CMPE110 Lecture 12
Pipelining 2

Heiner Litz

https://canvas.ucsc.edu/courses/19290

Announcements

- **Evaluation of the Tensor Processing Unit: A Deep Neural Network Accelerator for the Datacenter**
- David Patterson, UC Berkeley & Google
- Friday, February 8, 2:40 PM, E2-180

Announcements

- HA3 is released due: Friday 02/09
- Midterm: Wed 02/13
  - 1 double sided page of notes
  - RISC-V Green card/cheat sheet part of the exam

Review

Pipeline Hazards

- Situations that prevent completing an instruction every cycle
  - Lead to CPI > 1
- Structure hazards
  - A required resource is busy
- Data hazard
  - Must wait previous instructions to produce/consume data
- Control hazard
  - Next PC depends on previous instruction

Dependency Examples

- True dependency => RAW hazard

```
addu      xt0, xt1, xt2
subu      xt3, xt4, xt0
```

- Output dependency => WAW hazard

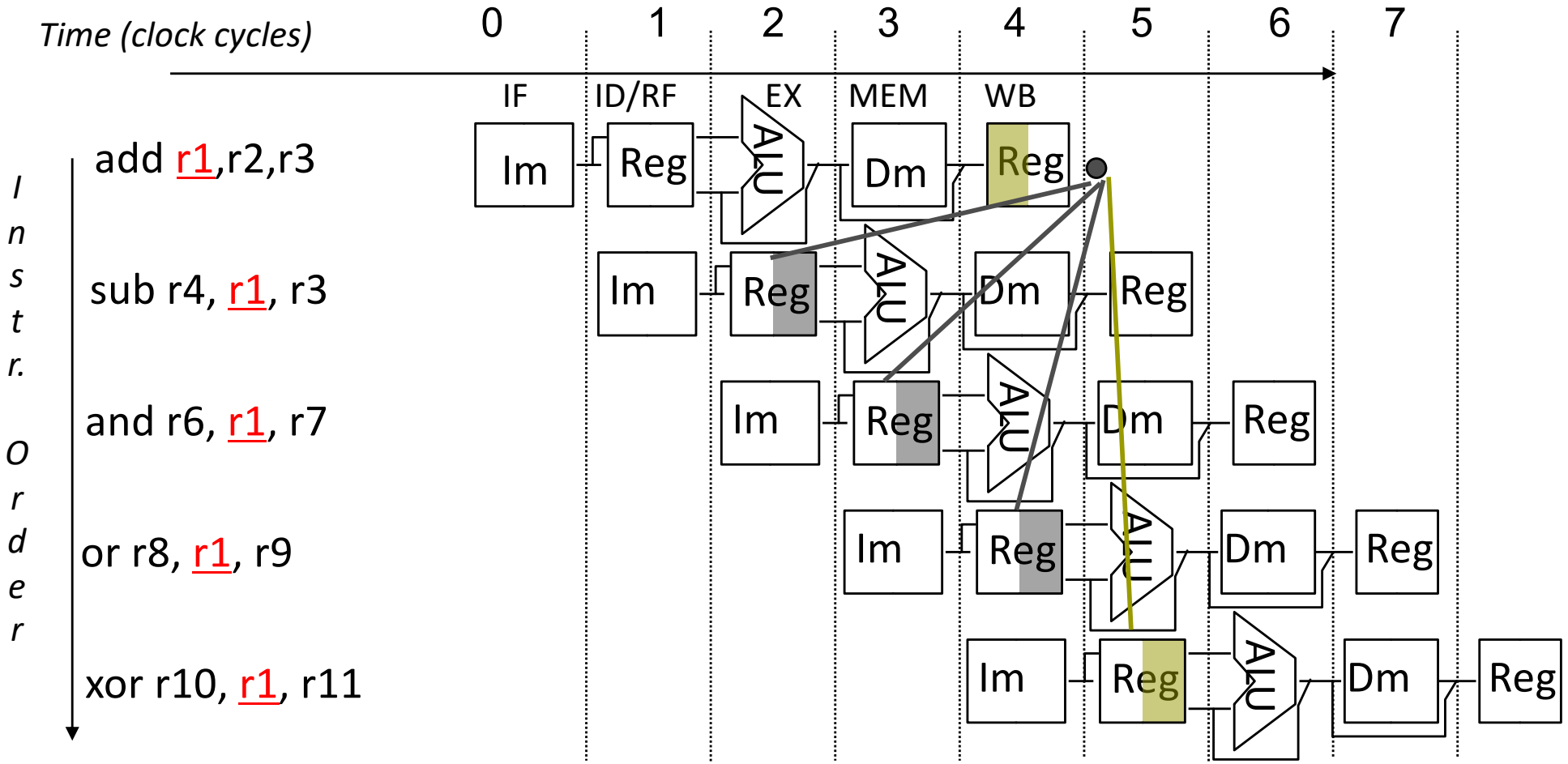
```
addu      xt0, xt1, xt2
subu      xt0, xt4, xt5
```

- Anti dependency => WAR hazard

```
addu      xt0, xt1, xt2
subu      xt1, xt4, xt5
```

RAW Hazard Example

- Dependencies backwards in time are hazards

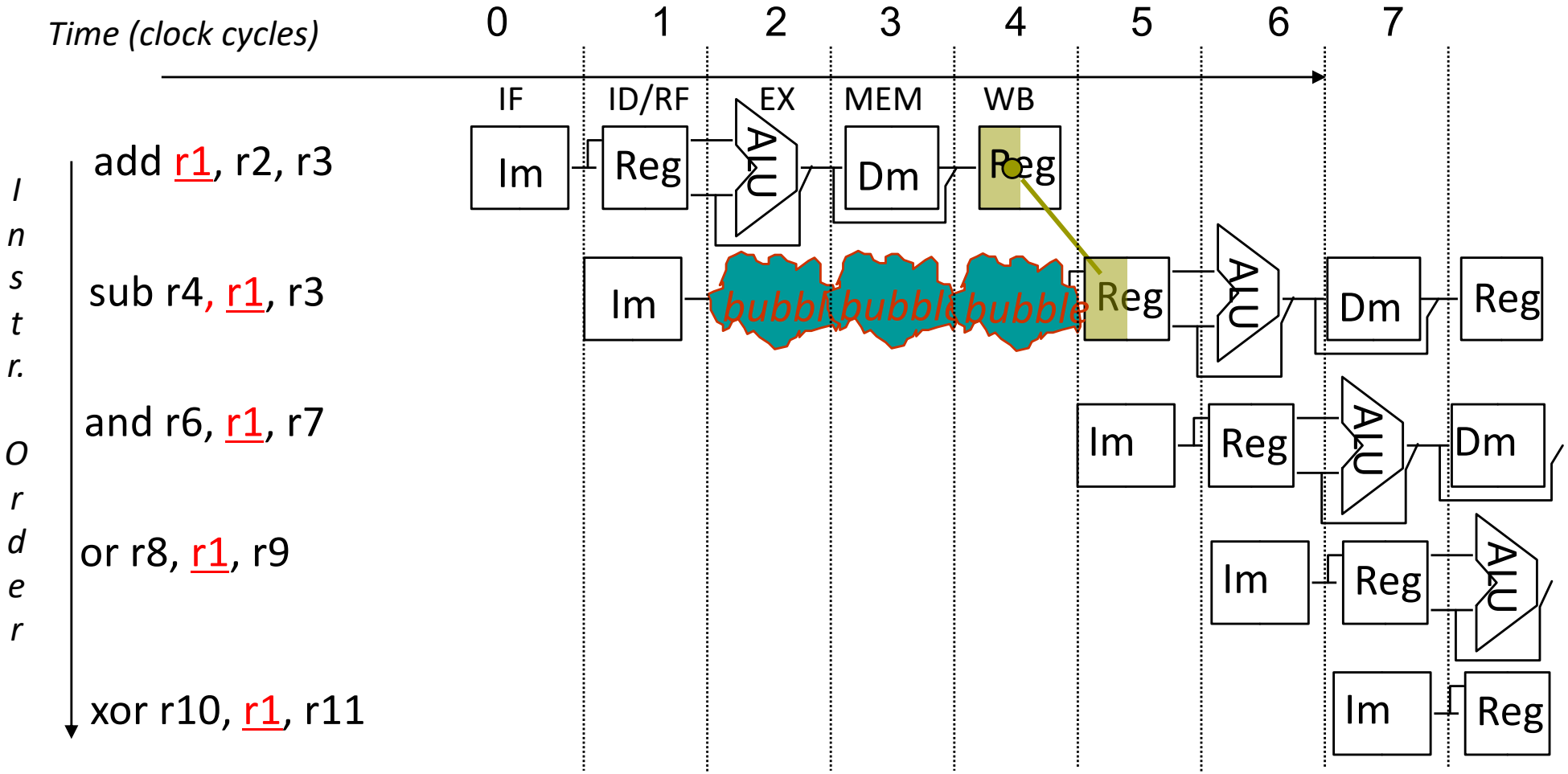


Solutions for RAW Hazards

- Delay the reading instruction until data is available
  - Also called stalling or inserting pipeline bubbles
- How can we delay the younger instruction?
  - Compiler insert independent work or NOPS ahead of it
    - NOP example: or $0, $0, $0
    - Disadvantage: pipeline-specific binary program
  - Hardware inserts NOPs as needed (interlocks)
    - Advantage: correct operation for all programs/pipelines
    - Disadvantage: may miss some optimization opportunities
  - Most modern machines
    - Hardware inserts NOPs but compiler may try to minimize need

Data Hazard - Stalls

- Eliminate reverse time dependency by stalling



How to Stall the Pipeline

- Discover need to stall when 2nd instruction is in ID stage
  - Repeat its ID stage until hazard resolved
  - Let all instructions ahead of it move forward
  - Stall all instructions behind it

1. Force control values in ID/EX register a NOP instruction
   - As if you fetched or $0, $0, $0
   - When it propagates to EX, MEM and WB, nothing will happen
2. Prevent update of PC and IF/ID register
   - Using instruction is decoded again
   - Following instruction is fetched again

Performance Effect

- Stalls can have a significant effect on performance
- Consider the following case
  - The ideal CPI of the machine is 1
  - A RAW hazard causes a 3 cycle stall
- If 40% of the instructions cause a stall?
  - The new effective CPI is $1 + 3 \times 0.4 = 2.2$
  - And the real % is probably higher than 40%
- You get less than ½ the desired performance!

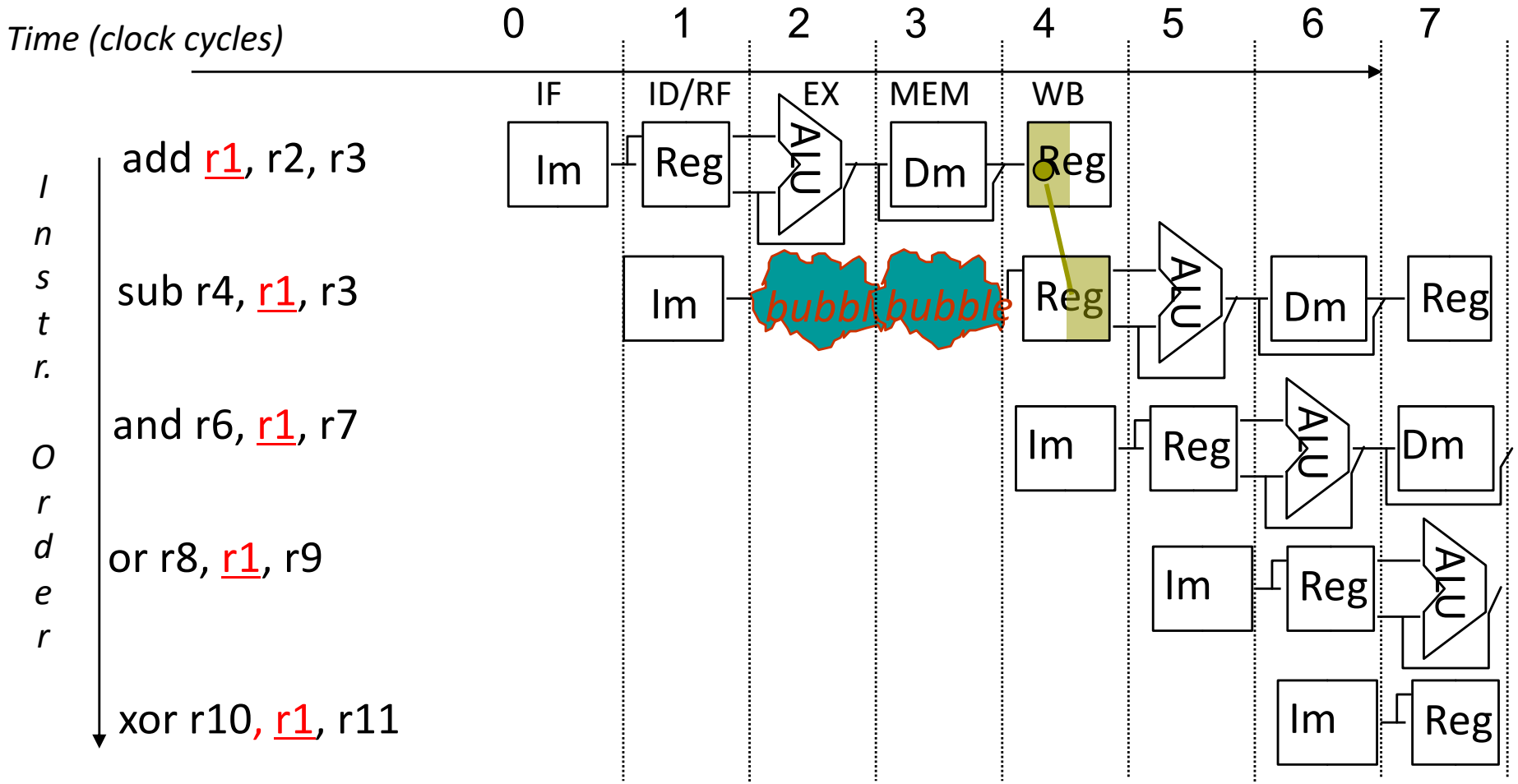
Reducing Stalls

- Key: when you say new data is actually available?

- In the 5-stage pipeline
  - After WB stage?
  - During WB stage?
    - Register file is typically fast
    - Write in the first half, read in the second half
  - After EX stage?

Decreasing Stalls: Fast RF

- Register file writes on first half and reads on second half



Performance Effect

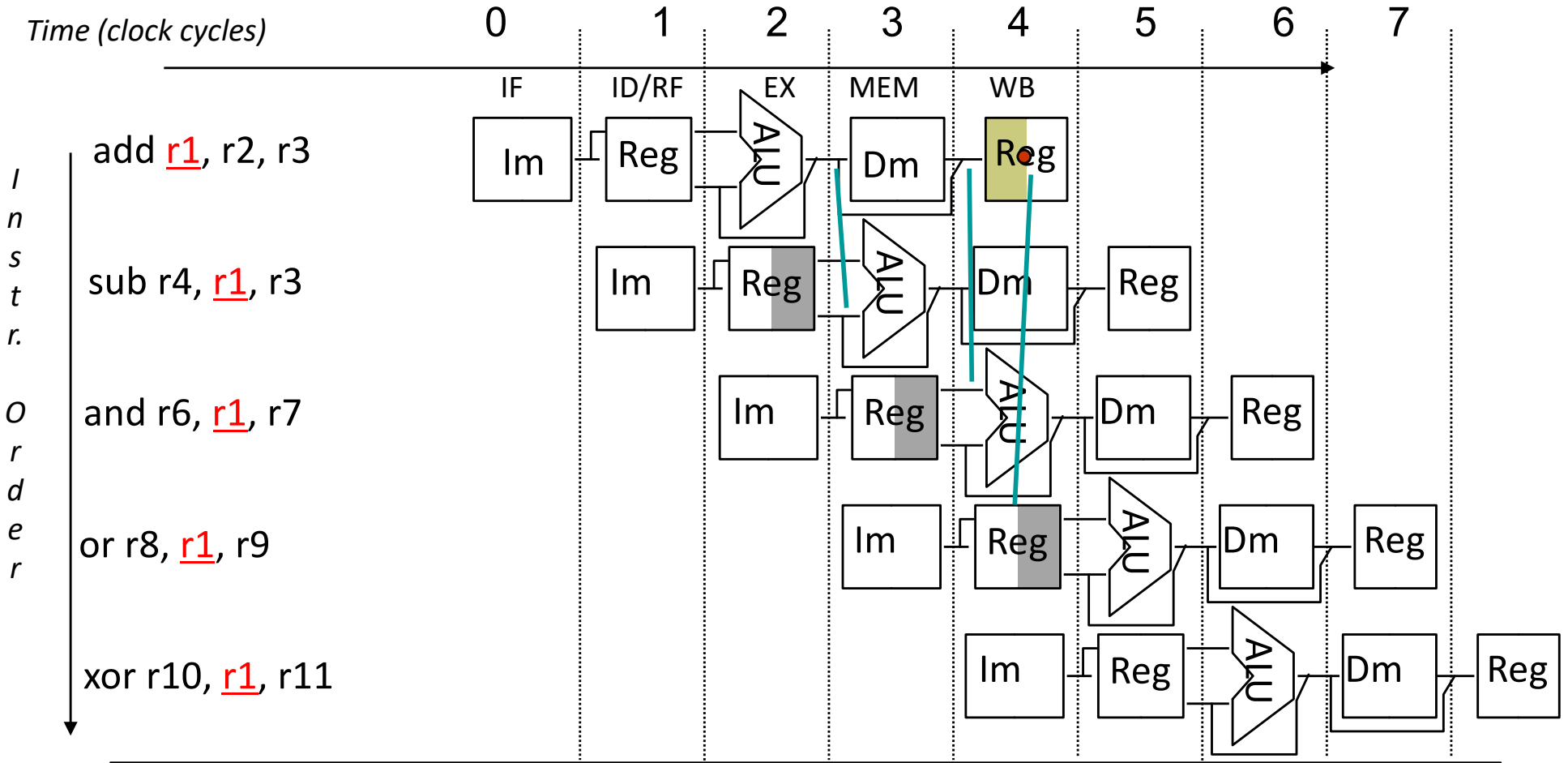
- Stalls can have a significant effect on performance
- Consider the following case
  - The ideal CPI of the machine is 1
  - A RAW hazard causes a 2 cycle stall
- If 40% of the instructions cause a stall?
  - The new effective CPI is $1 + 2 \times 0.4 = 1.8$
  - And the real % is probably higher than 40%
- You get a little more than ½ the desired performance!

Reducing Stalls – one step beyond

- Key is to be careful about when
  - Data is actually available as output
  - Data is actually required as an input
- In our example:
  - Data becomes available when add finishes EX stage
    - Cycle 2
  - Data needed by sub at the beginning of its EX stage
    - Cycle 3 (the soonest possible)
  - If you can use this value, the stall for ALU is zero!
- Fastest, but requires more hardware – called forwarding
  - Aka bypassing, short-circuiting

Decreasing Stalls: Forwarding

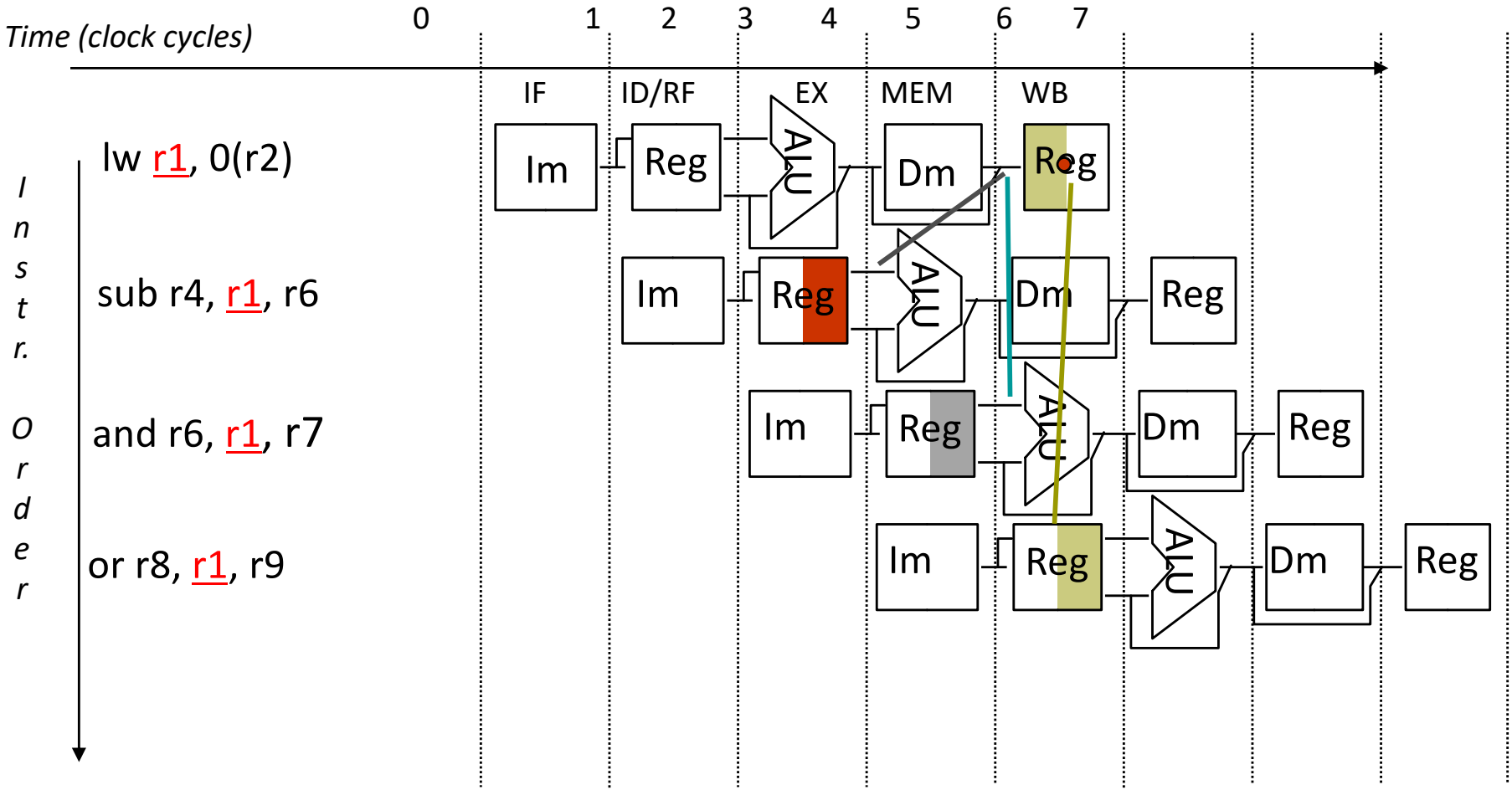
- "Forward" the data to the appropriate unit



Eliminates stalls for dependencies between ALU instrs.

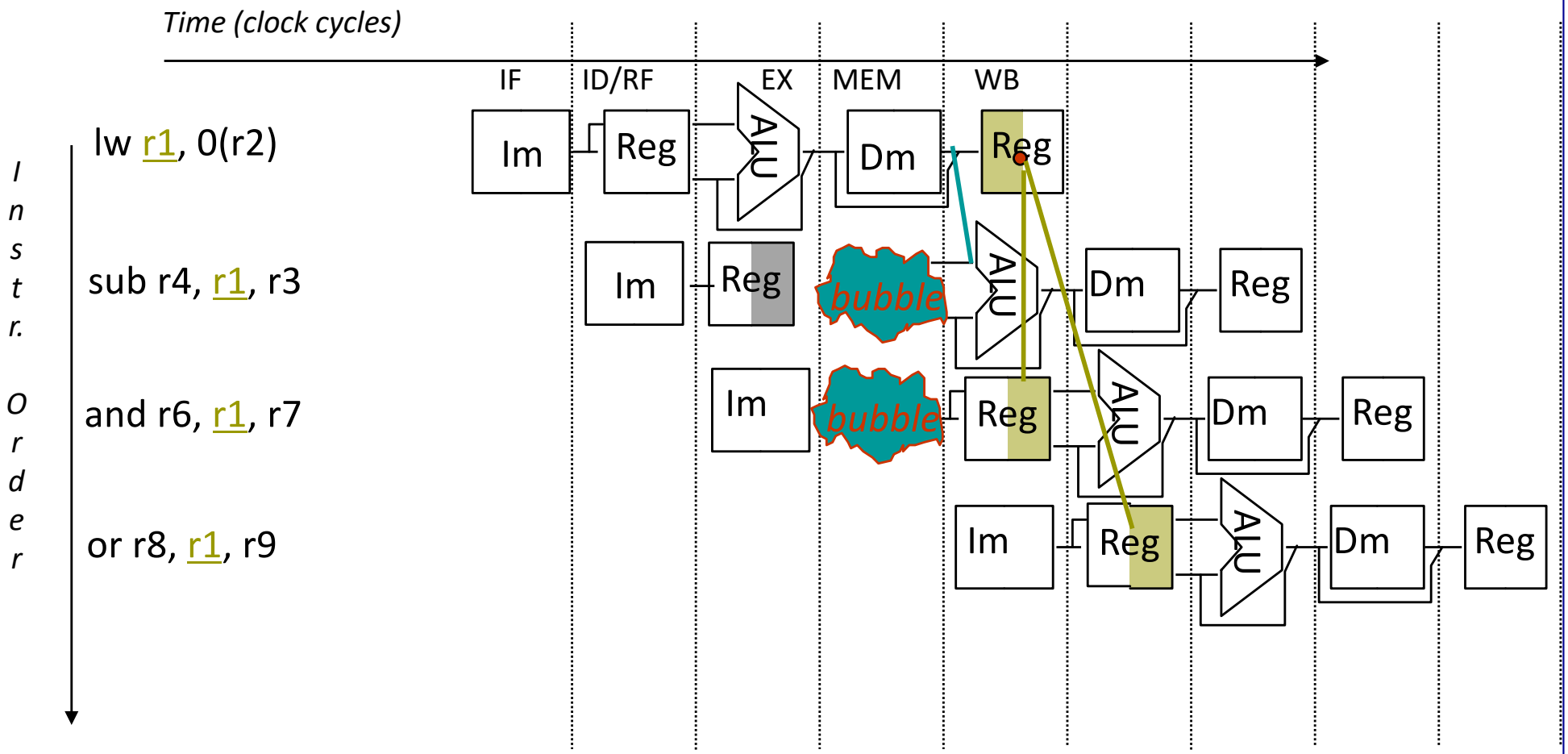
Forwarding Limitation: Load-Use Case

- Data is not available yet to be forwarded



Load-Use Case: Hardware Stall

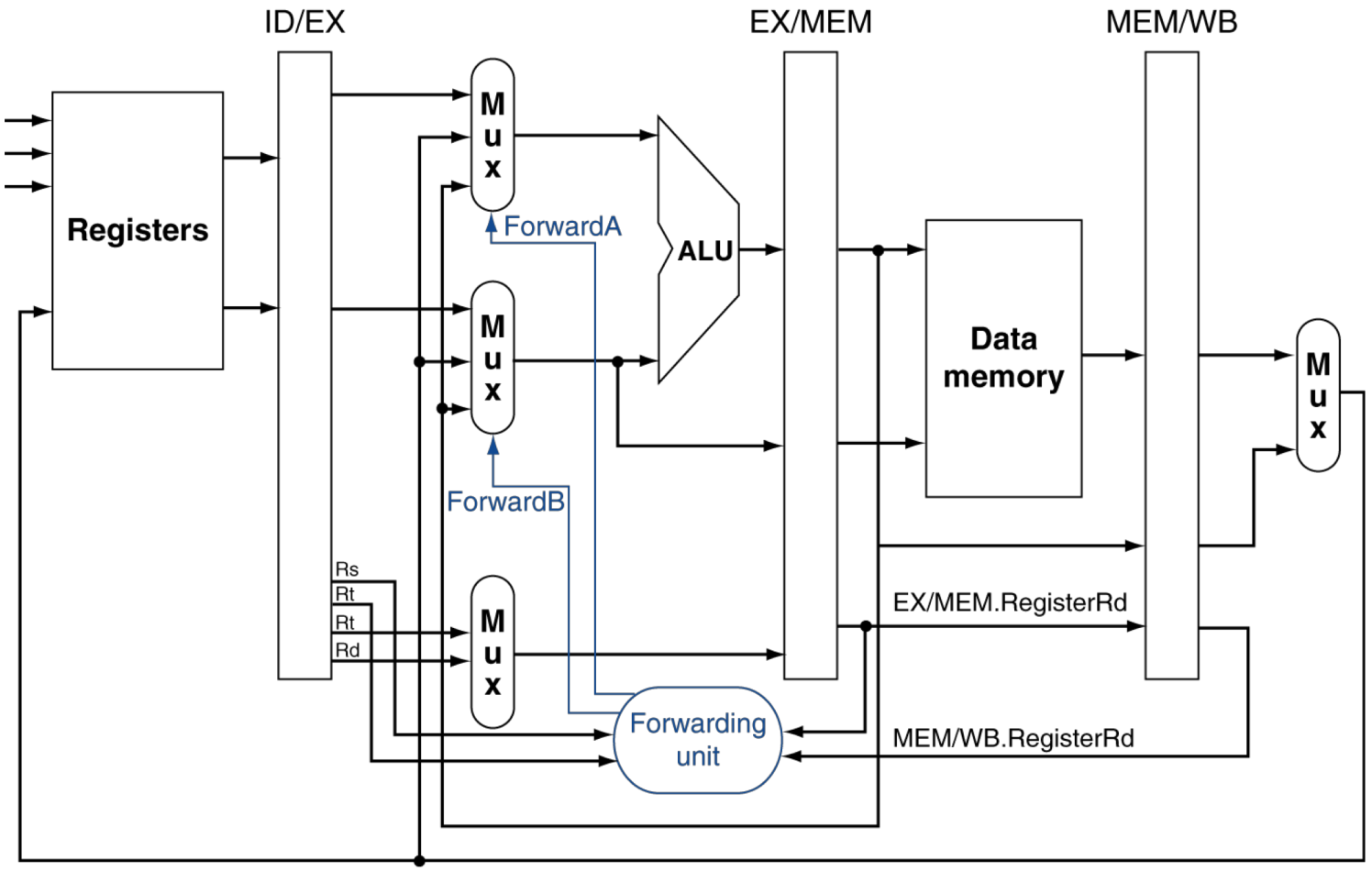
- A *pipeline interlock* checks and stops the *instruction issue*



Identifying the Forwarding Datapaths

- Identify all stages that produce new values
  - EX and MEM
- All stages after first producer are sources of forwarding data
  - MEM, WB
- Identify all stages that really consume values
  - EX and MEM
- These stages are the destinations of a forwarding data
- Add multiplexor for each pair of source/destination stages
  - Consider both possible instruction operands

Forwarding Paths: Partial



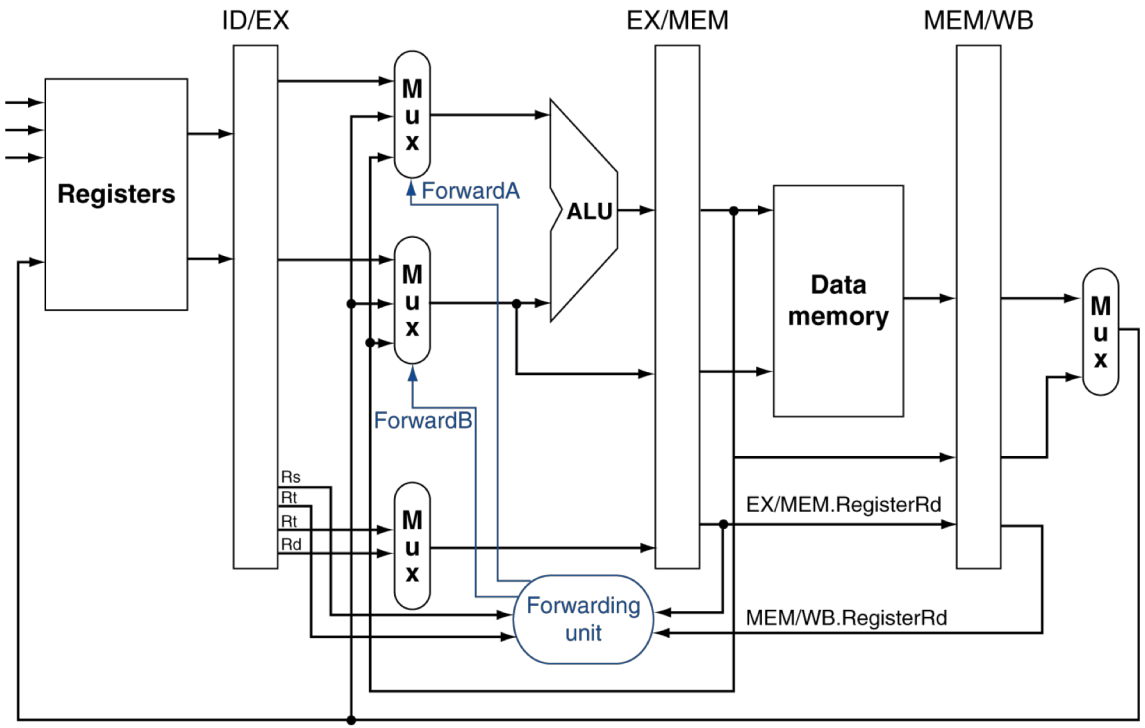
b. With forwarding

Forwarding Paths: Example

```
Example 1:
add $1,$1,$2
or  $1,$1,$4

Example 2:
add $1,$1,$2
sub $4,$5,$3
or  $1,$1,$4

Example 3:
Ld  $1,0($2)
sub $4,$5,$3
or  $1,$1,$4
```



b. With forwarding

Forwarding Control

- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Data hazards possible when
  - 1a. EX/MEM.RegisterRd == ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd == ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd == ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd == ID/EX.RegisterRt

Fwd from EX/MEM pipeline reg

Fwd from MEM/WB pipeline reg

Forwarding Control

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And if Rd for that instruction is not $zero
  - EX/MEM.RegisterRd ≠ 0,
    MEM/WB.RegisterRd ≠ 0
- And if forwarding instruction is not a load in MEM stage
  - EX/MEM.MemToReg==0
  - This is a case we have to stall…

Forwarding Control (Stall Case not Shown)

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRt))
    ForwardB = 01

Double Data Hazard

- Consider the sequence:

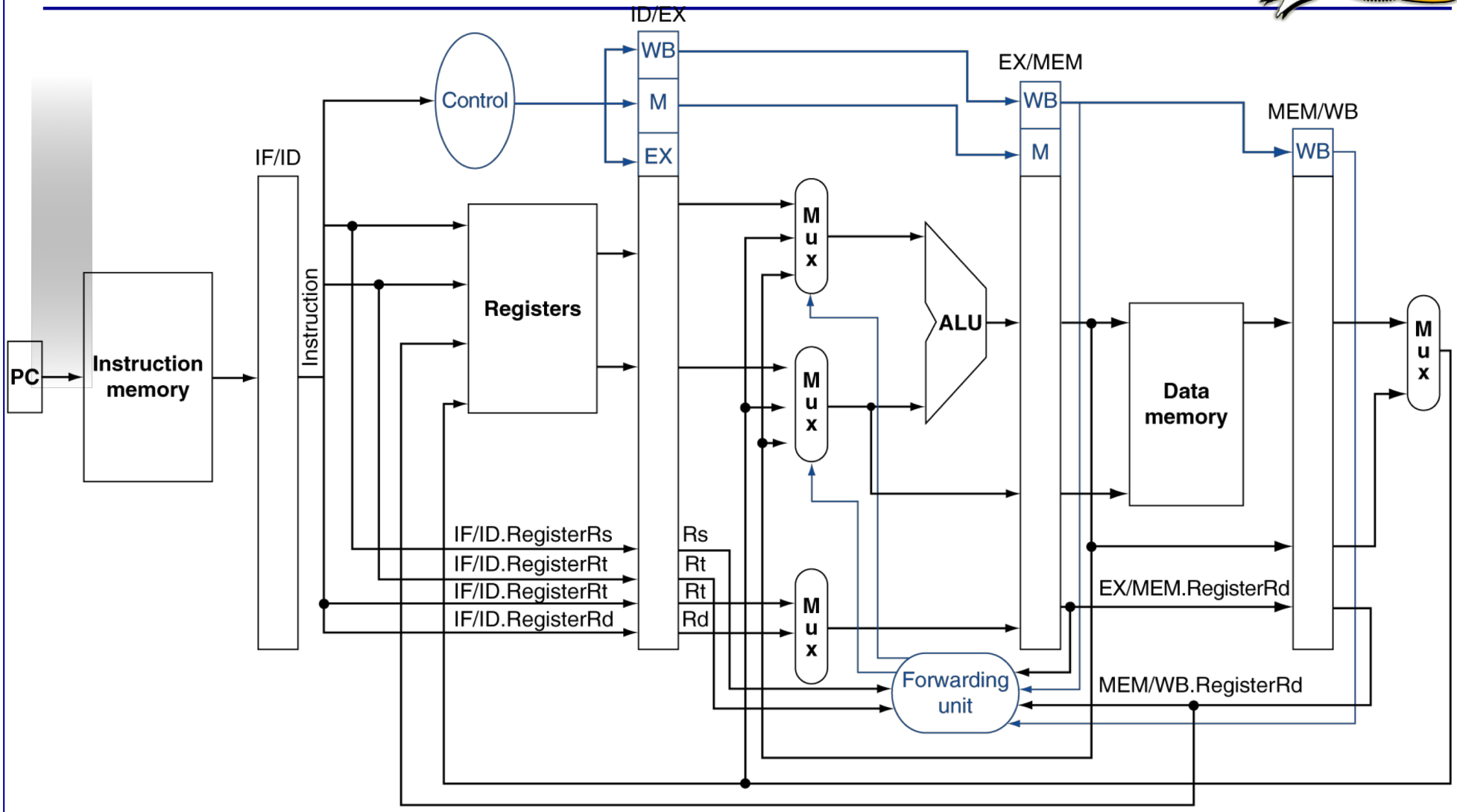
```
add $1,$1,$2
sub $1,$1,$3
or  $1,$1,$4
```

- Both hazards occur
  - Want to use the most recent result from the sub
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

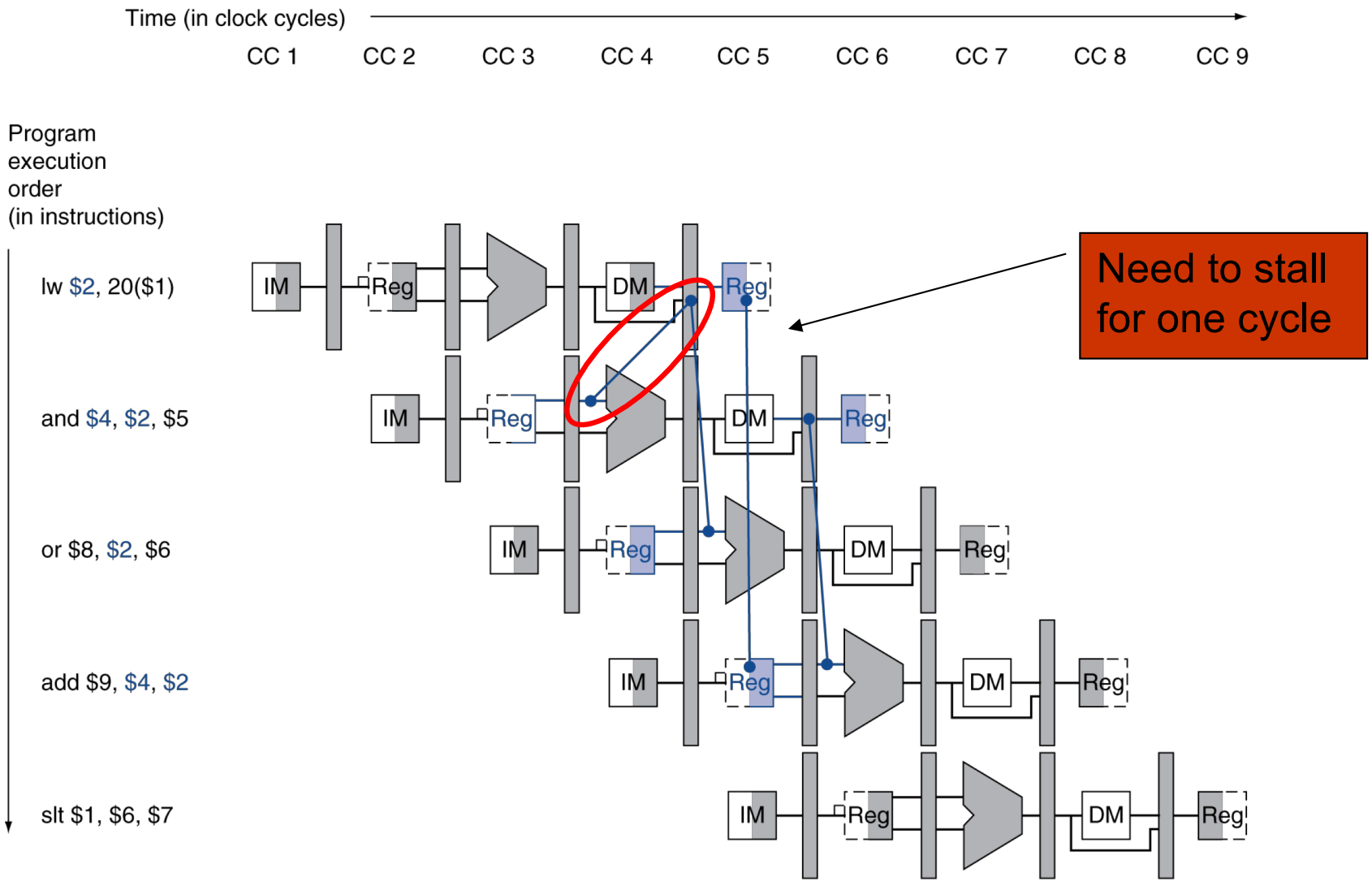
Forwarding Control (Revised)

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Datapath with Forwarding



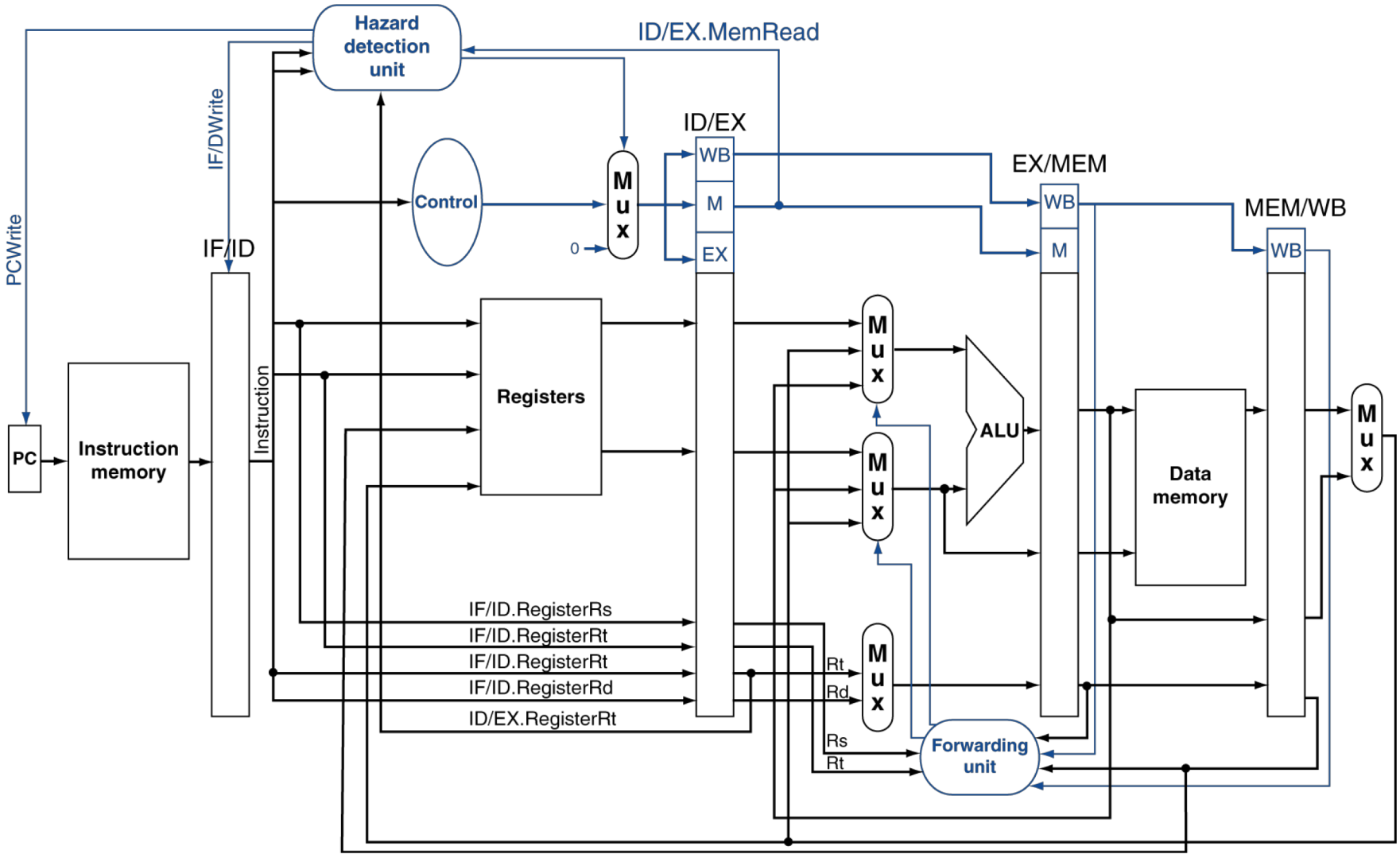
Load-Use Data Hazard



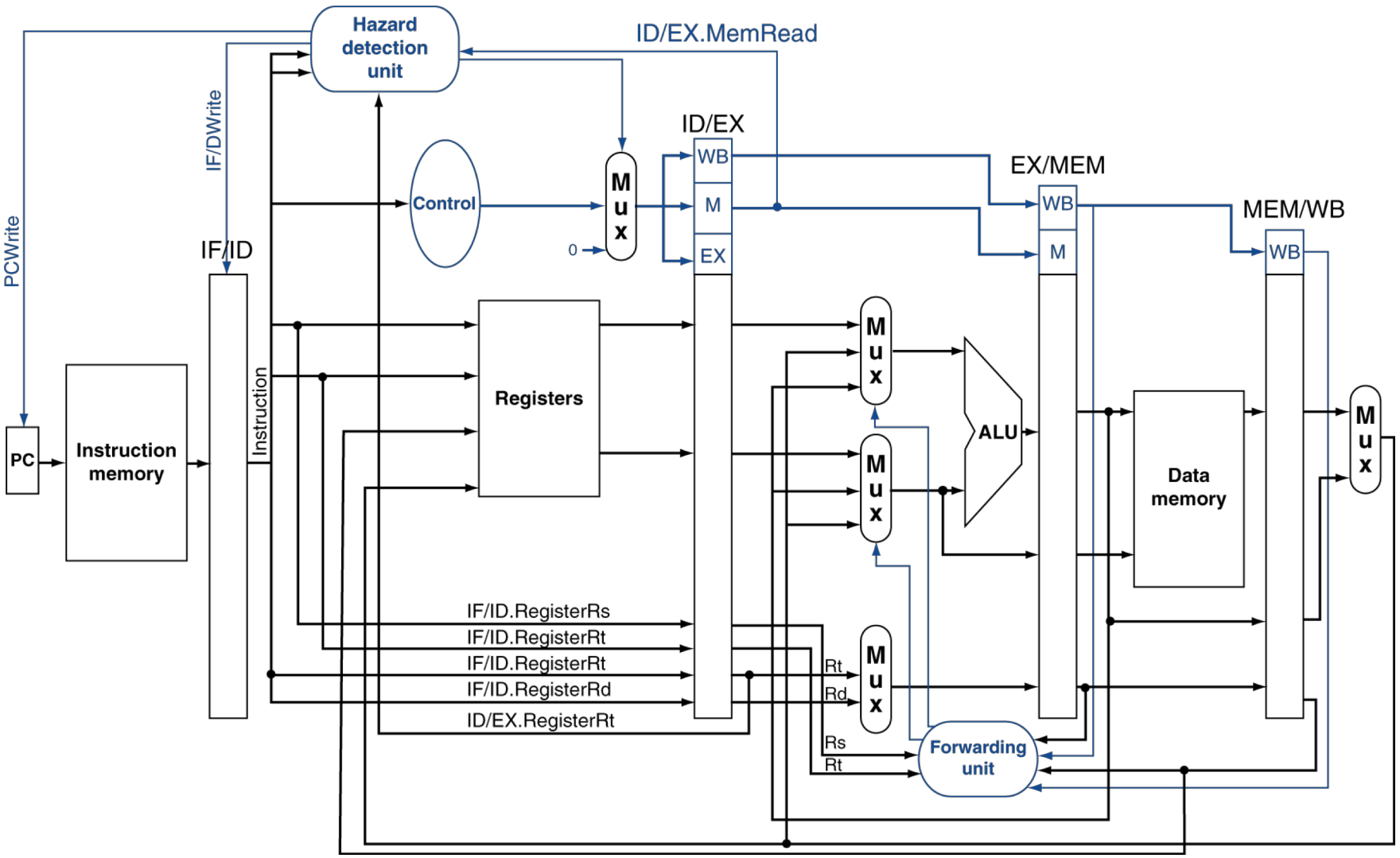
Load-Use Hazard Detection

- Check when use instruction is decoded in ID stage
- ALU register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

Datapath with Hazard Detection

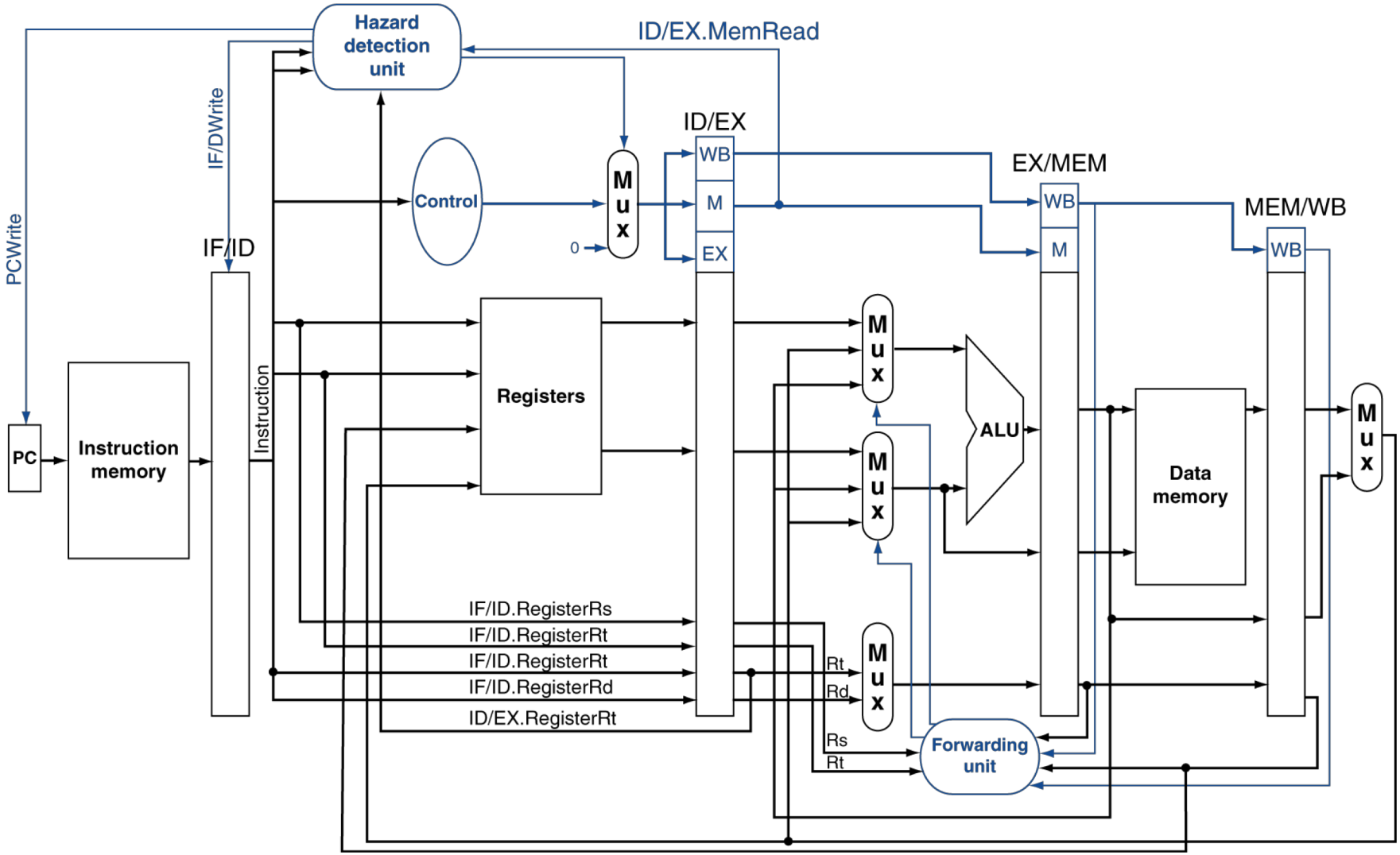


Example: Load-Use Stall



Example: Load-Use Stall 1 cycle later

```
sub r4, r1, r3        nop     lw r1, 0(r2)
```



Looking Ahead

- Compilers and data hazards
- Control hazards
- Exceptions and interrupts
- Advanced pipelining – (CPI < 1.0)