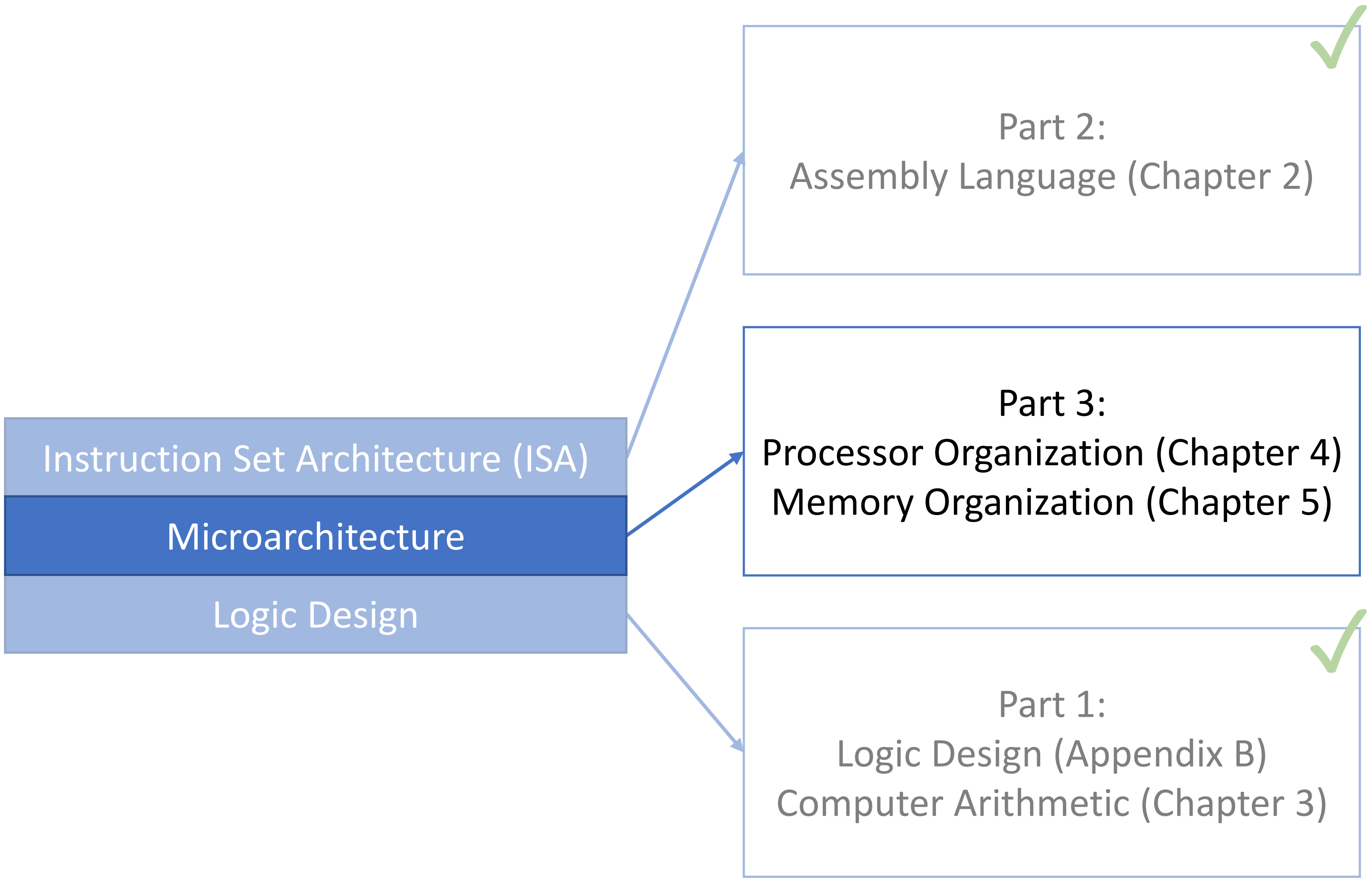
Lecture 25: Forwarding

CMPS 221 – Computer Organization and Design

Adapted by Izzat El Hajj from textbook slides for Computer Organization and Design, by David Patterson and John Hennessy, Morgan Kaufmann.

Last Time: Pipeline Hazards

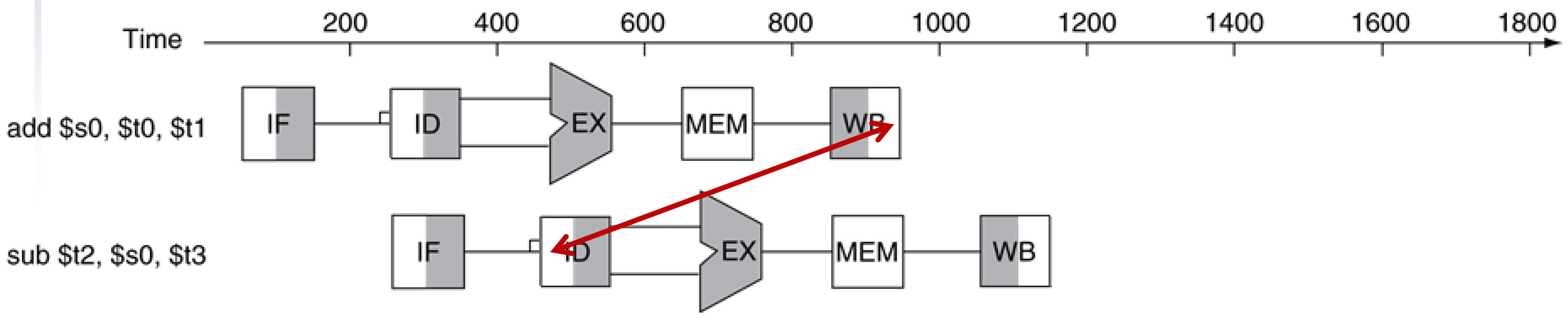

Instruction Set Architecture (ISA)
Microarchitecture
Logic Design
✓
Part 2:
Assembly Language (Chapter 2)
Part 3:
Processor Organization (Chapter 4)
Memory Organization (Chapter 5)
✓
Part 1:
Logic Design (Appendix B)
Computer Arithmetic (Chapter 3)

Hazards

- A **hazard** is a situation that prevents an instruction from executing a pipeline stage in the next cycle. Types of hazards:
  - **Data hazards**
    - An instruction needs to wait for another executing instruction to complete its data read/write
  - **Structure hazards**
    - A hardware structure required by an instruction is being used by another executing instruction
  - **Control hazards**
    - Deciding the next instruction to fetch depends on the outcome of another executing instruction

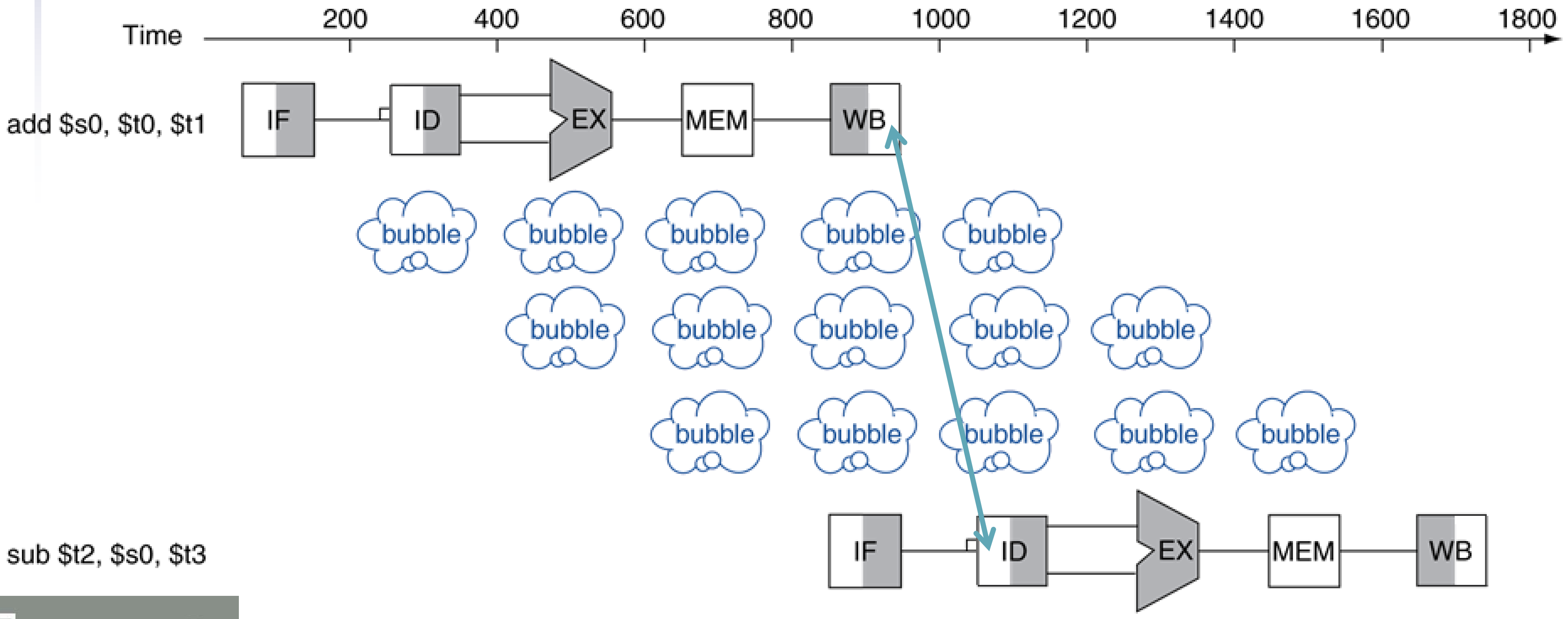
Data Hazards

- Recall: An instruction needs to wait for another executing instruction to complete its data read/write



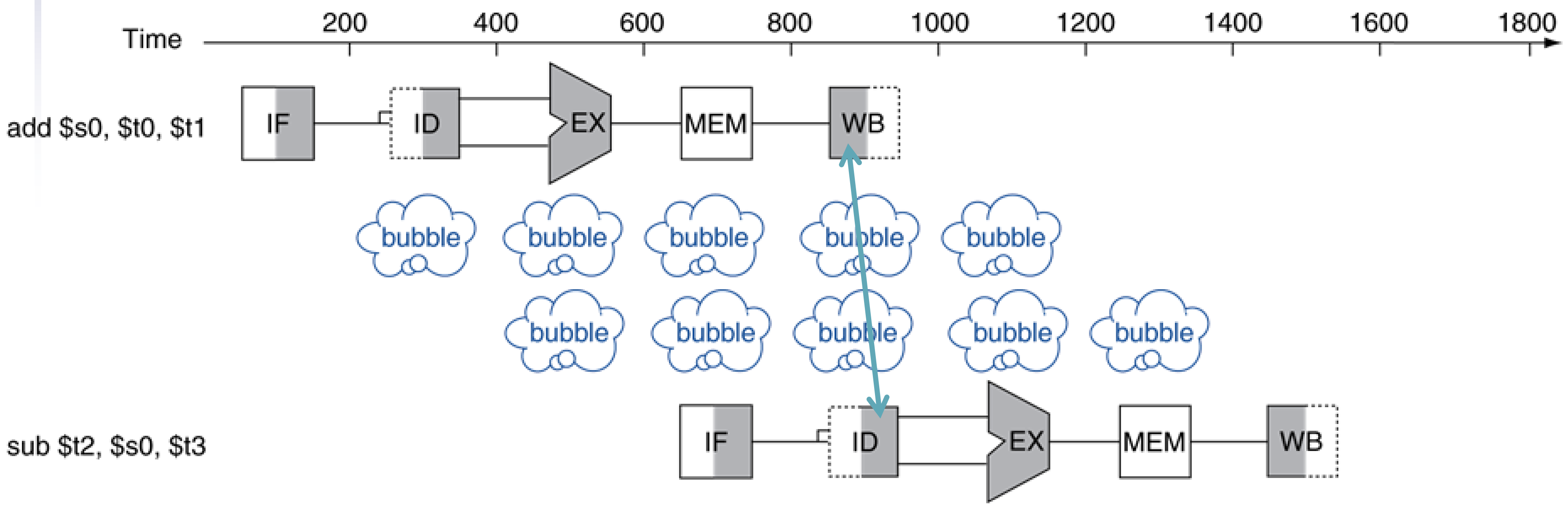
Stalling

- Solution: **stall** the instruction until the data hazard is gone
  - Insert “bubbles” in the pipeline
  - Disadvantage: many cycles wasted



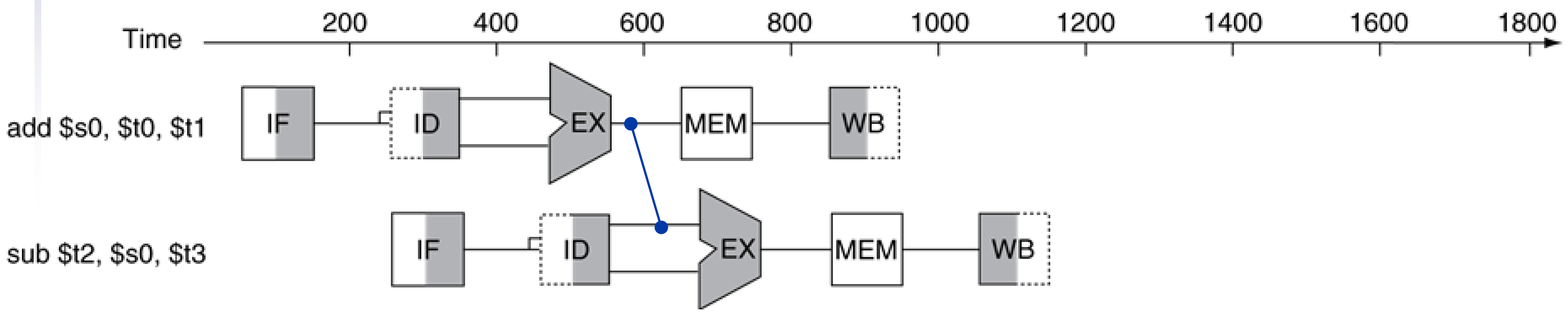
Optimizing WB/ID

- Another solution: perform WB in the first half of the cycle and ID in the second half
  - Allows values written during a cycle to be read in the same cycle



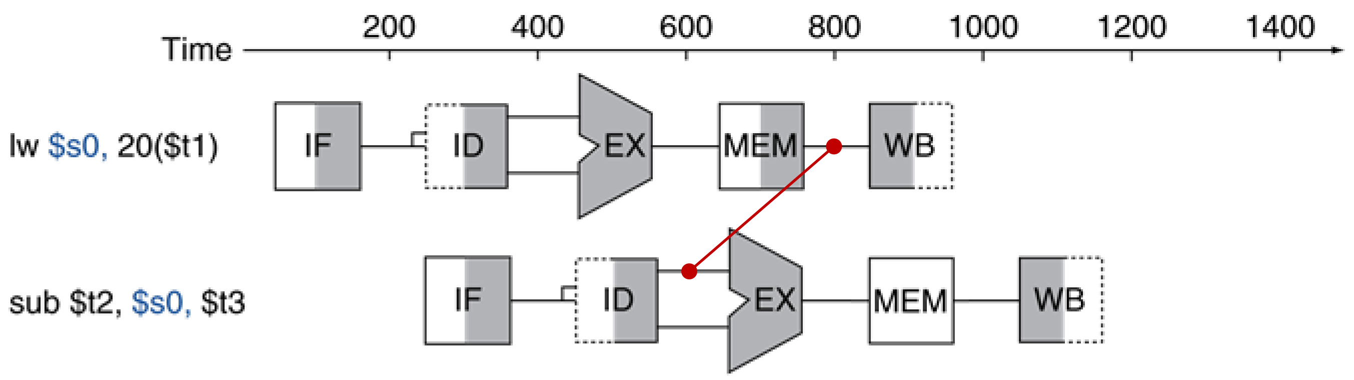
Forwarding (aka Bypassing)

- Yet another solution: **forwarding**
  - Result can be used as soon as it is computed without waiting for it to be stored in a register
  - Requires extra connections in the datapath



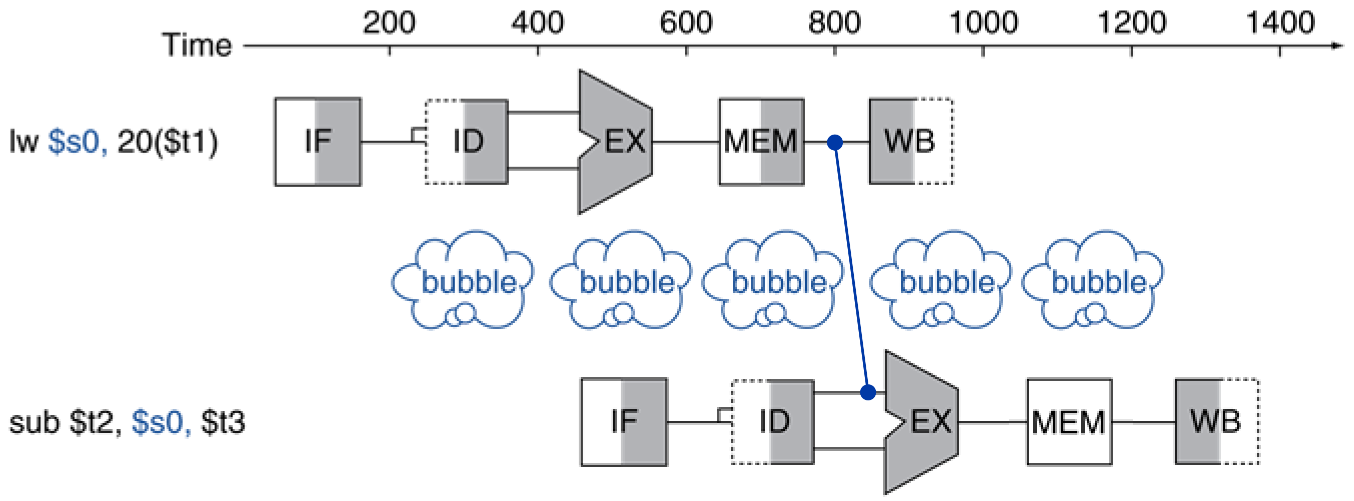
Load-Use Data Hazard

- Cannot always avoid stalls by forwarding
    - e.g., result of a load not available until after the MEM stage



Load-Use Data Hazard

- Solution: stalling is inevitable in this case

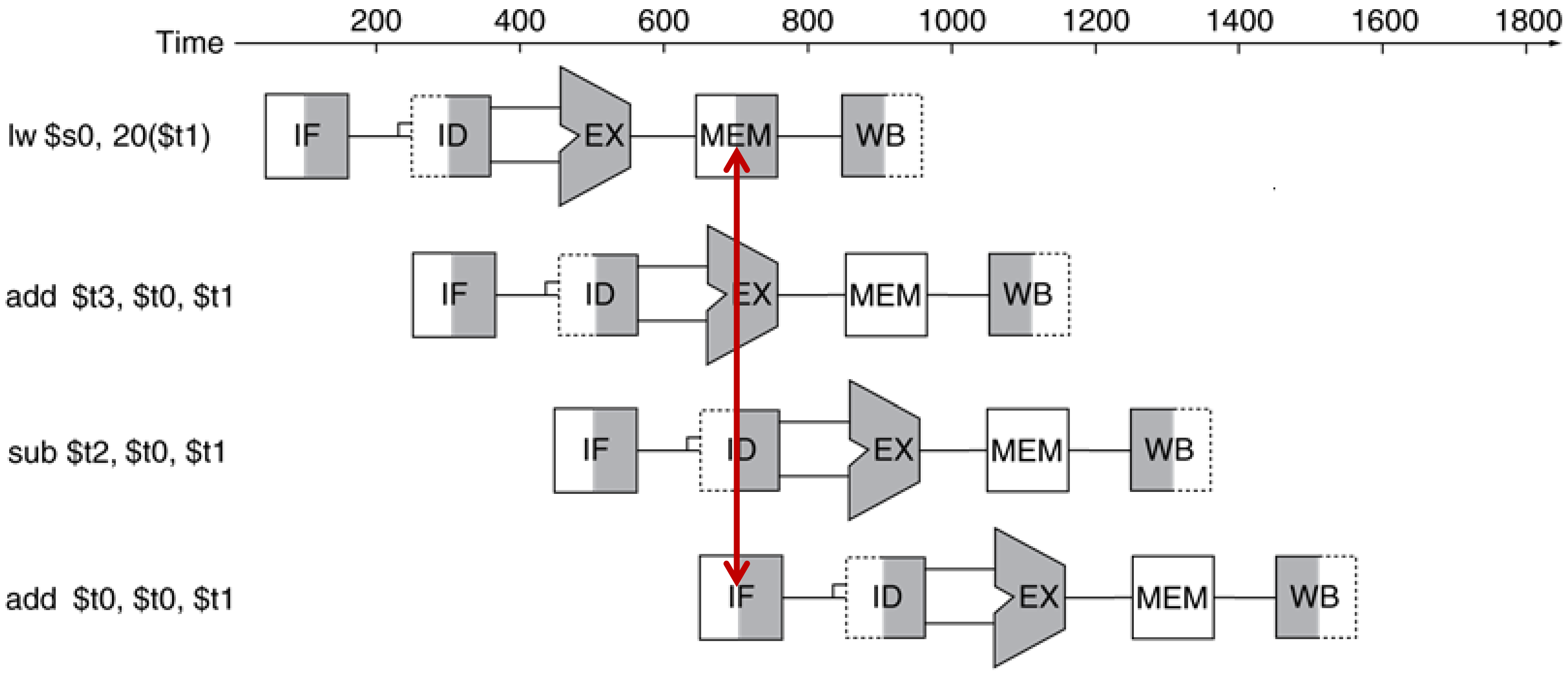


Structure Hazards

- Recall: A hardware structure required by an instruction is being used by another executing instruction
- The MIPS pipeline does not have structure hazards because:
  - It requires separate instruction and data memories
    - More precisely separate instruction and data caches
    - Otherwise, there could be a structure hazard between the IF and MEM stages on the memory structure
  - It allows reading to and writing from the register file on the same cycle
    - Otherwise, there could be a structure hazard between the ID and WB stages on the register file structure

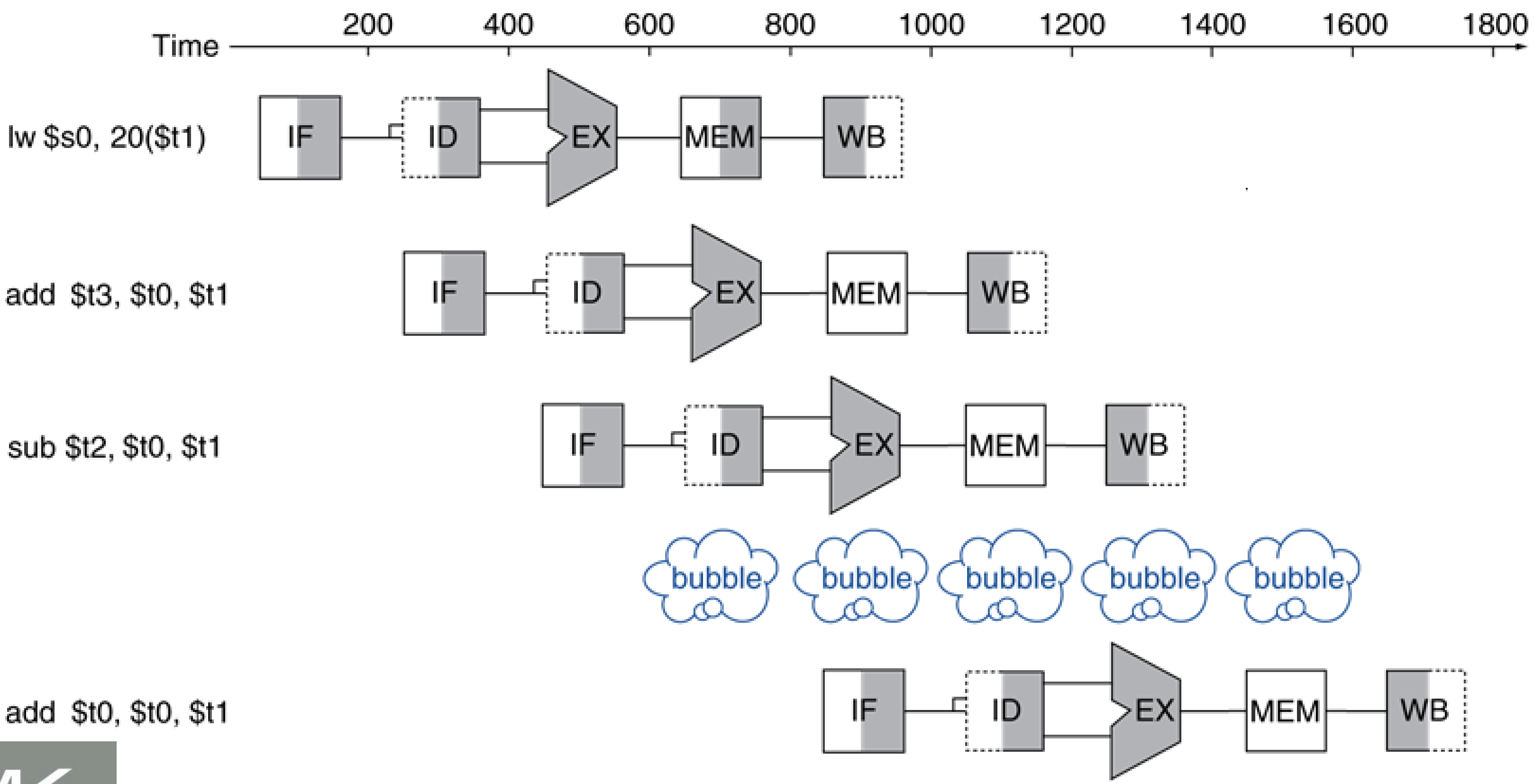
Structure Hazard Example

- Hypothetically, if the MIPS pipeline used a single memory for instructions and data, there could be a structure hazard between IF and MEM stages



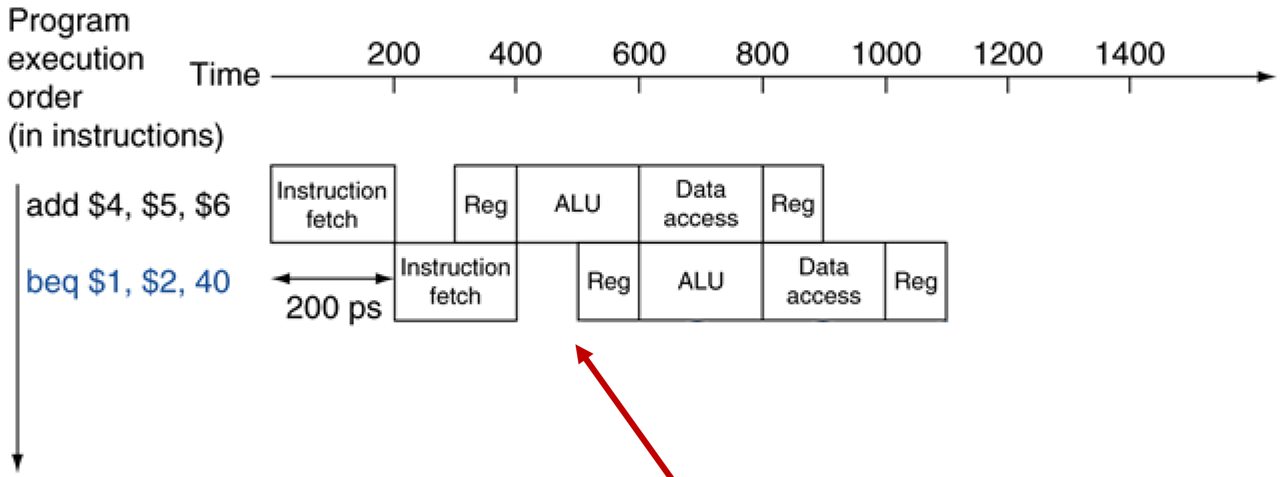
Structure Hazard Example

- Instruction fetch would have to *stall* when load/store instructions performs data access



Control Hazards

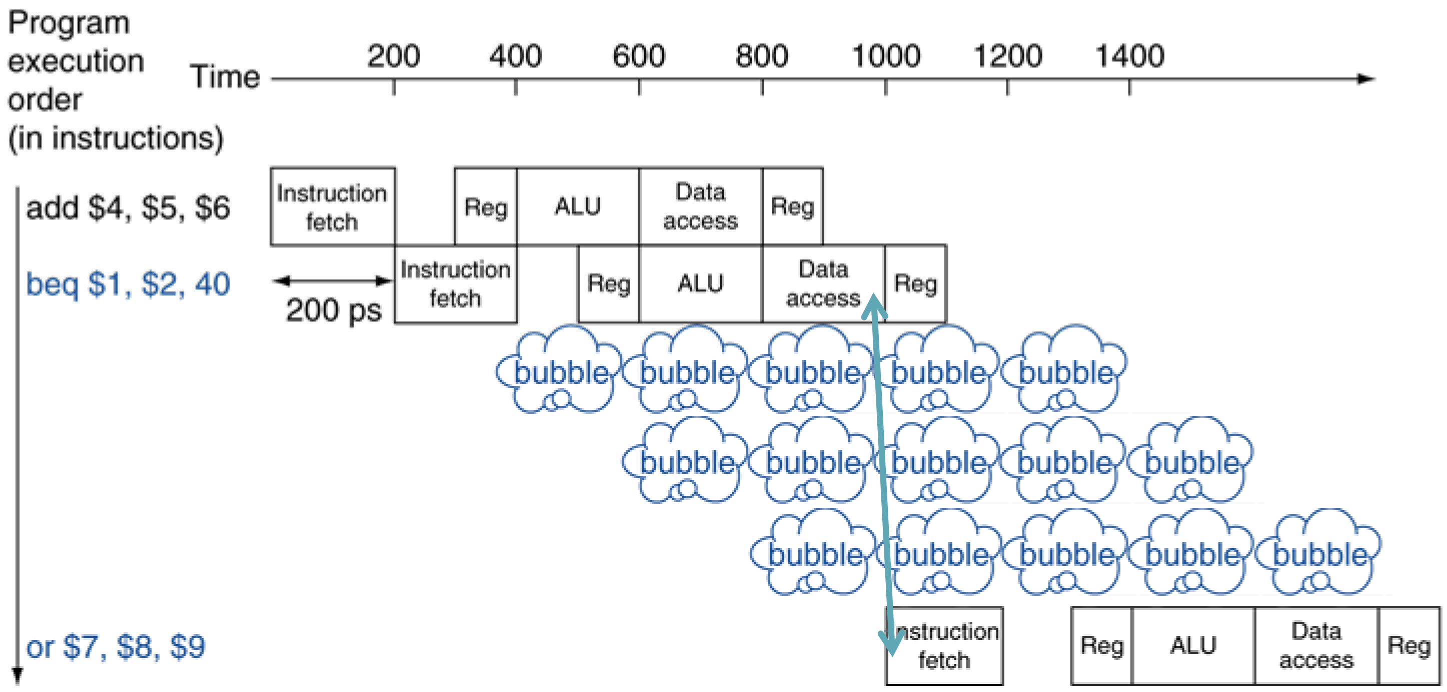
- Recall: Deciding the next instruction to fetch depends on the outcome of another executing instruction
    - e.g., branches



cannot fetch next instruction because branch outcome unknown

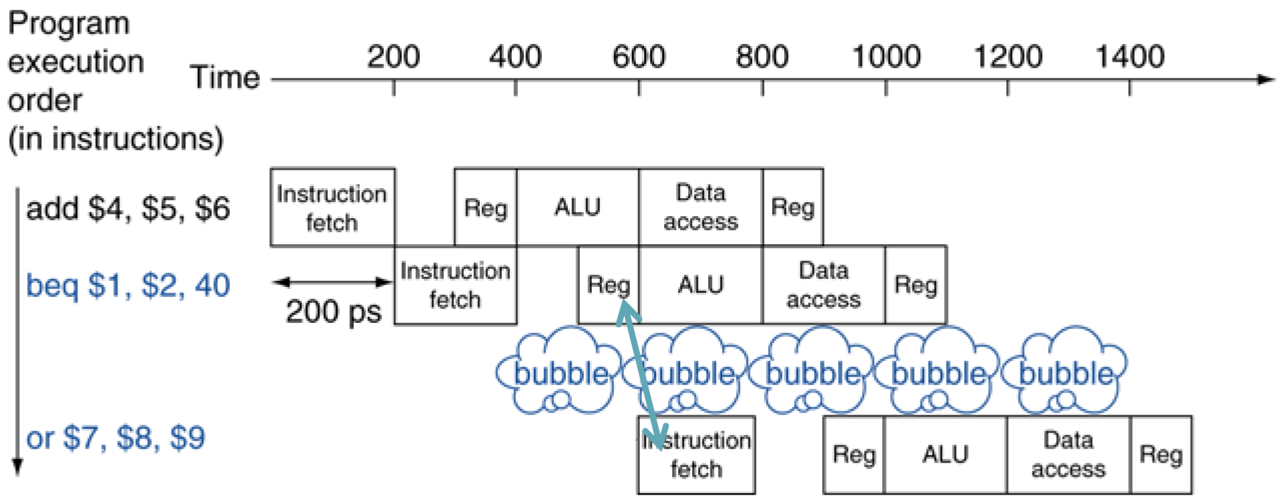
Stall on Branch

- Solution: stall until the branch outcome is determined before fetching the next instruction



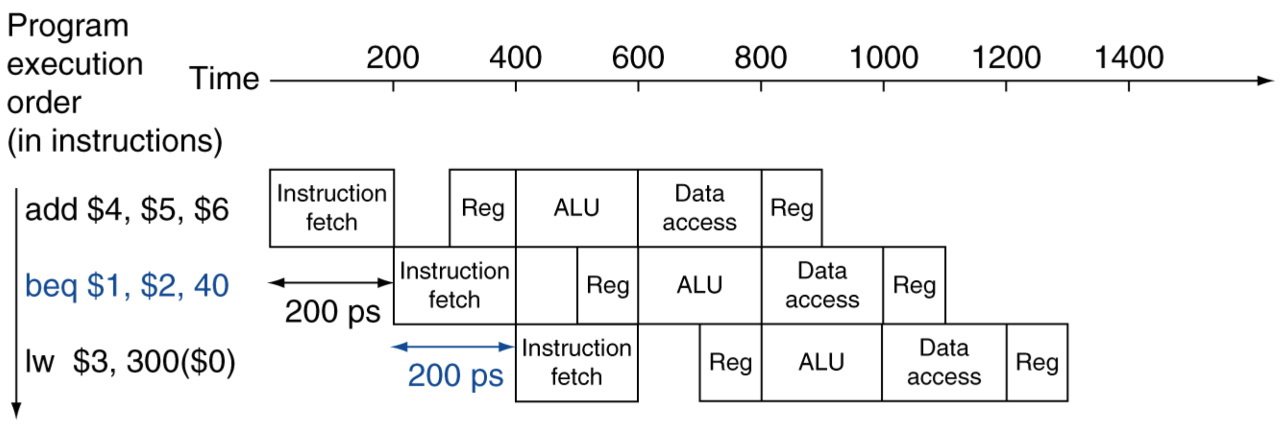
Resolve Branches Earlier

- Another solution: compare registers and compute target earlier in the pipeline
  - Add hardware to do it in ID stage instead of EX stage

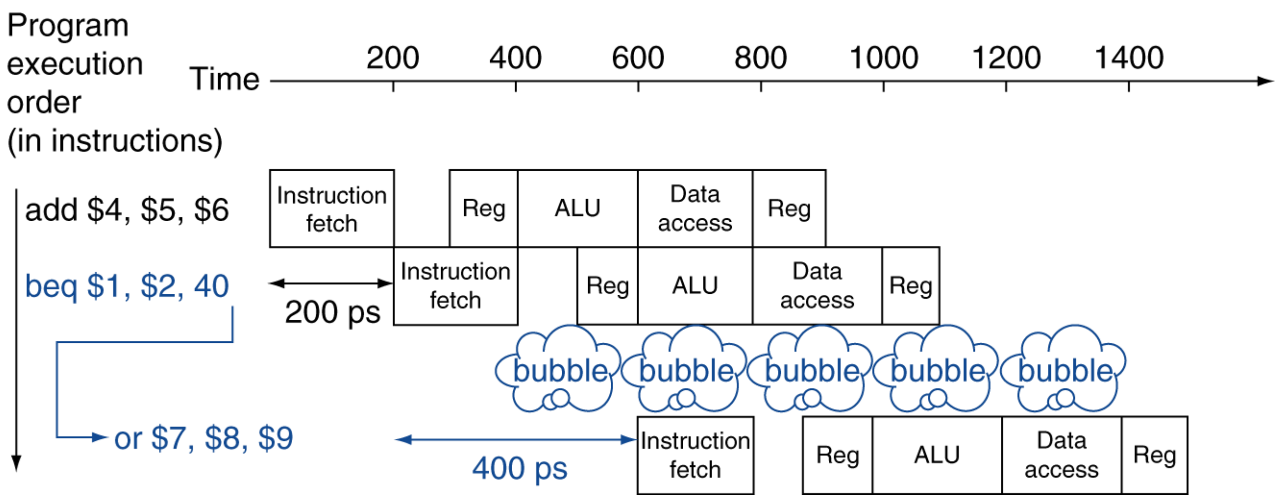


MIPS with Predict Not Taken

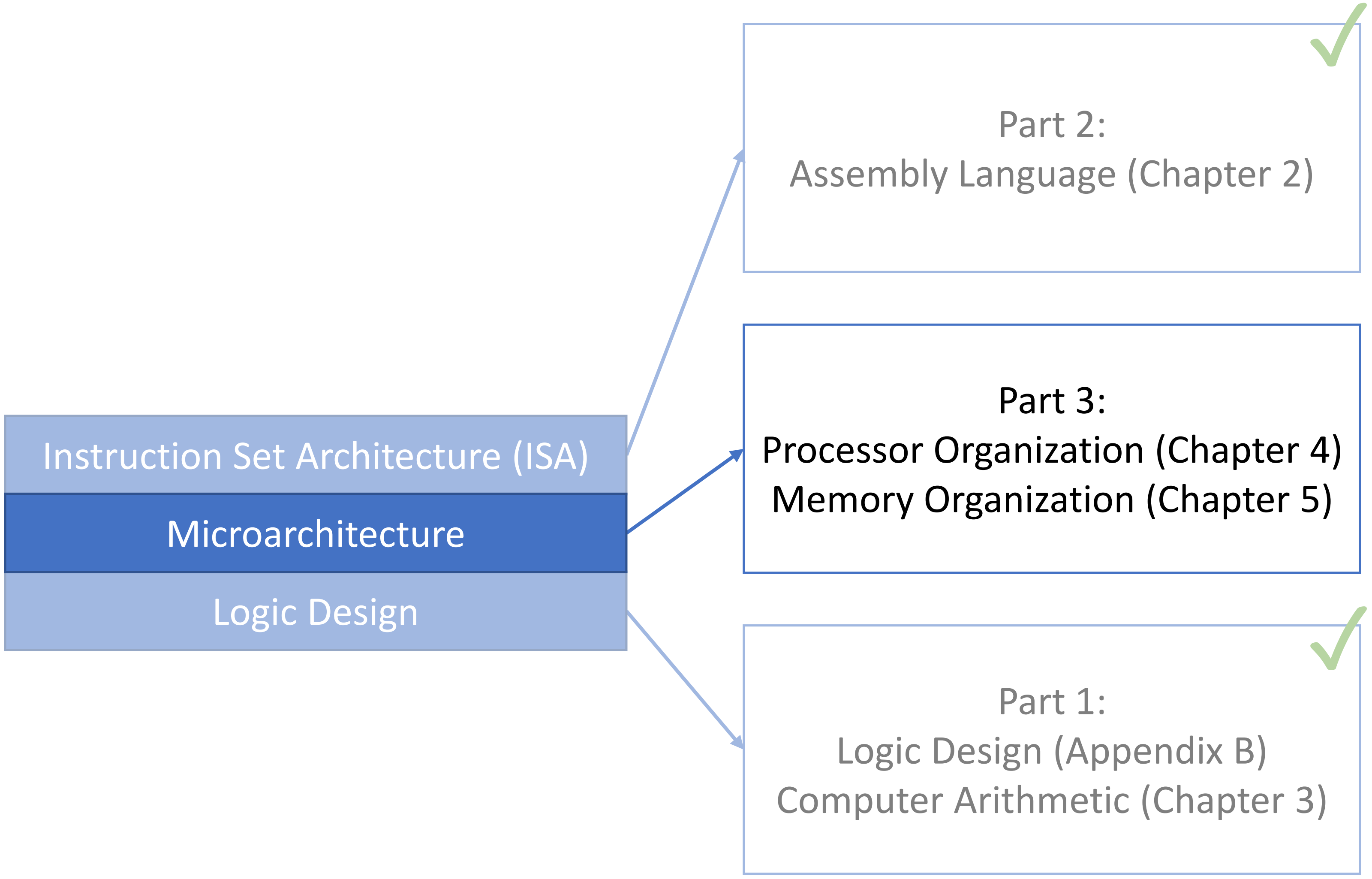
Prediction correct



Prediction incorrect



Today: Forwarding


Instruction Set Architecture (ISA)
Microarchitecture
Logic Design
✓
Part 2:
Assembly Language (Chapter 2)
Part 3:
Processor Organization (Chapter 4)
Memory Organization (Chapter 5)
✓
Part 1:
Logic Design (Appendix B)
Computer Arithmetic (Chapter 3)

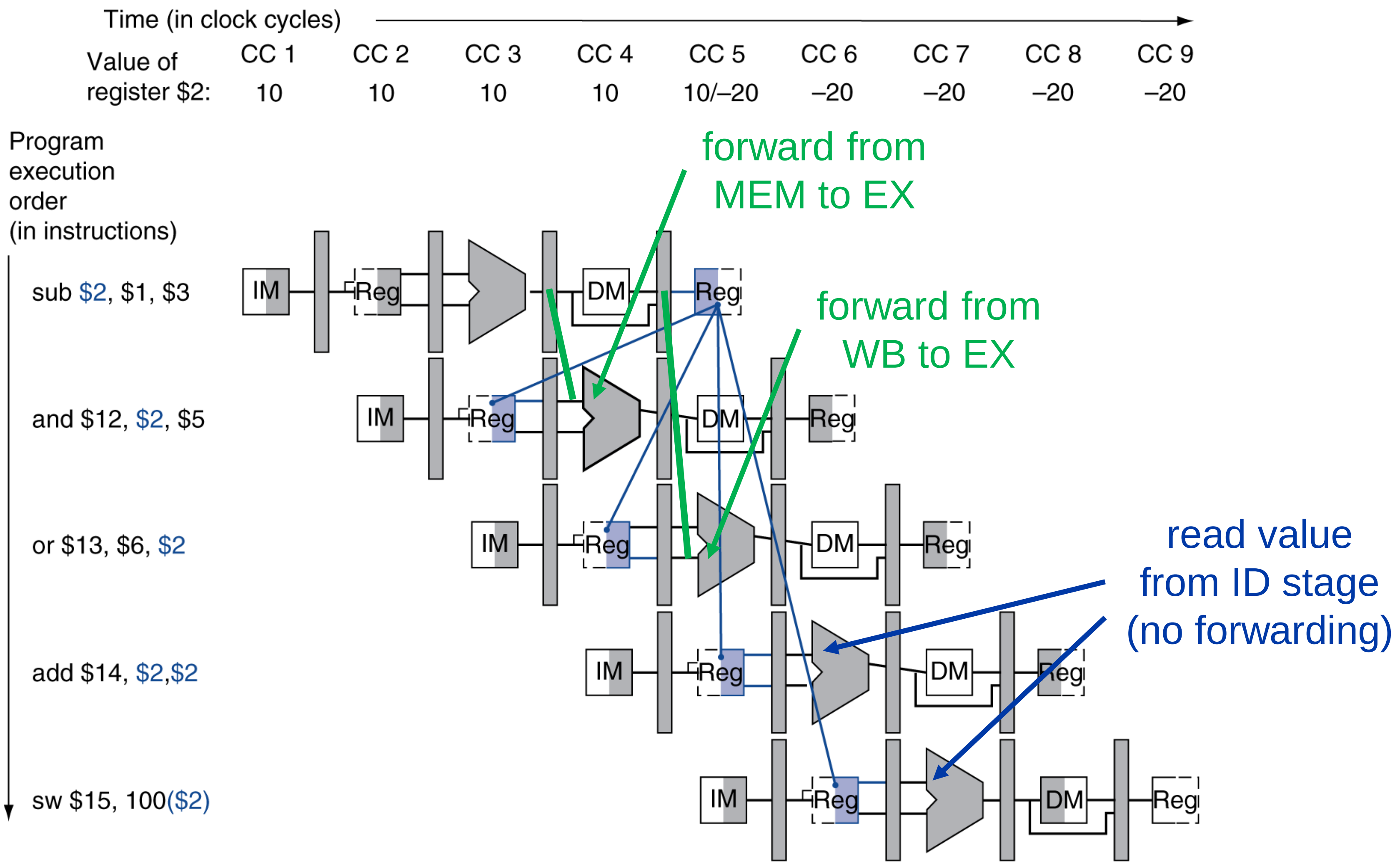
Data Hazards in ALU Instructions

- Consider the following sequence:

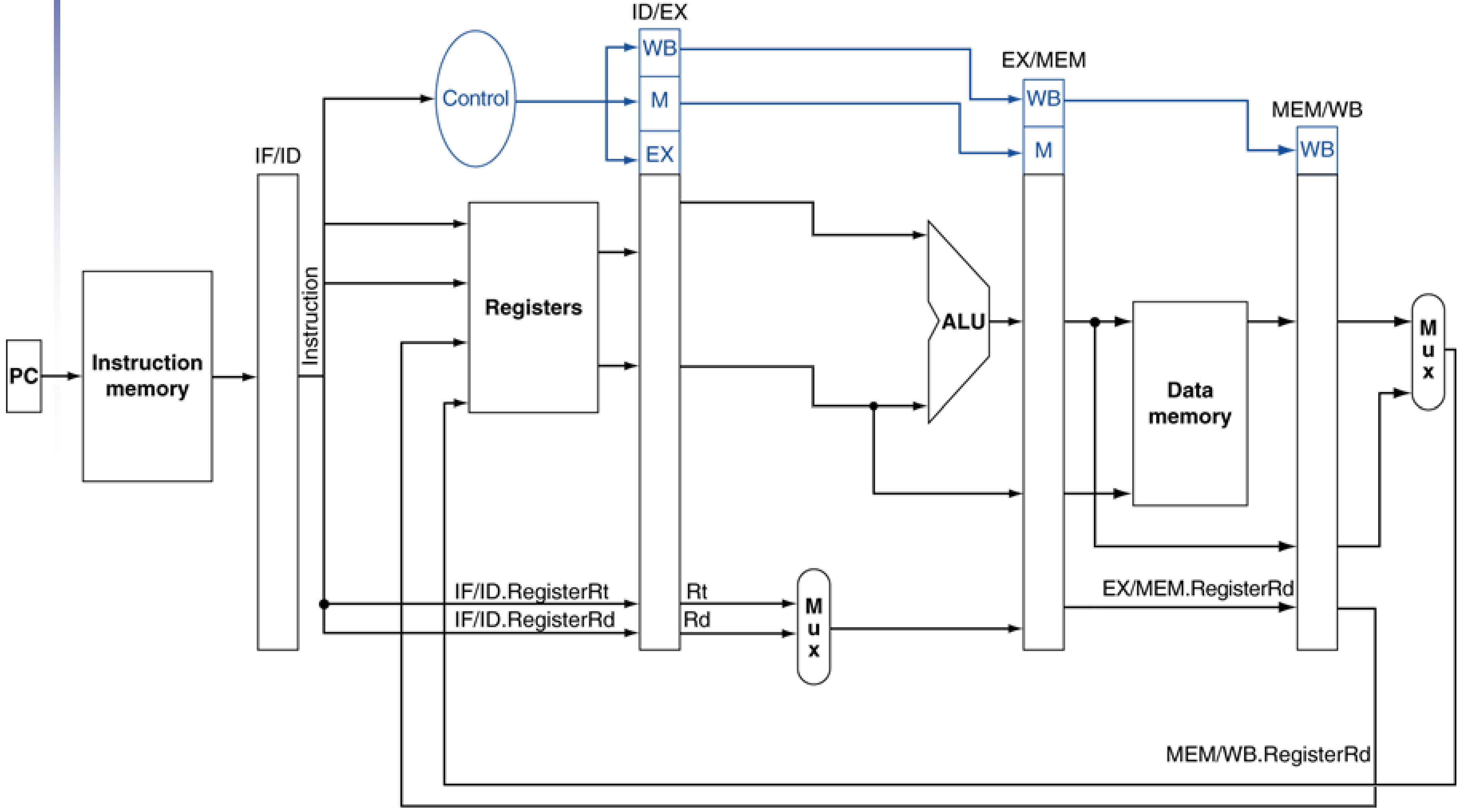
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- We can resolve hazards with forwarding
  - How do we detect when to forward?

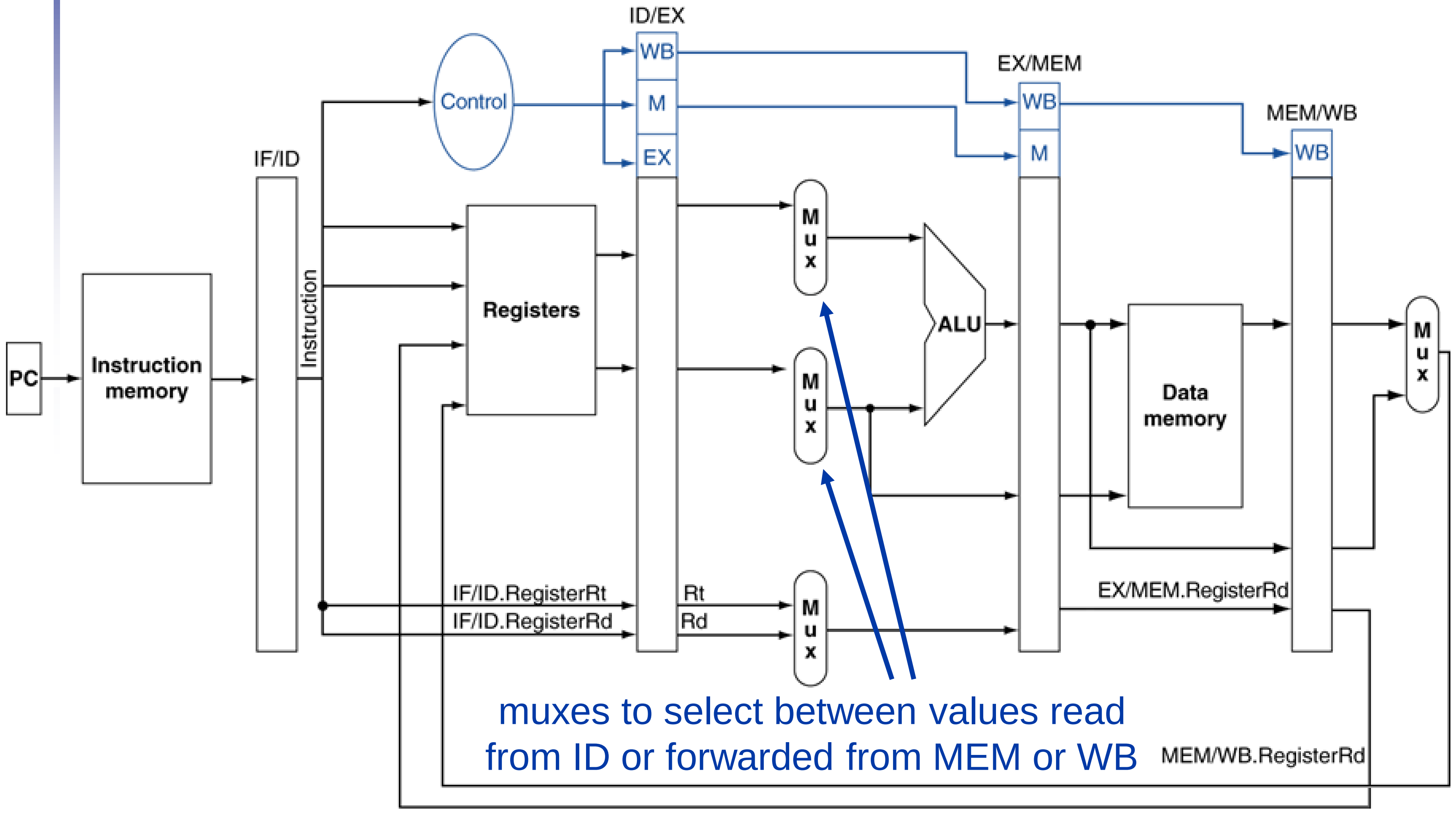
Dependencies & Forwarding



Recall: Pipelined Datapath

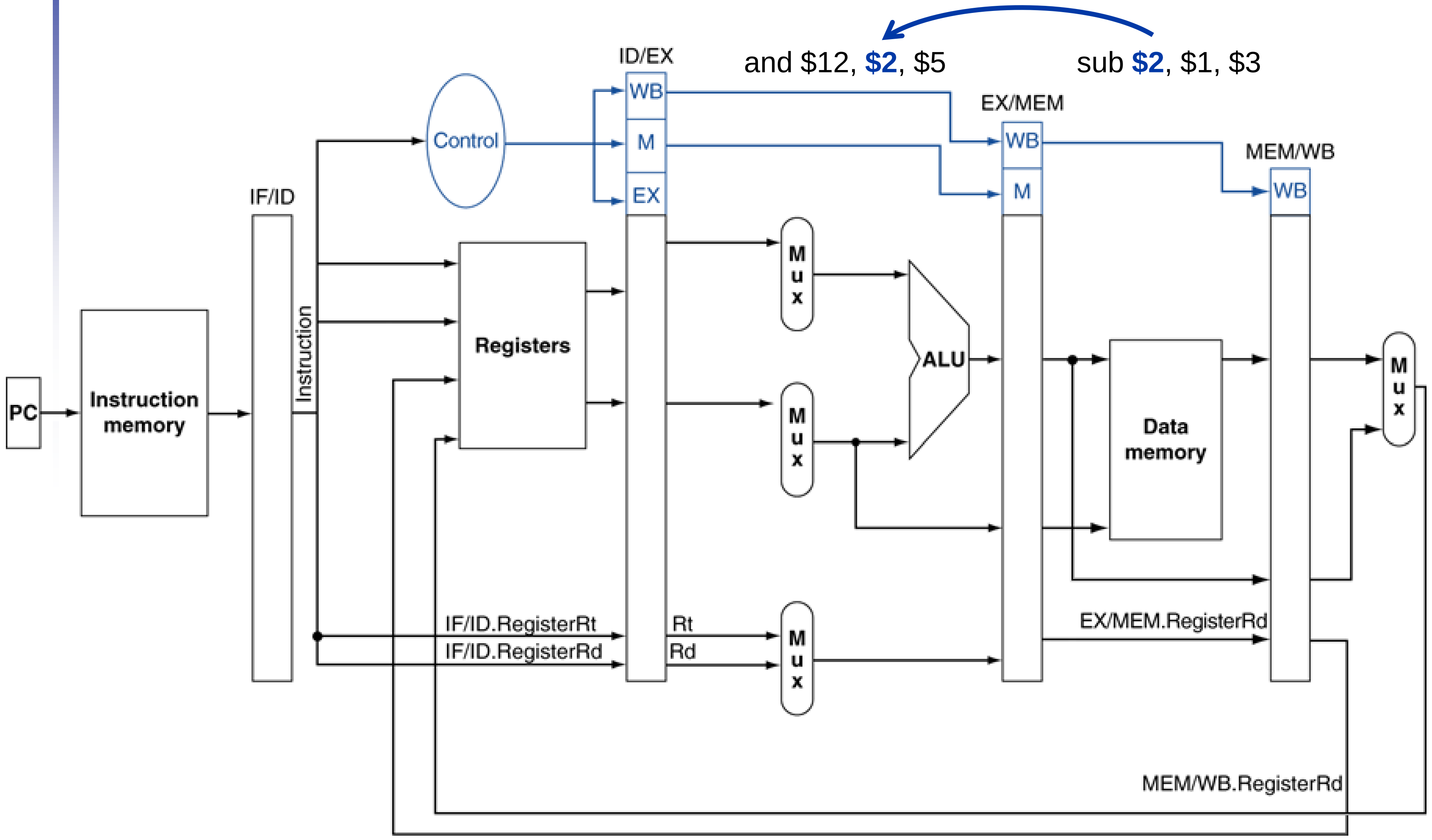


Datapath with Forwarding

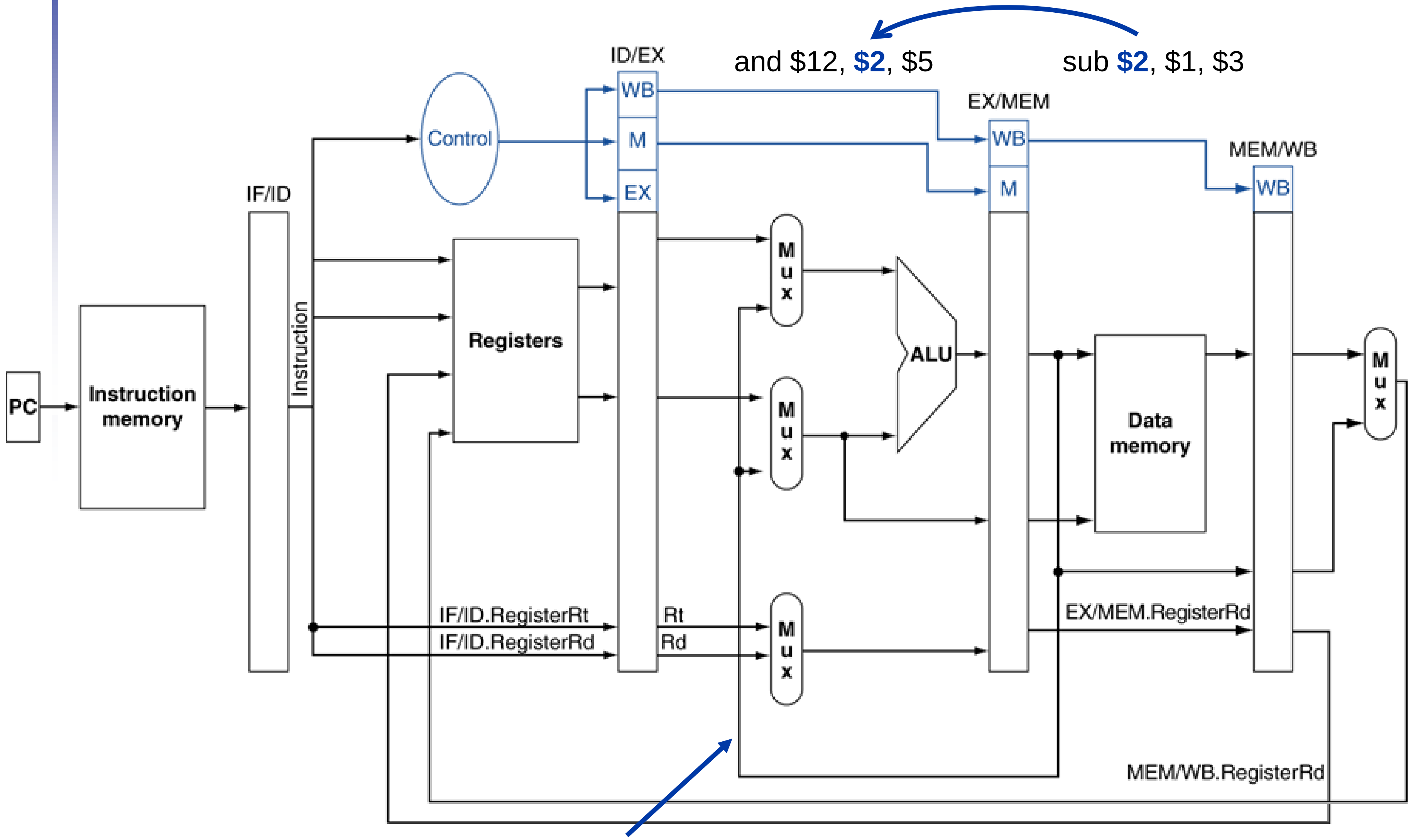


muxes to select between values read from ID or forwarded from MEM or WB

Forwarding from MEM Stage

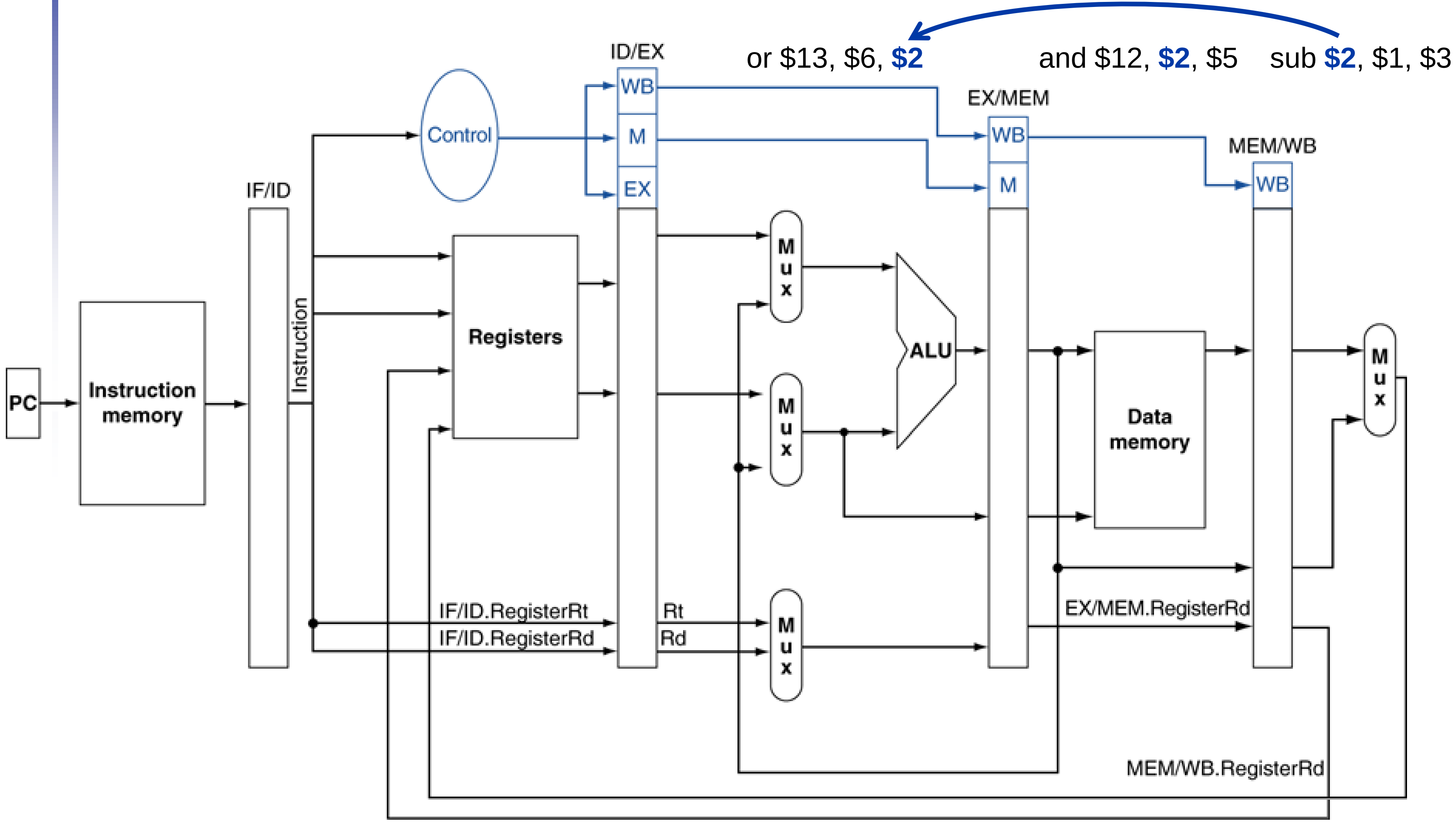


Forwarding from MEM Stage

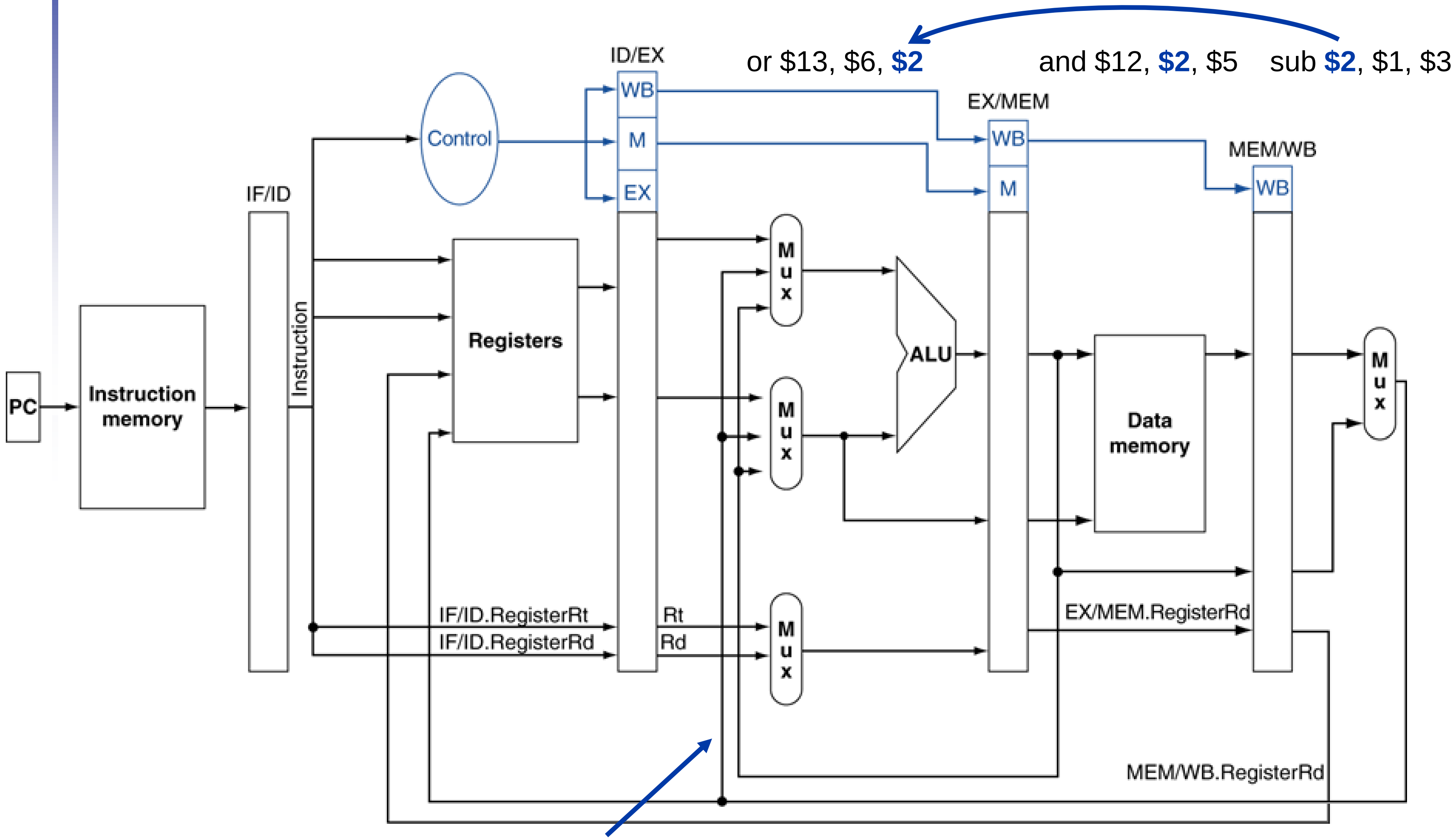


forward from MEM stage

Forwarding from WB Stage

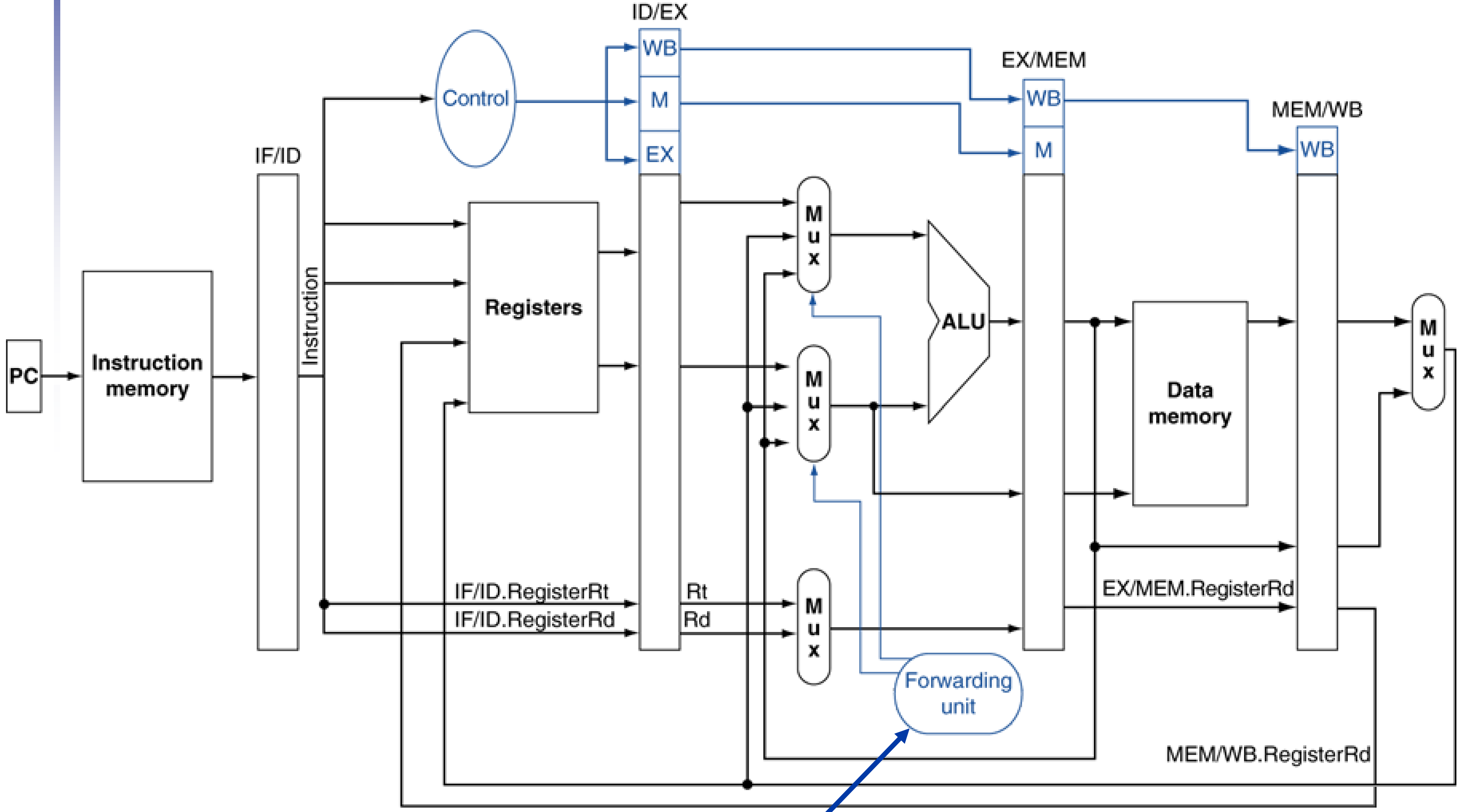


Forwarding from WB Stage



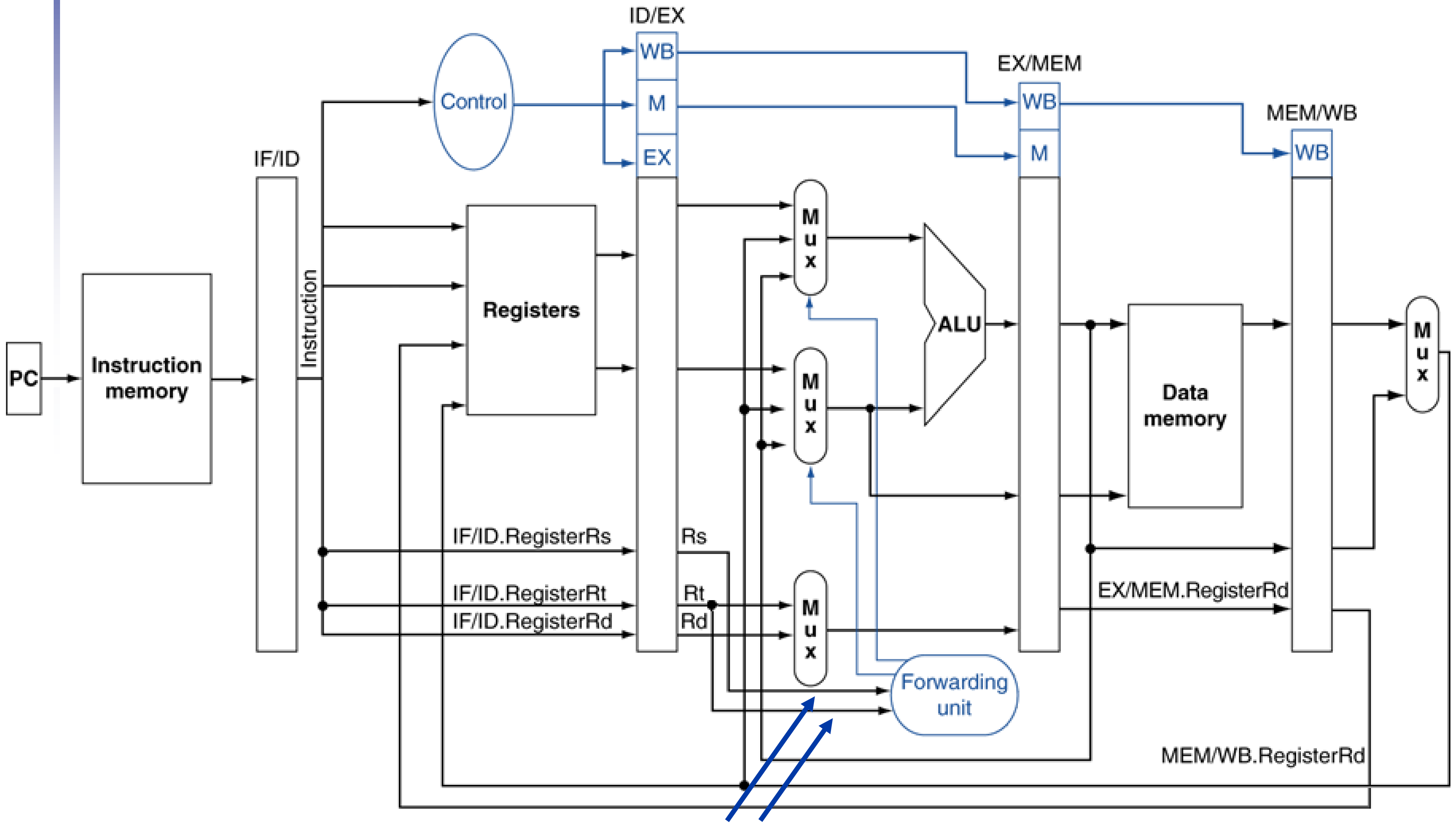
forward from WB stage

Forwarding Unit



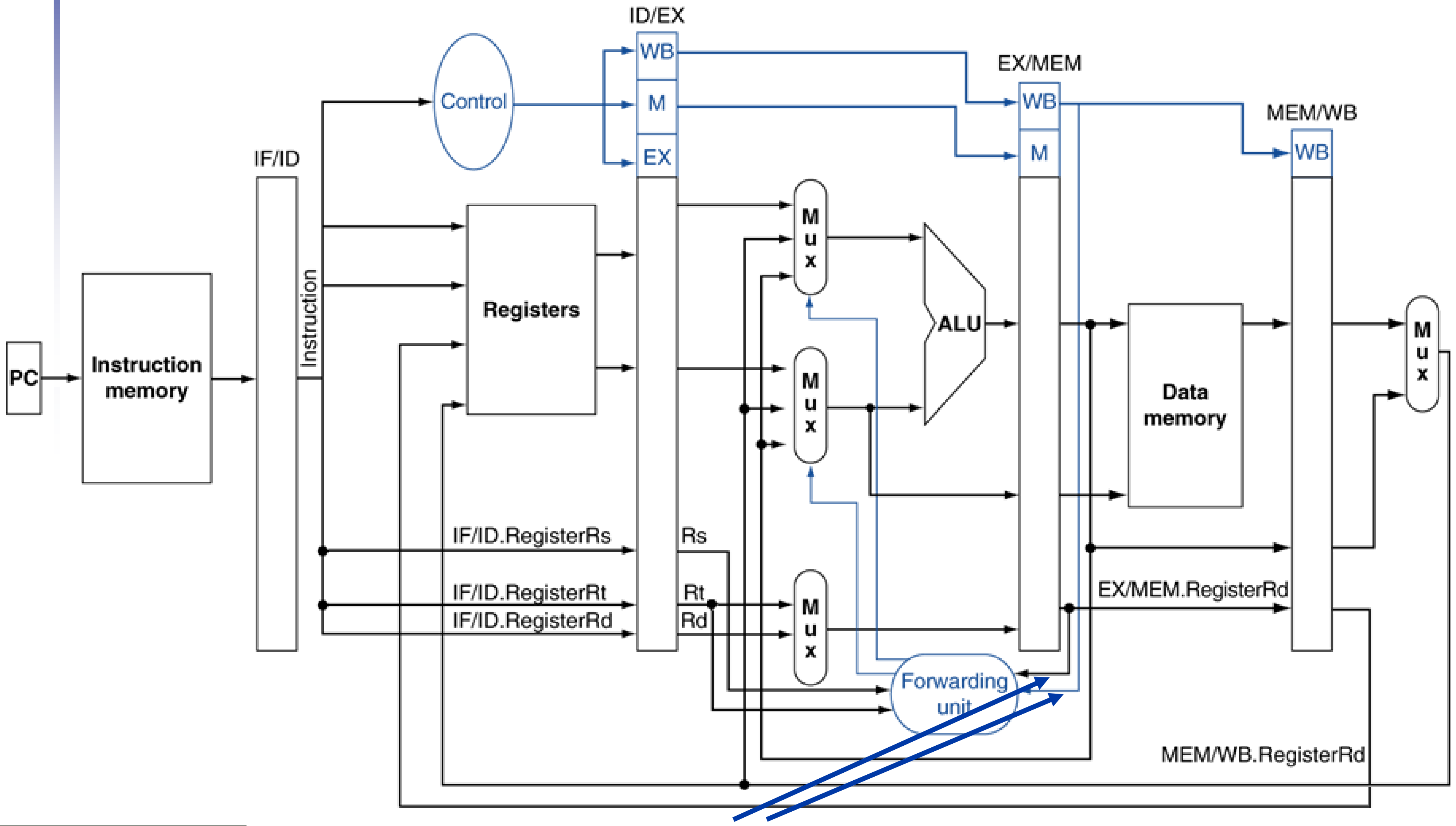
Forward unit to control the muxes

Forwarding Unit



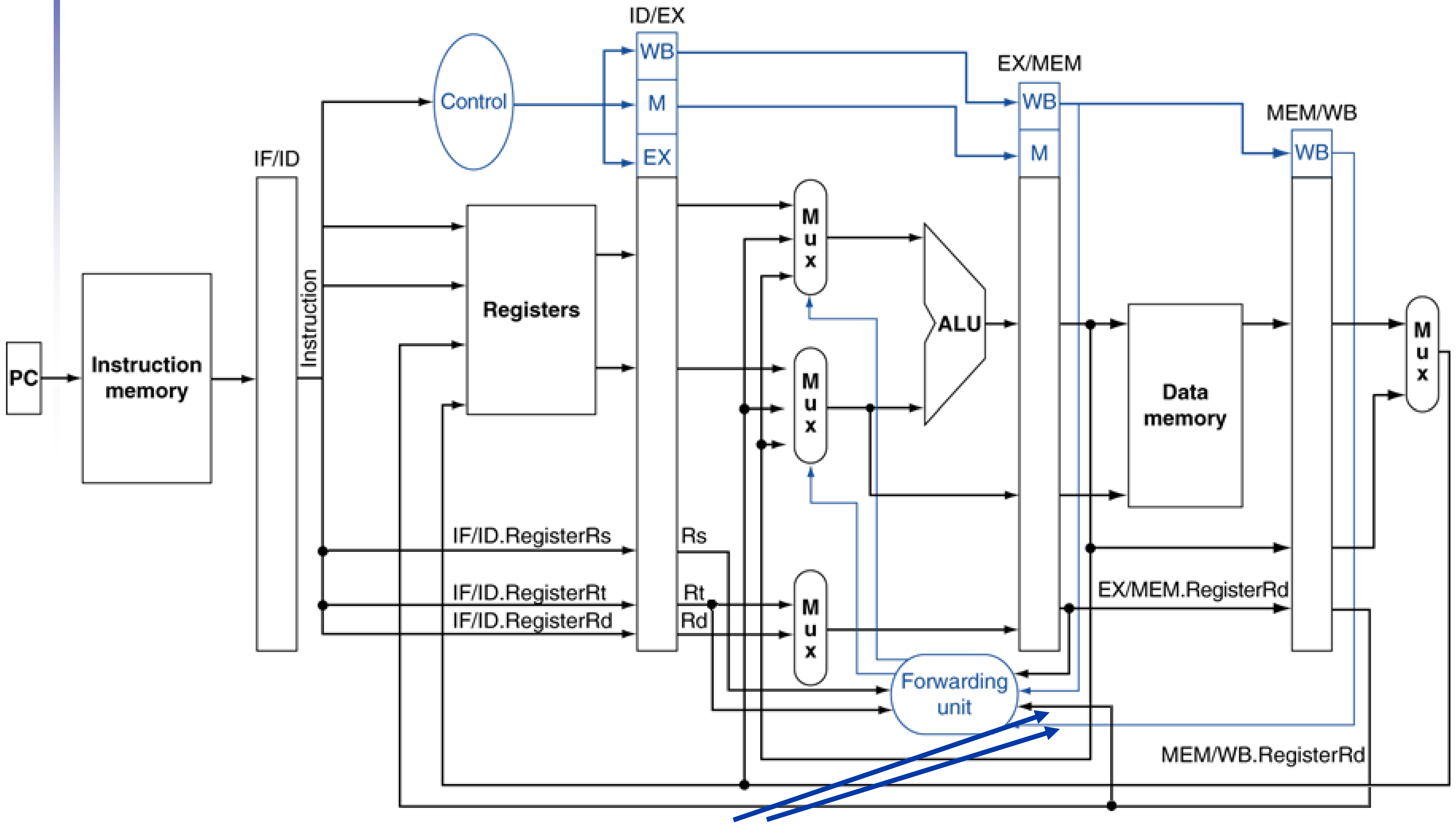
needs to know source registers of instruction in EX

Forwarding Unit



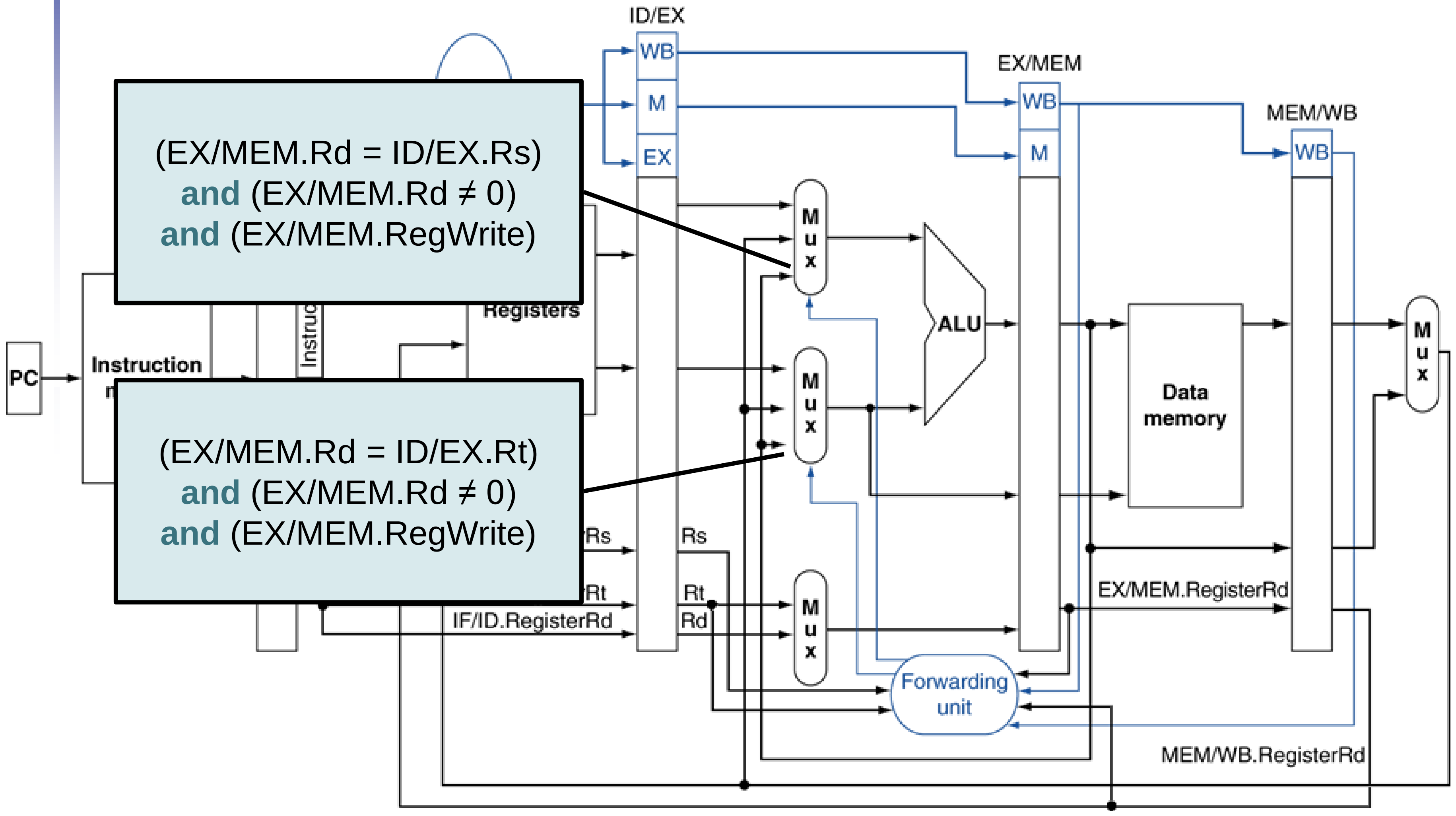
destination register and
RegWrite of instruction in MEM

Forwarding Unit



destination register and RegWrite of instruction in WB

Forwarding Conditions

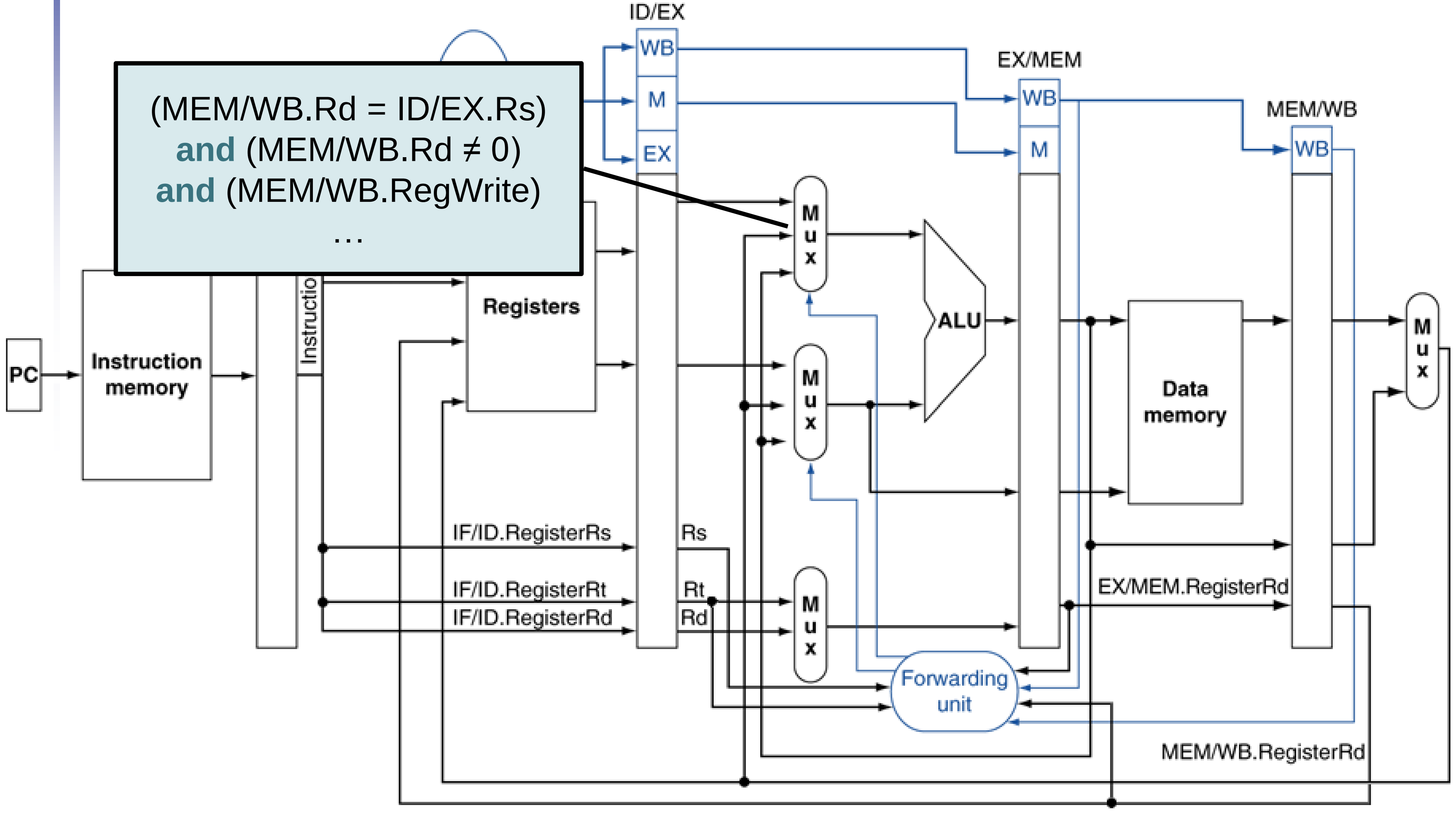


(EX/MEM.Rd = ID/EX.Rs)
and (EX/MEM.Rd ≠ 0)
and (EX/MEM.RegWrite)

(EX/MEM.Rd = ID/EX.Rt)
and (EX/MEM.Rd ≠ 0)
and (EX/MEM.RegWrite)

Forwarding Conditions

(MEM/WB.Rd = ID/EX.Rs)
and (MEM/WB.Rd ≠ 0)
and (MEM/WB.RegWrite)
…



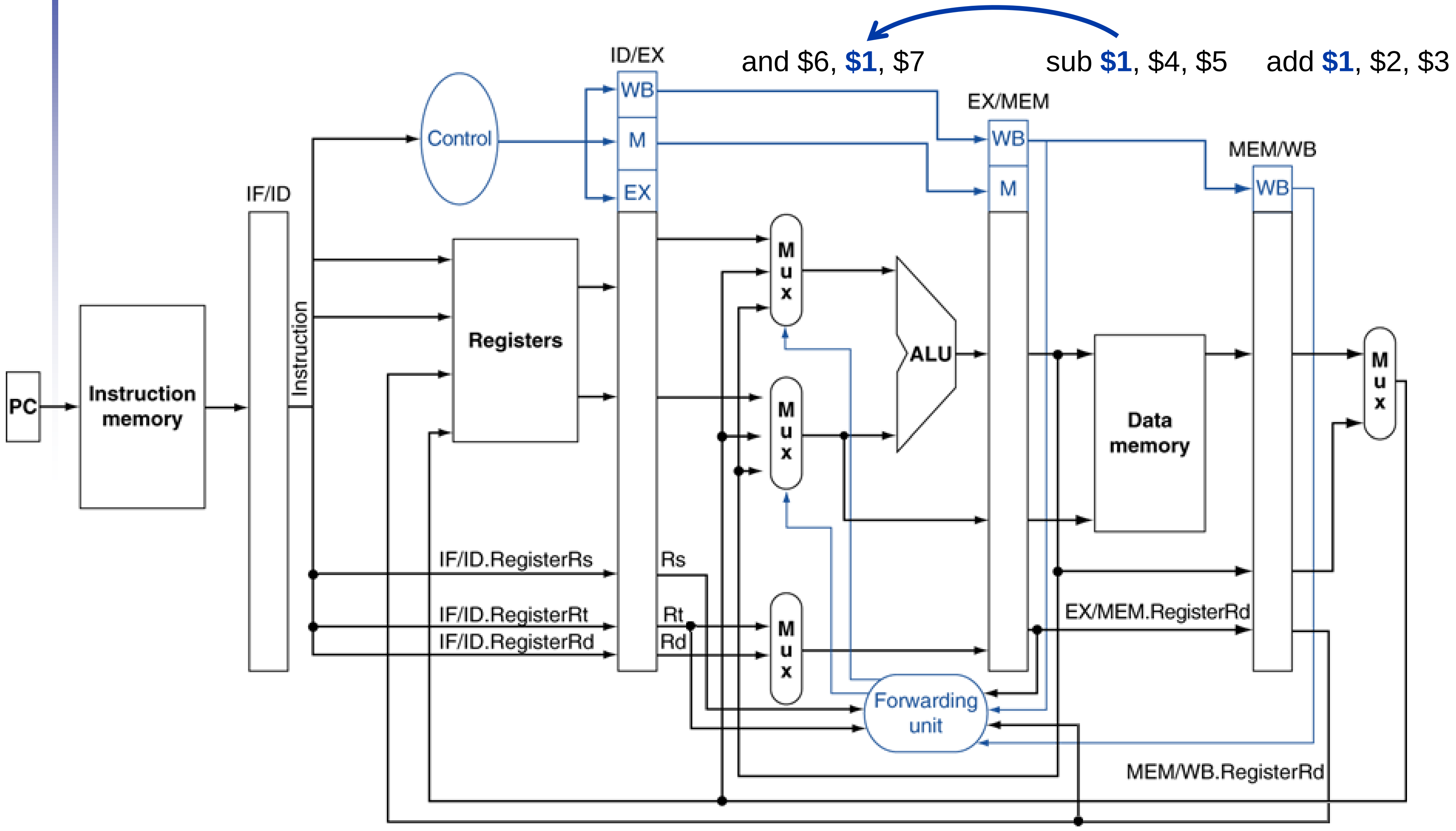
Double Data Hazard

- Consider the sequence:

```
add $1,$2,$3
sub $1,$4,$5
and $6,$1,$7
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM/WB hazard condition to only forward if EX/MEM hazard condition is false

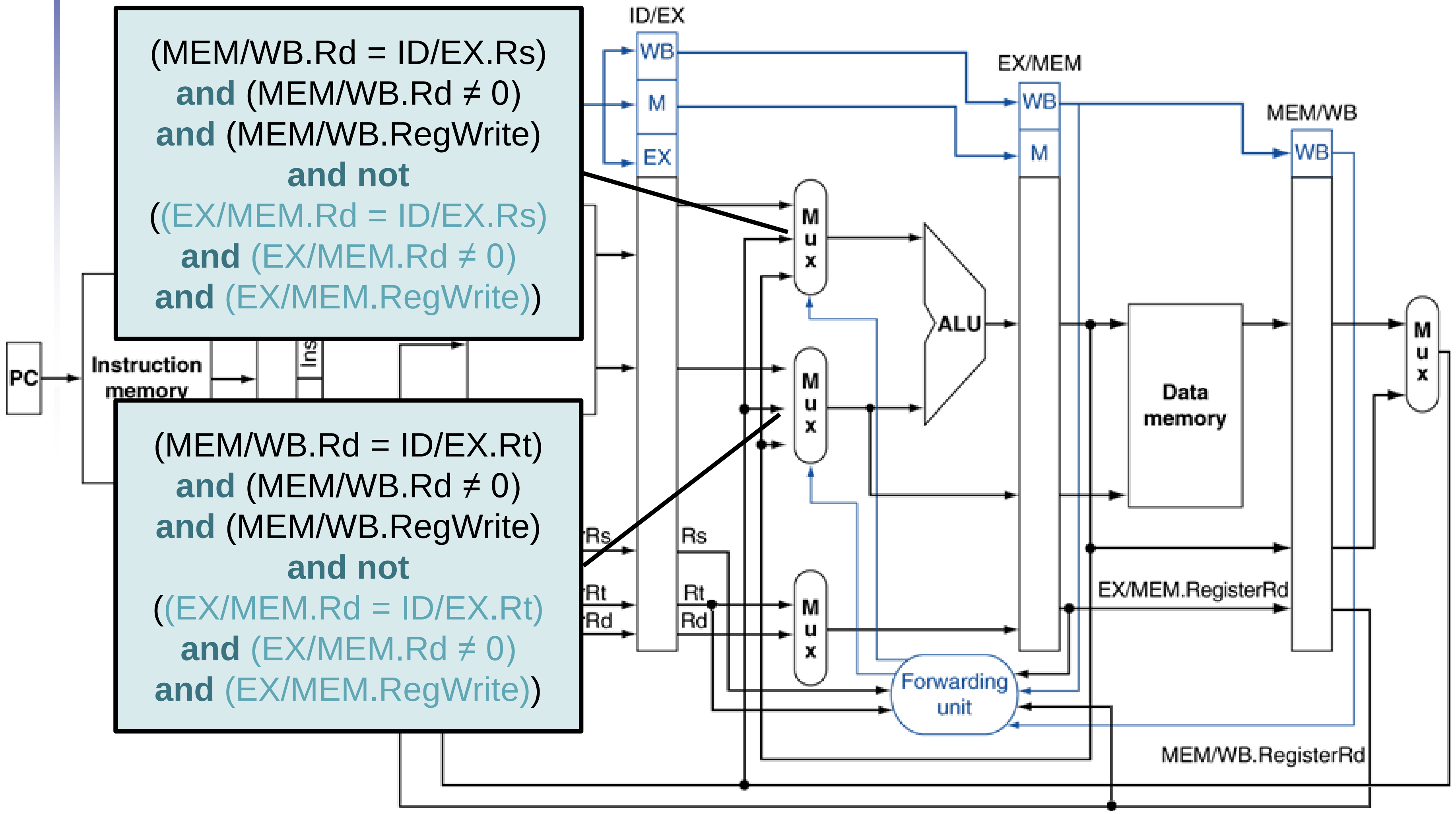
Double Data Hazard



Forwarding Conditions

(MEM/WB.Rd = ID/EX.Rs)
and (MEM/WB.Rd ≠ 0)
and (MEM/WB.RegWrite)
and not
((EX/MEM.Rd = ID/EX.Rs)
and (EX/MEM.Rd ≠ 0)
and (EX/MEM.RegWrite))

(MEM/WB.Rd = ID/EX.Rt)
and (MEM/WB.Rd ≠ 0)
and (MEM/WB.RegWrite)
and not
((EX/MEM.Rd = ID/EX.Rt)
and (EX/MEM.Rd ≠ 0)
and (EX/MEM.RegWrite))



Textbook Sections

- The content in these slides corresponds to:
  - Textbook:
    - *Computer Organization and Design, 5th Edition by David Patterson and John Hennessy, Morgan Kaufmann, 2014.*
  - Sections:
    - 4.7