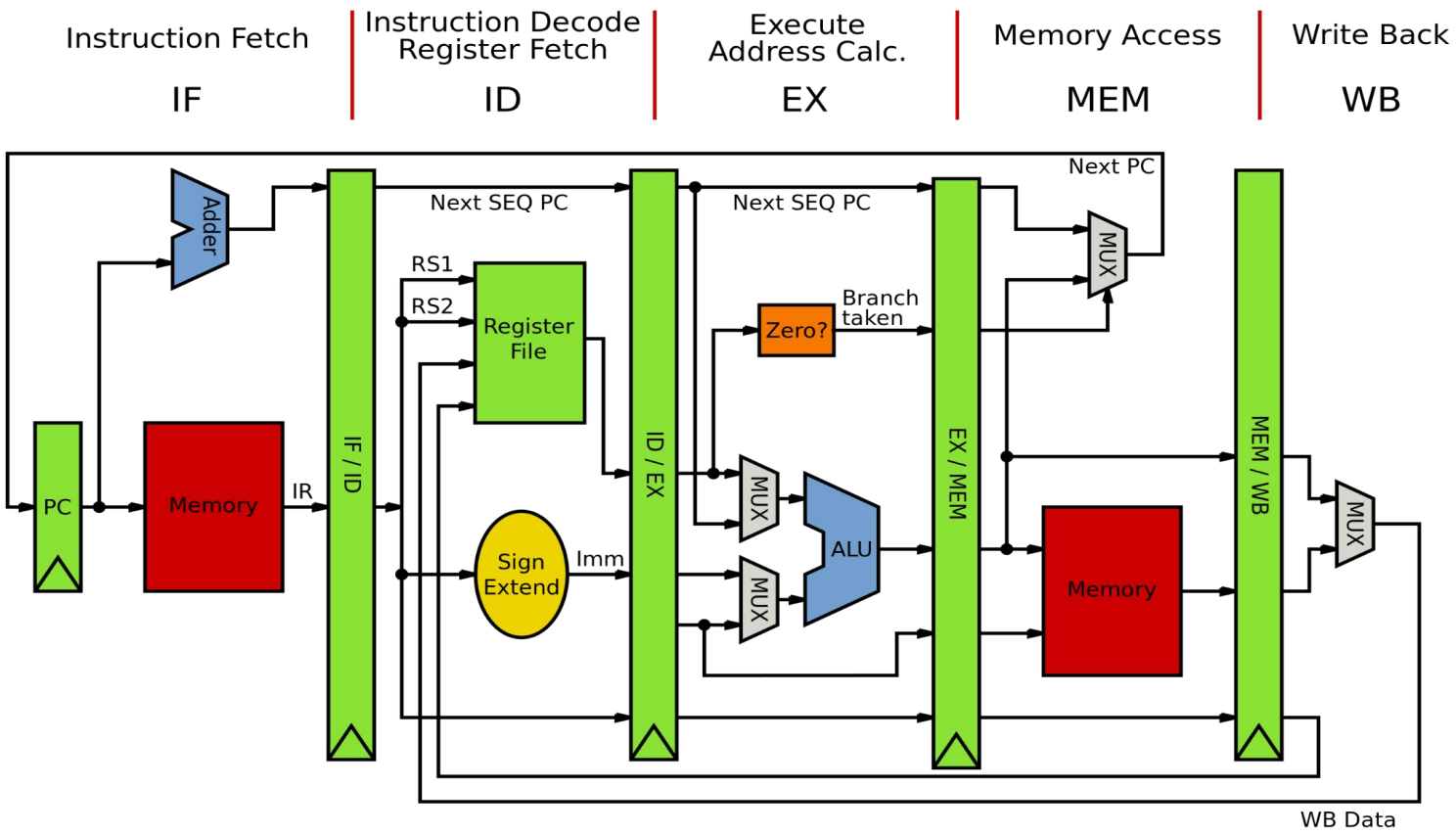
	Clock cycle number									
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				Stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX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

Figure C.5 A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load

Practice some examples as question from the Ref book - to tell how average CPI gets affected by different hazards under different conditions

Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

Standard (Conceptual): MIPS Pipelined Architecture



Branching: Reducing delay in branch address computation

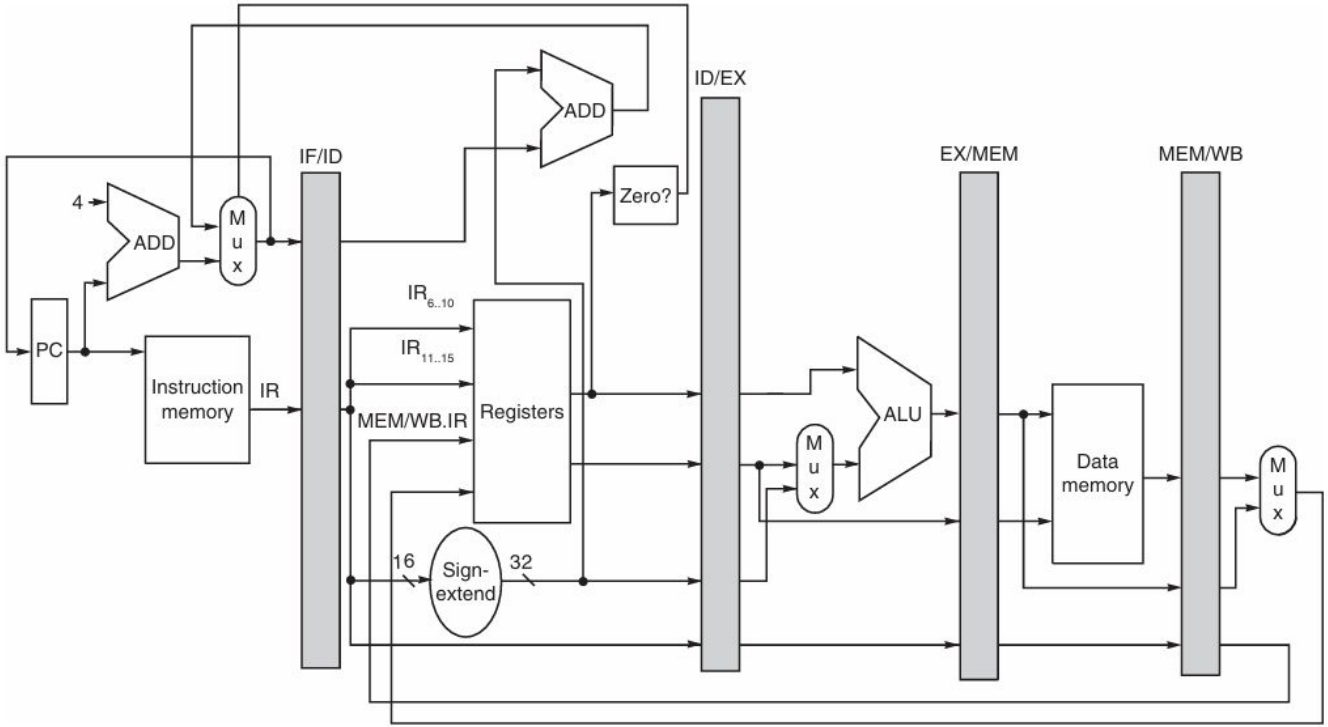


Figure C.28 The stall from branch hazards can be reduced by moving the zero test and branch-target calculation into the ID phase of the pipeline. Notice that we have made two important changes, each of which removes 1

Control Hazard: Scheduling Delay slot

What about unconditional branching?

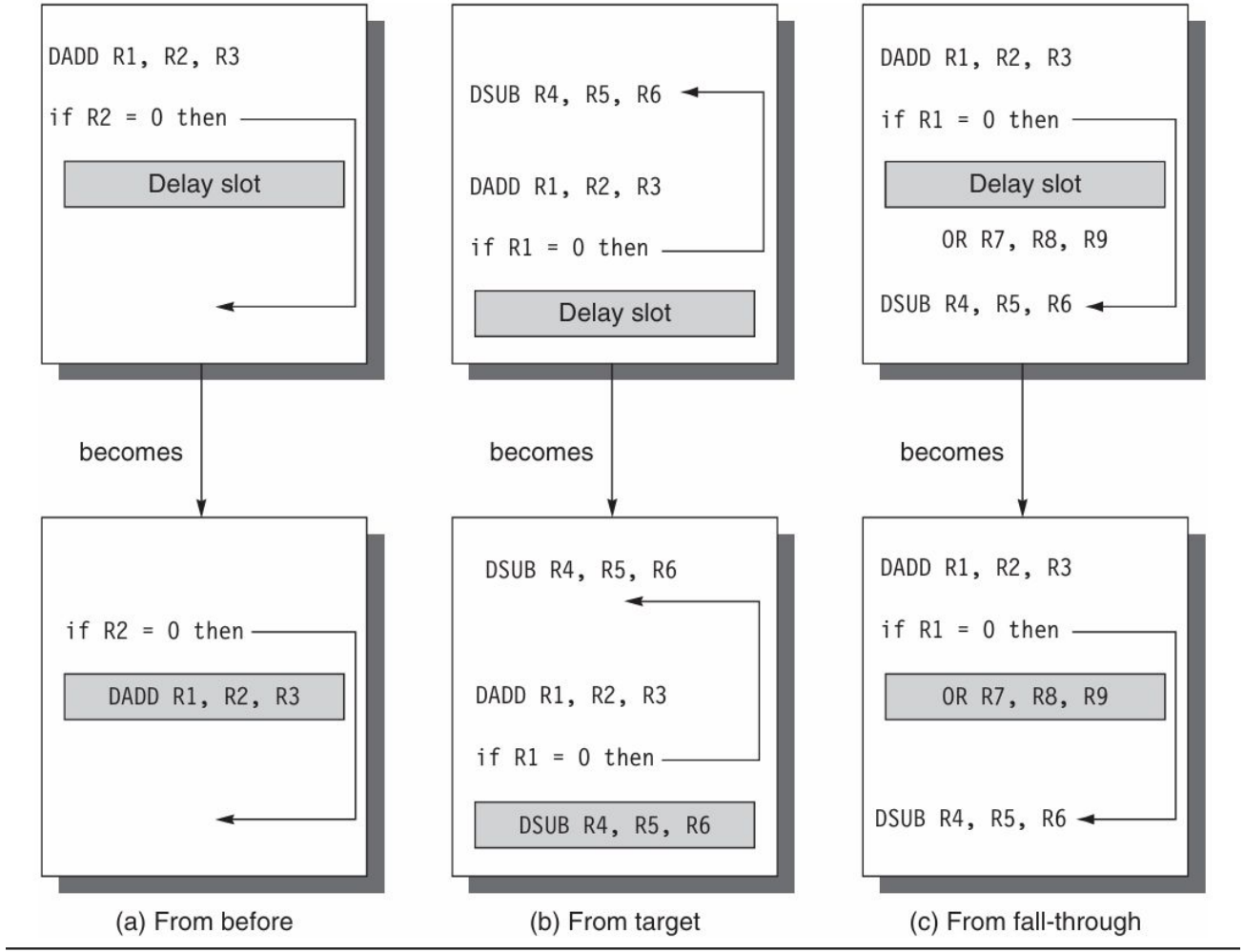


Figure C.14
Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

Branch prediction to improve performance

- Static: Decided apriori (after running benchmarks)
- Dynamic: Decided dynamically (during the execution of the program)

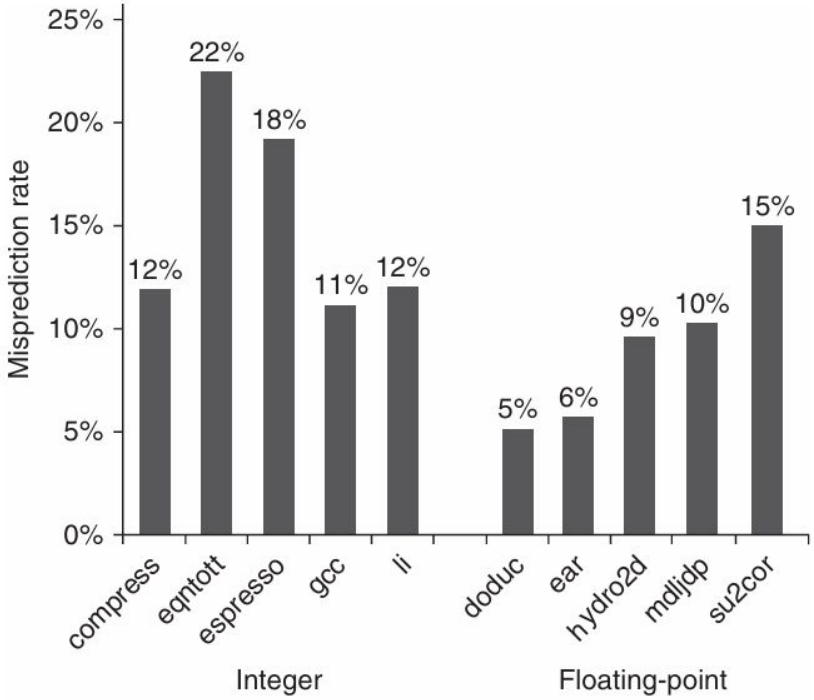


Figure C.17: Misprediction after profiling
Courtesy: Computer Architecture: A Quantitative Approach, 5th Edition by David A. Patterson and John L. Hennessy (2012)

Super-pipelining

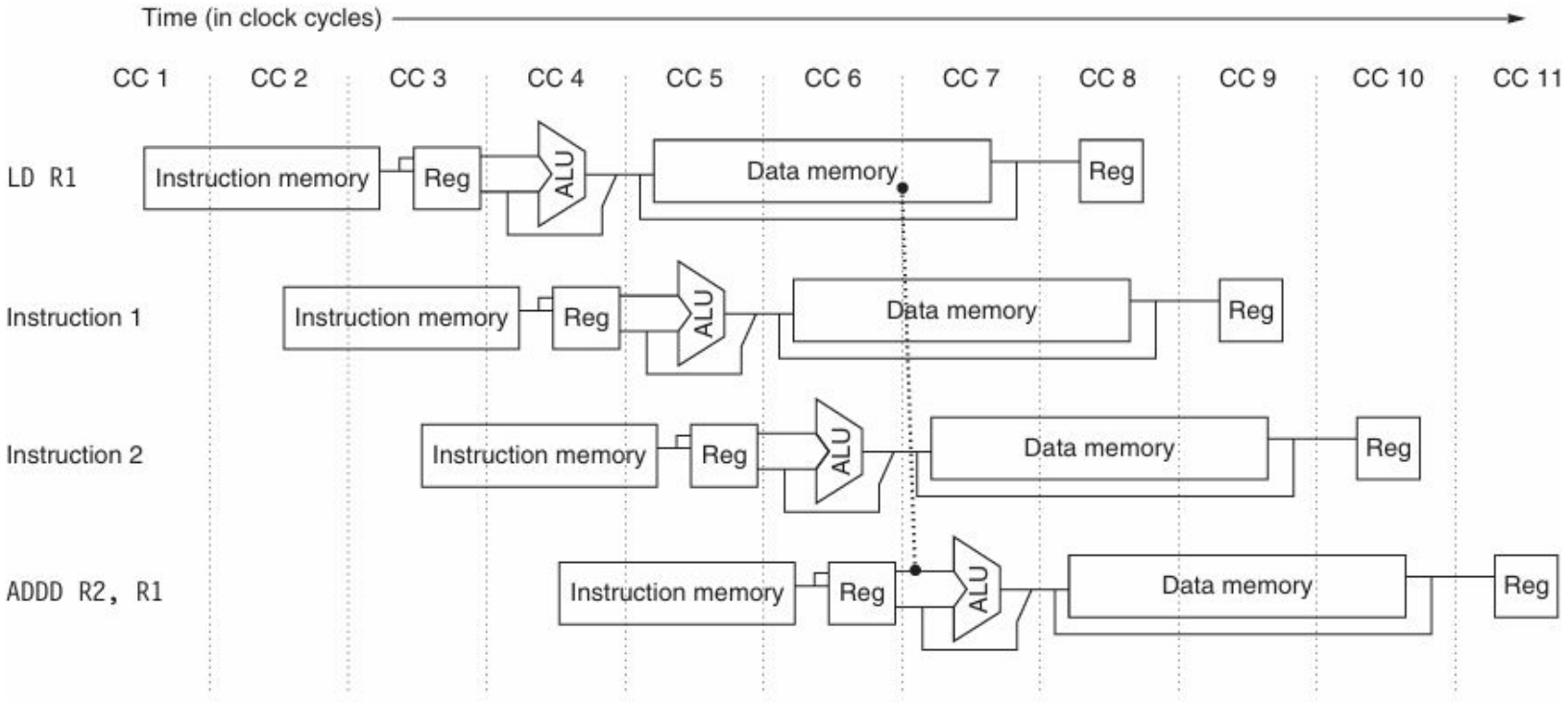


Figure C.42 The structure of the R4000 integer pipeline leads to a 2-cycle load delay. A 2-cycle delay is possible

Sample problem: Endsem 2013

Consider a set of instructions for a pipelined MIPS processor having five pipeline stages: IF (instruction fetch), ID (instruction decode), EX (Execution), MEM (memory Reference), WB (write back), as discussed in the class. Instructions 1, 2 load registers R1 and R2, respectively. Instructions 3 and 4 calculate R3=R1+R2 and R4=R1+R5, respectively, where R5 already contains valid data. Instruction 5 is an unconditional jump instruction to a new address that is available during ID cycle itself). Instruction 6 at the new location stores the value of R4 into the memory. Assume that two registers can be read from and one register can be written to concurrently in the register bank. Also, if there is a concurrent read and write to the same register, the value read from the register is the updated value.

(a) Show the timing for the instructions in the pipeline, if data forwarding is NOT used and data and instruction memories are shared (indicate stall cycles and corresponding hazards).

(b) Show the timing for the instructions in the pipeline, if data forwarding is NOT used but the data and instruction memories ARE SEPARATE (indicate stall cycles and corresponding hazards).

(c) Show the timing for the instructions in the pipeline, if data forwarding IS used, the data and instruction memories ARE SEPARATE, and the compiler can rearrange the instructions to minimize the hazards (indicate stall cycles and corresponding hazards). You can also assume that some instruction(s) can be moved after the unconditional branch instruction