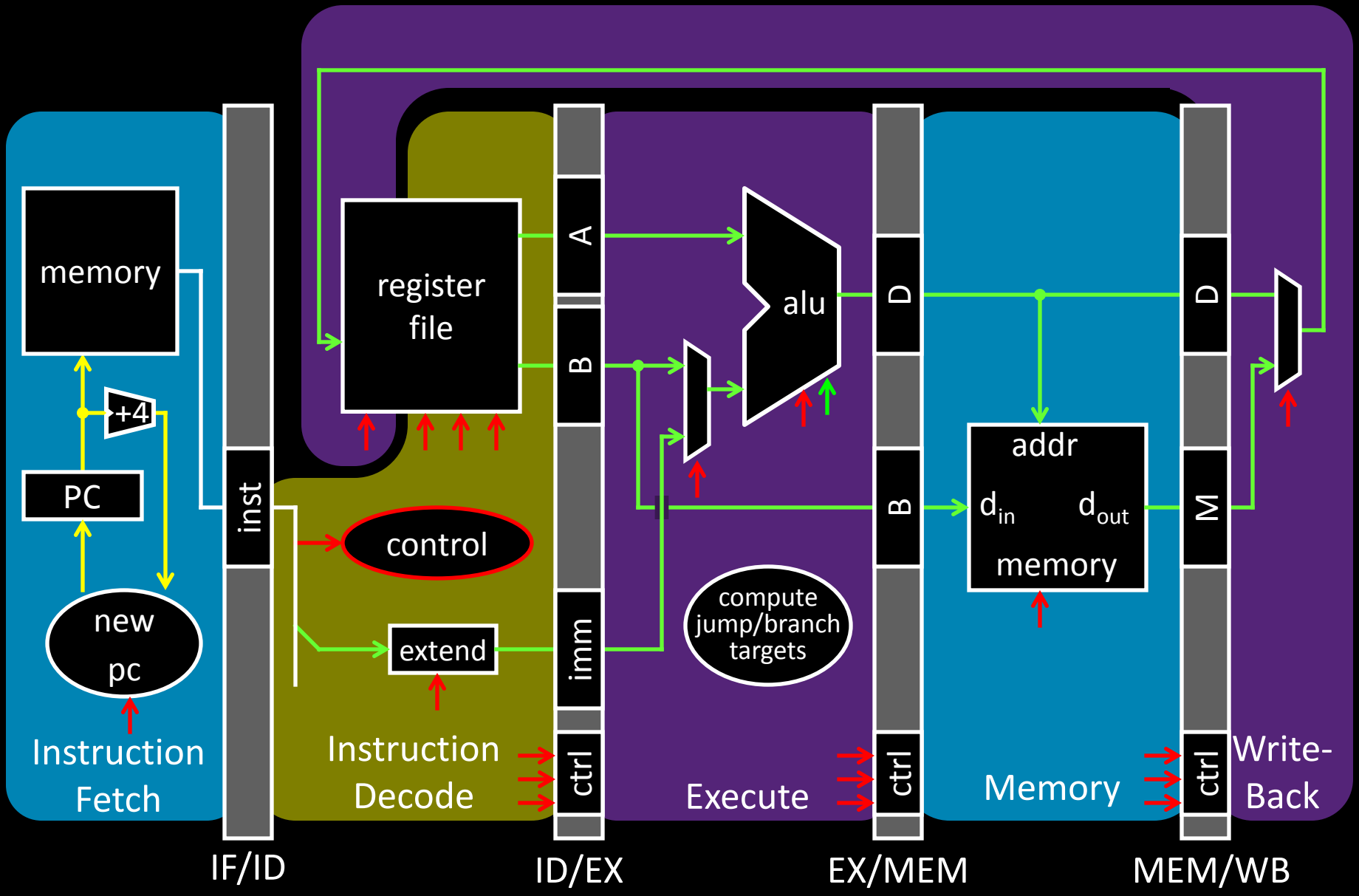
# Pipeline Hazards

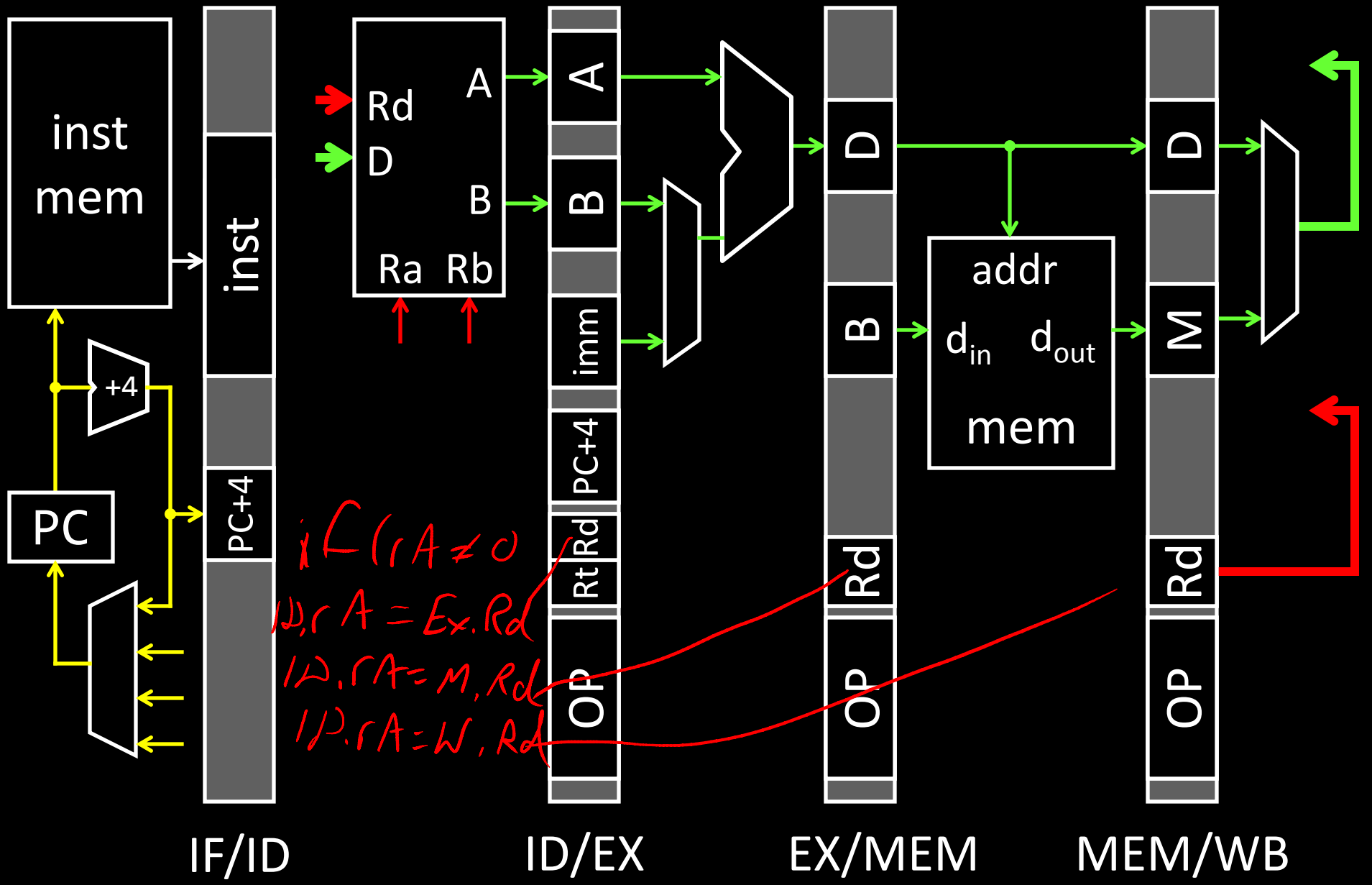
**Hakim Weatherspoon**
**CS 3410, Spring 2012**
Computer Science
Cornell University

See P&H Appendix 4.7

# Pipelined Processor



# Pipelined Processor



# Goals for Today

Data Hazards

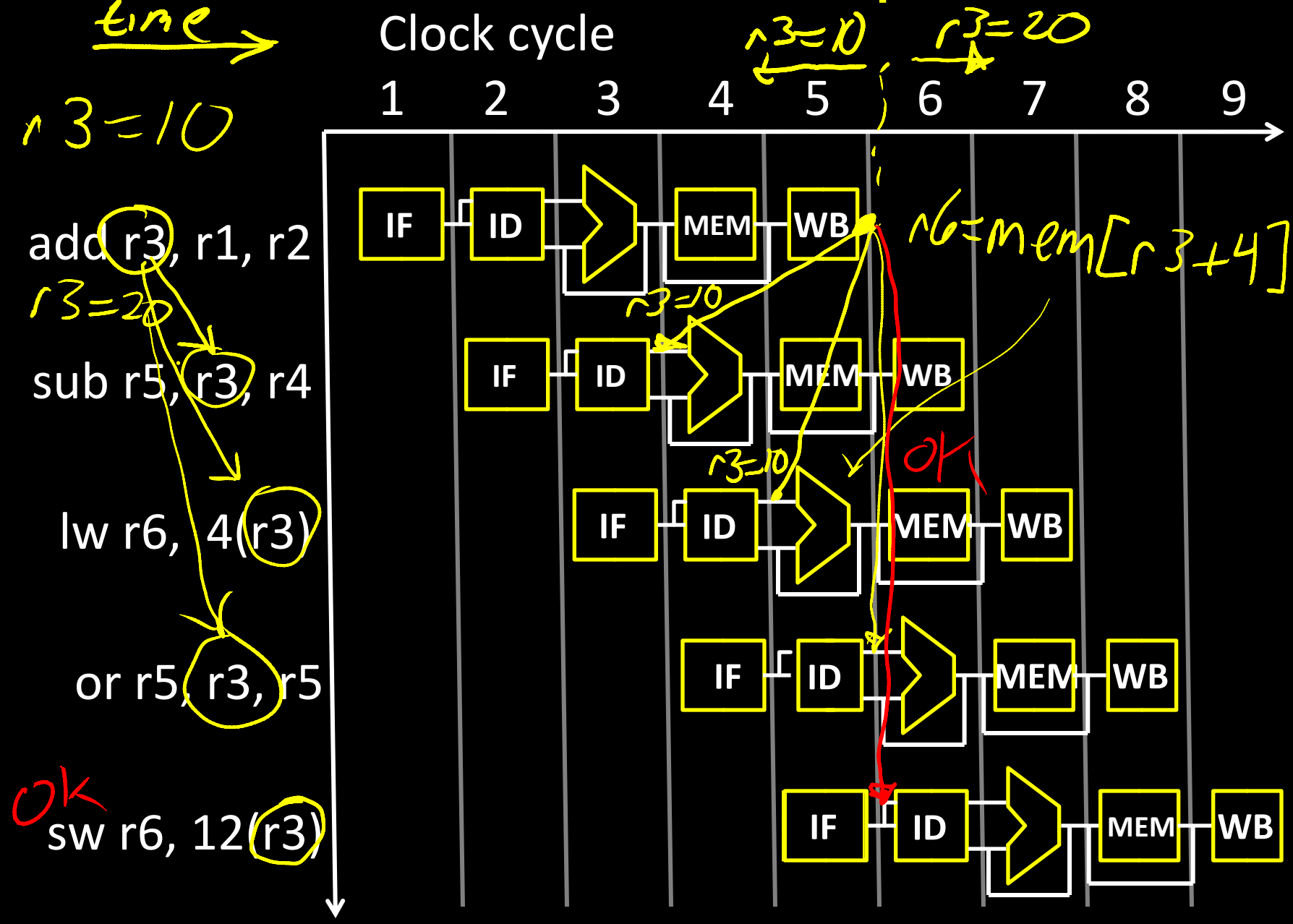
- Data dependencies
- Problem, detection, and solutions
  - (delaying, stalling, forwarding, bypass, etc)
- Forwarding unit
- Hazard detection unit

Next time

- Control Hazards

  What is the next instruction to execute if a branch is taken? Not taken?

# Broken Example



# What Can Go Wrong?

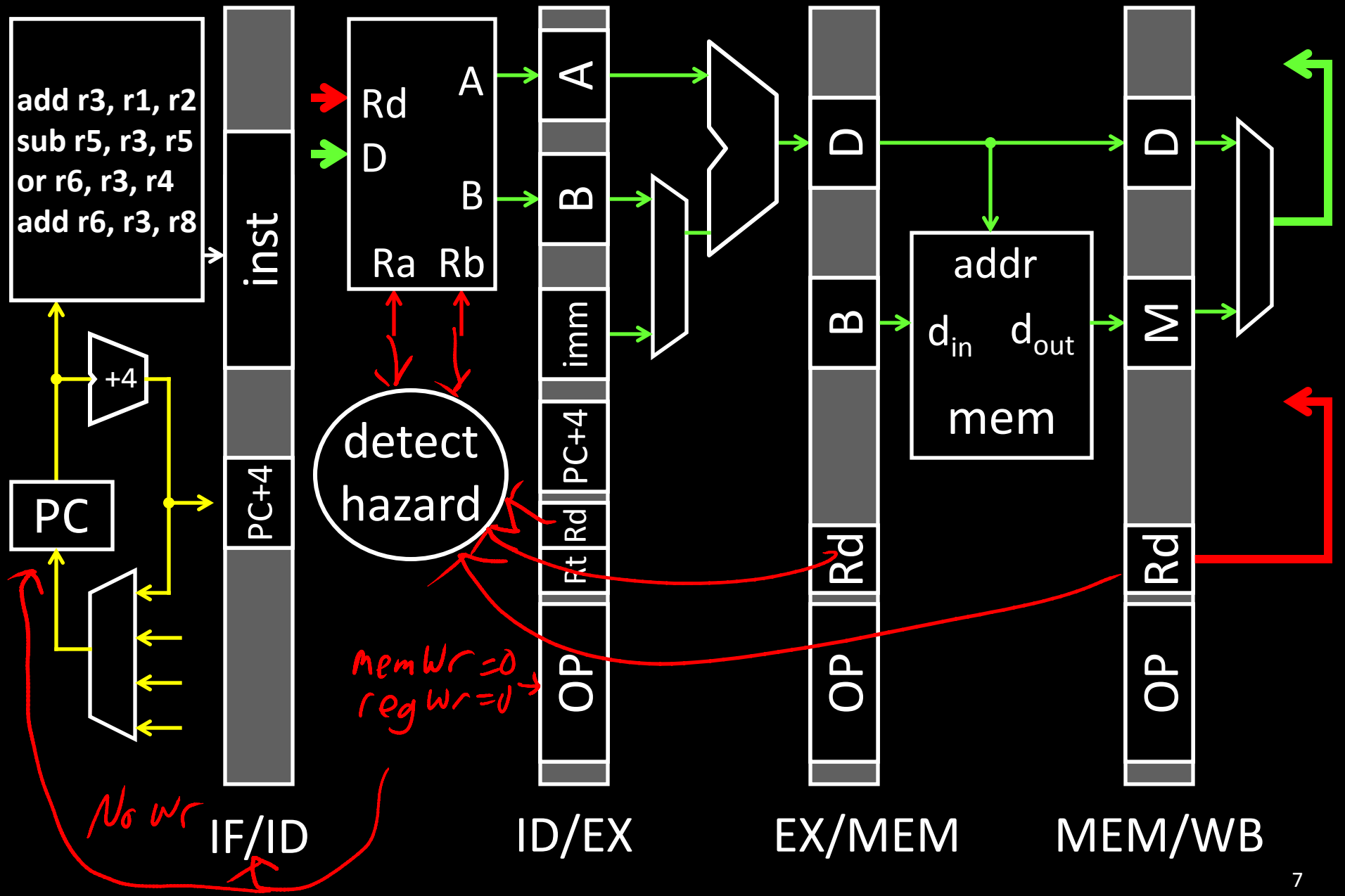
## Data Hazards

- register file reads occur in stage 2 (ID)
- register file writes occur in stage 5 (WB)
- next instructions may read values about to be written

## How to detect? Logic in ID stage:

```
stall = (IF/ID.rA != 0 && (IF/ID.rA == ID/EX.rD ||
                           IF/ID.rA == EX/M.rD ||
                           IF/ID.rA == M/WB.rD))
        || (same for rB)
```

# Detecting Data Hazards



# Resolving Data Hazards

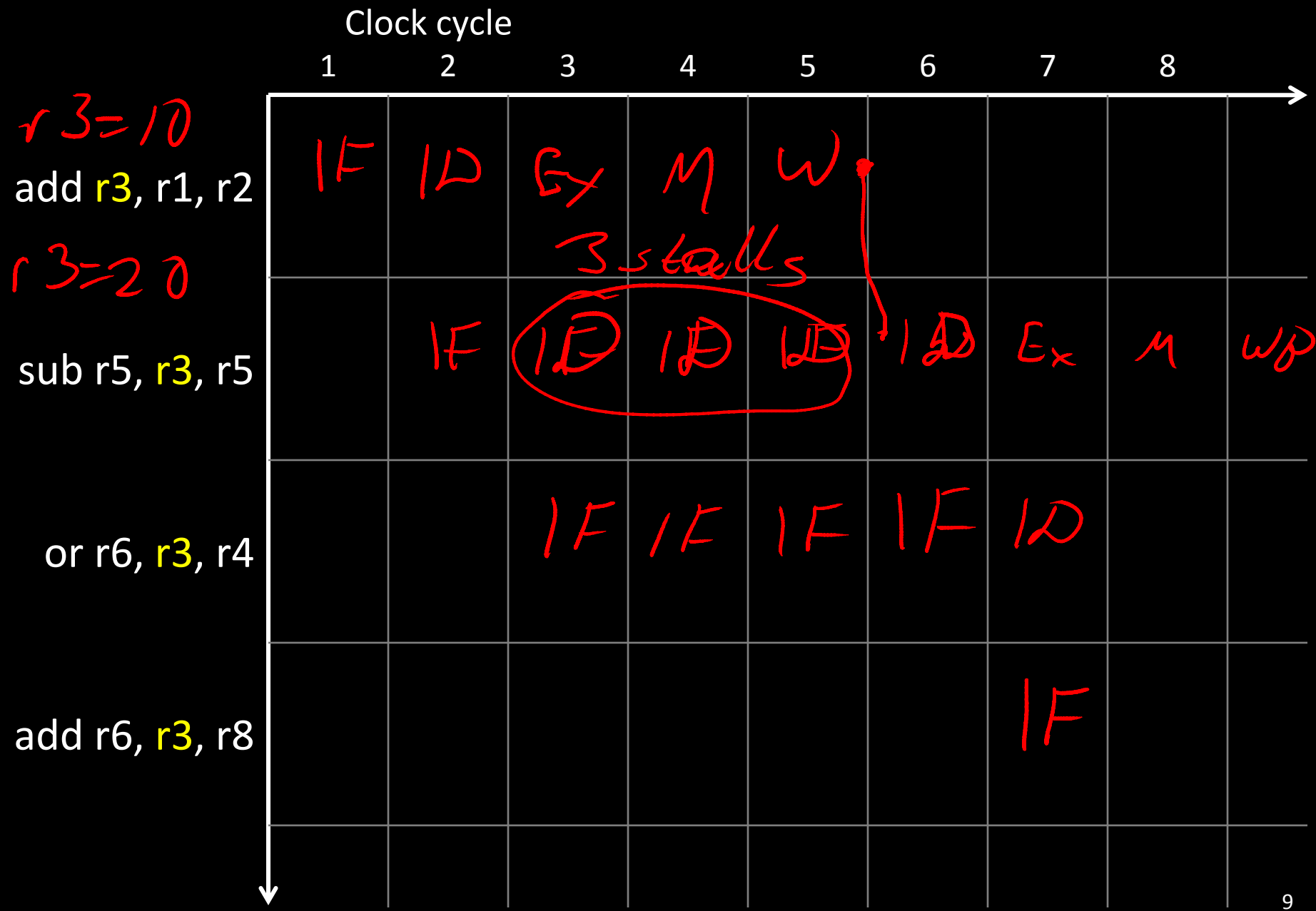
What to do if data hazard detected?

1) Wait/stall

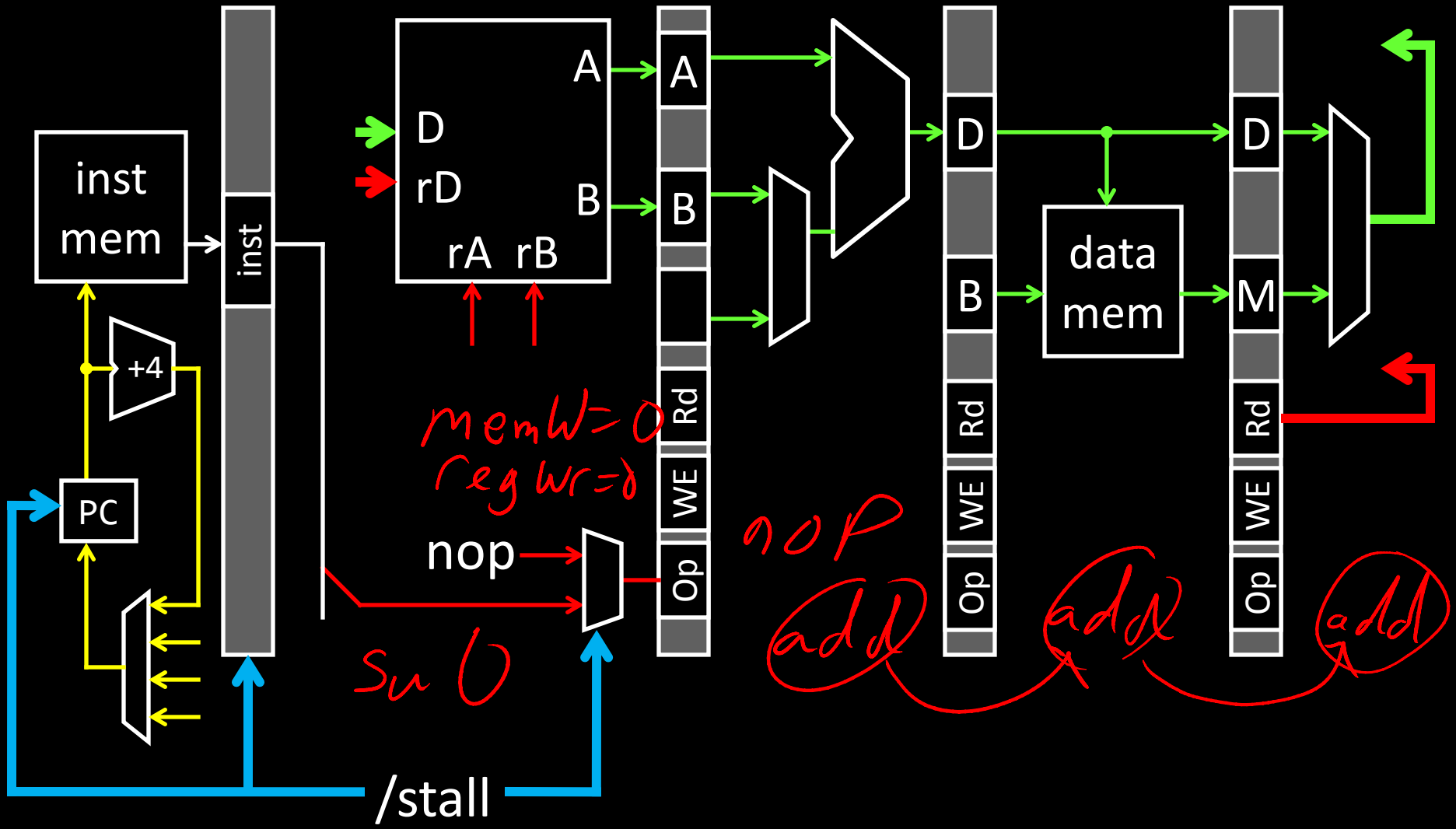
2) in sw reorder

3) Forward/ Bypass

# Stalling


Clock cycle
1
2
3
4
5
6
7
8
r3=10
add r3, r1, r2
r3=20
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8
IF ID Ex M W
3 stalls
IF ID ID ID ID Ex M WB
IF IF IF IF ID
IF

# Stalling

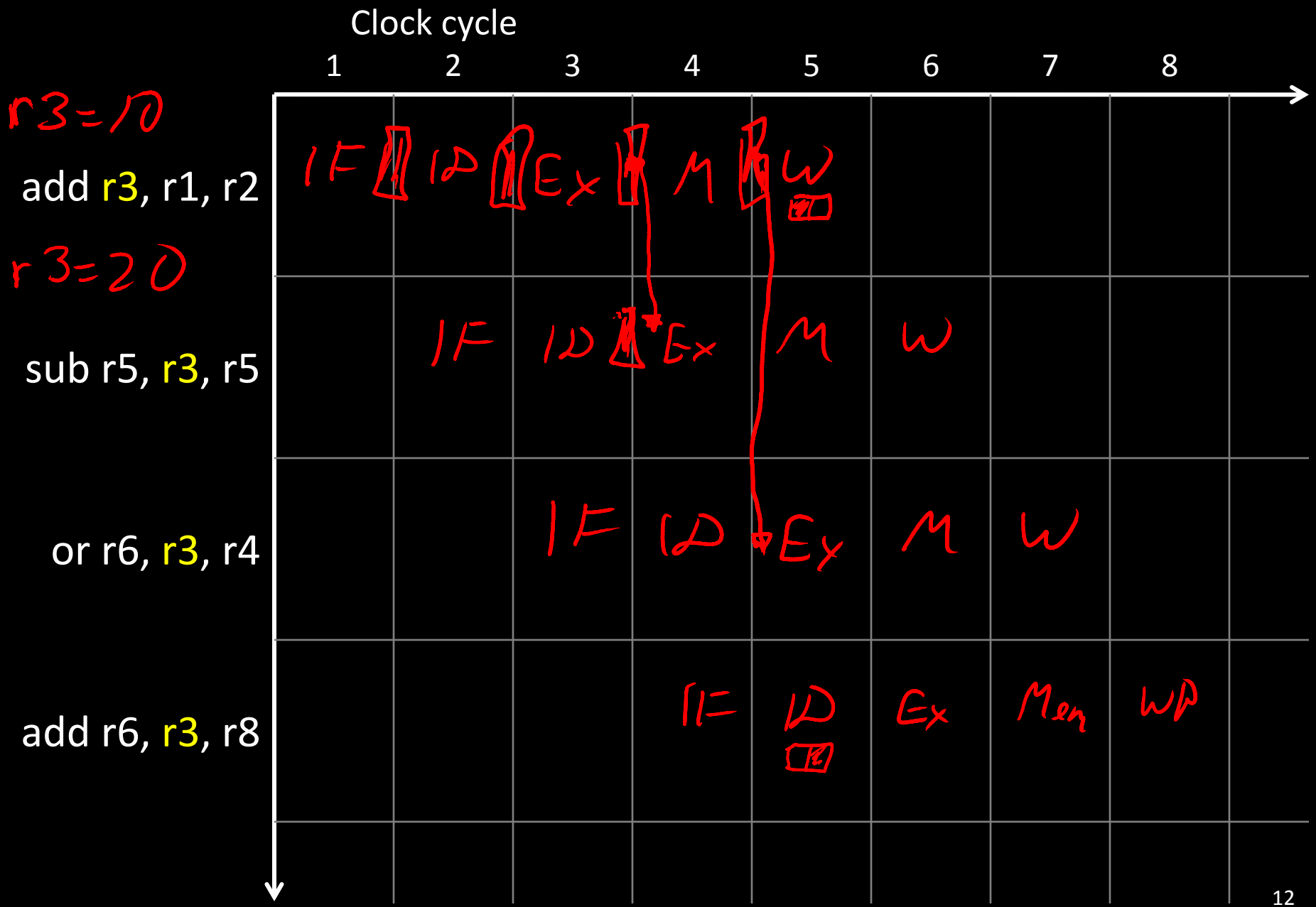

inst mem
inst
+4
PC
A
D
rD
B
rA rB
A
B
Op WE Rd
nop
memW=0
regWr=0
sub
/stall
D
B
Op WE Rd
data mem
D
M
Op WE Rd
nop
add
add
add

# Stalling

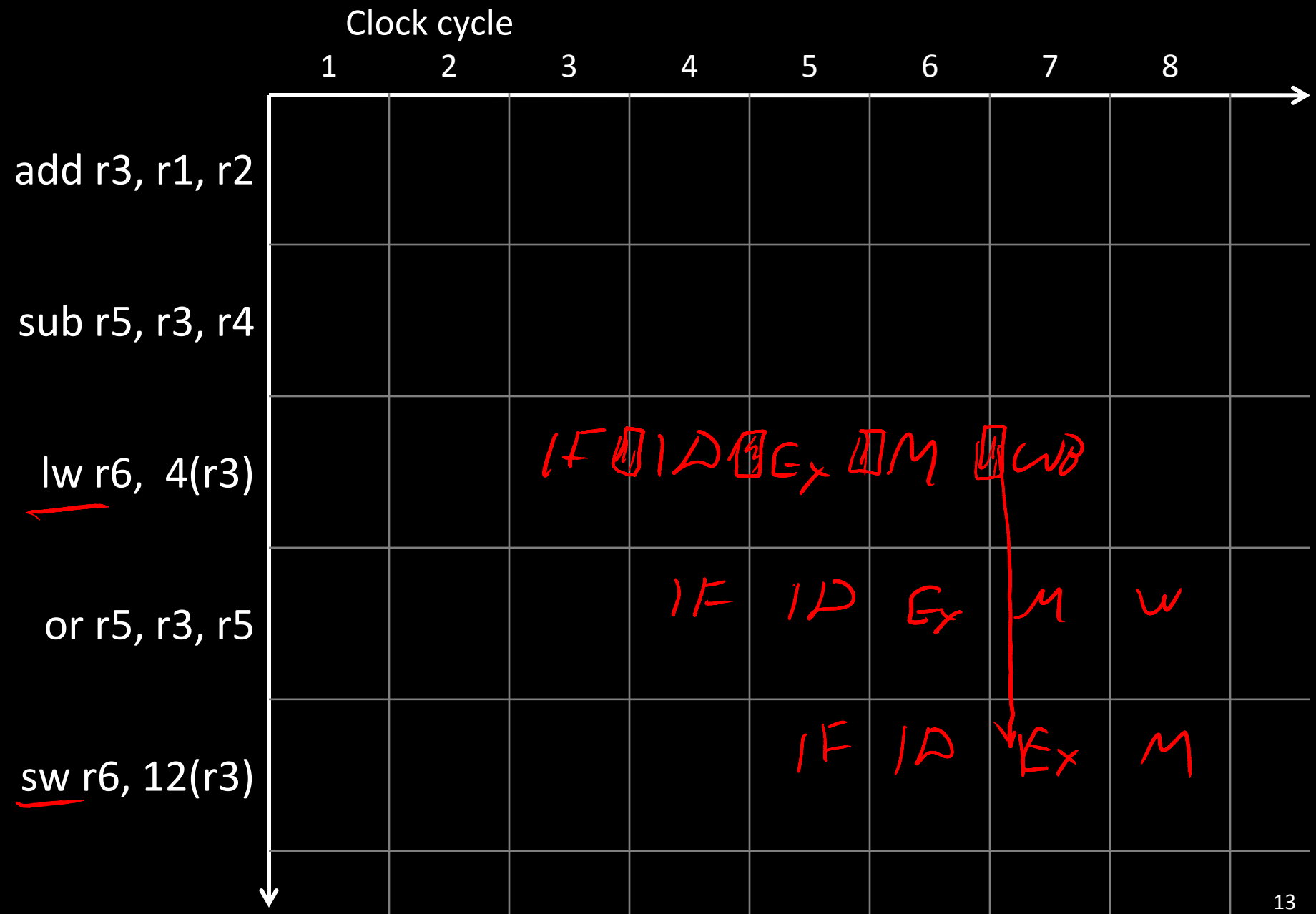
How to stall an instruction in ID stage

- prevent IF/ID pipeline register update
  - stalls the ID stage instruction
- convert ID stage instr into nop for later stages
  - innocuous “bubble” passes through pipeline
- prevent PC update
  - stalls the next (IF stage) instruction

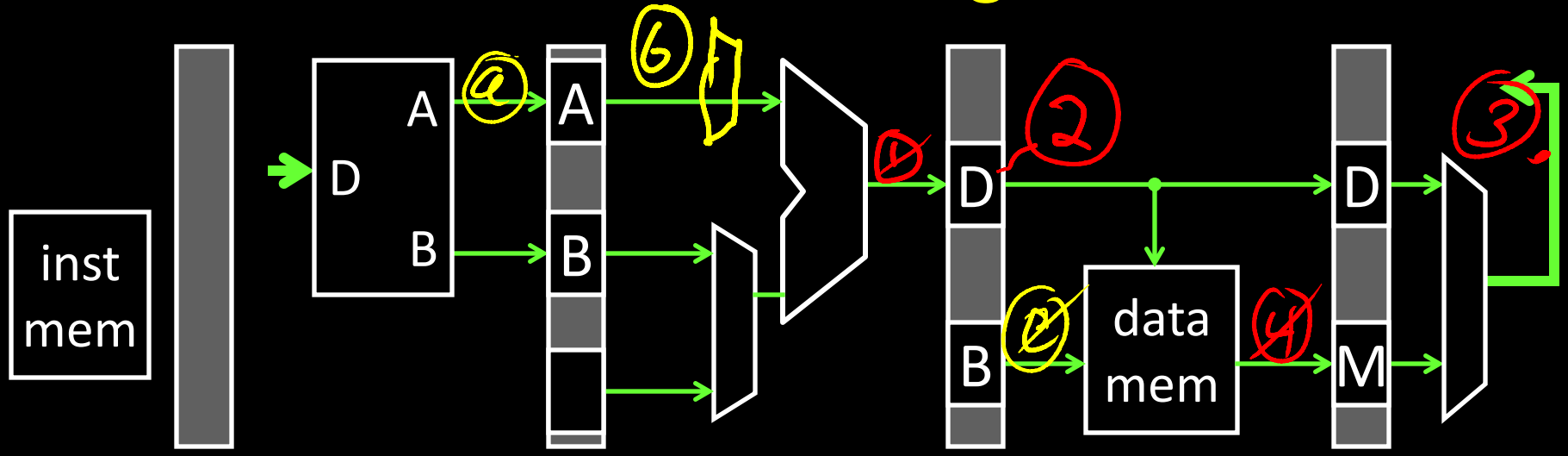
# Forwarding


Clock cycle
1
2
3
4
5
6
7
8
r3=10
add r3, r1, r2
r3=20
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8
IF
ID
Ex
M
W
IF
ID
Ex
M
W
IF
ID
Ex
M
W
IF
ID
Ex
Mem
WB

# Forwarding


Clock cycle
1
2
3
4
5
6
7
8
add r3, r1, r2
sub r5, r3, r4
lw r6, 4(r3)
or r5, r3, r5
sw r6, 12(r3)
IF ID Ex M WB
IF ID Ex M W
IF ID Ex M

# Forwarding




Forward correct value from? to?

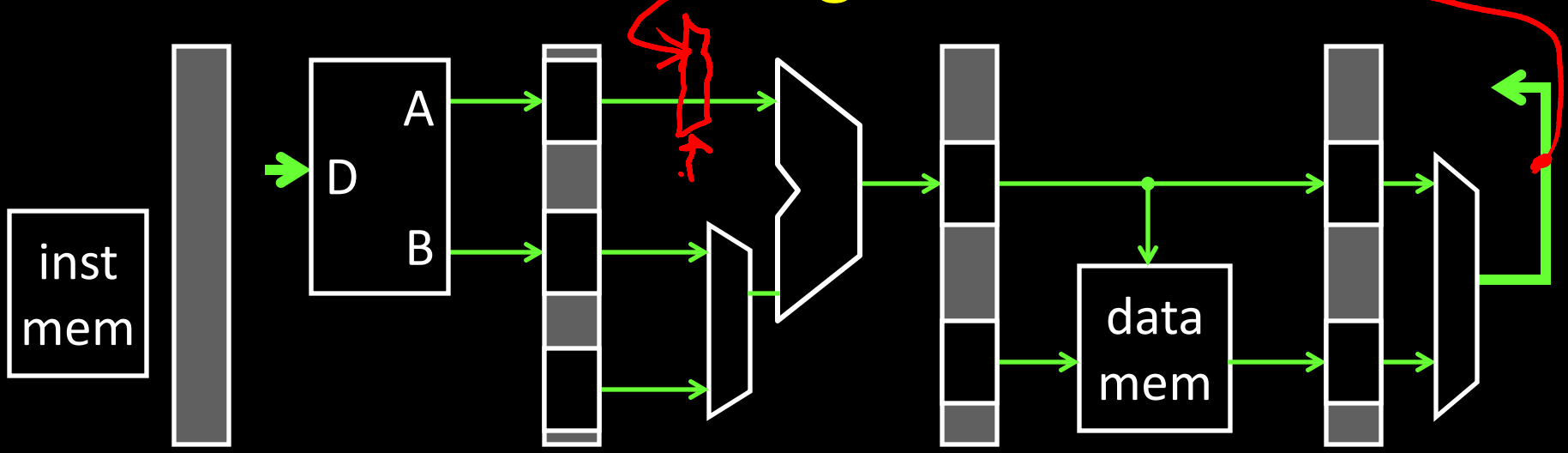
1. ALU output: too late in cycle?
2. EX/MEM.D pipeline register (output from ALU)
3. WB data value (output from ALU or memory)
4. MEM output: too late in cycle, on critical path

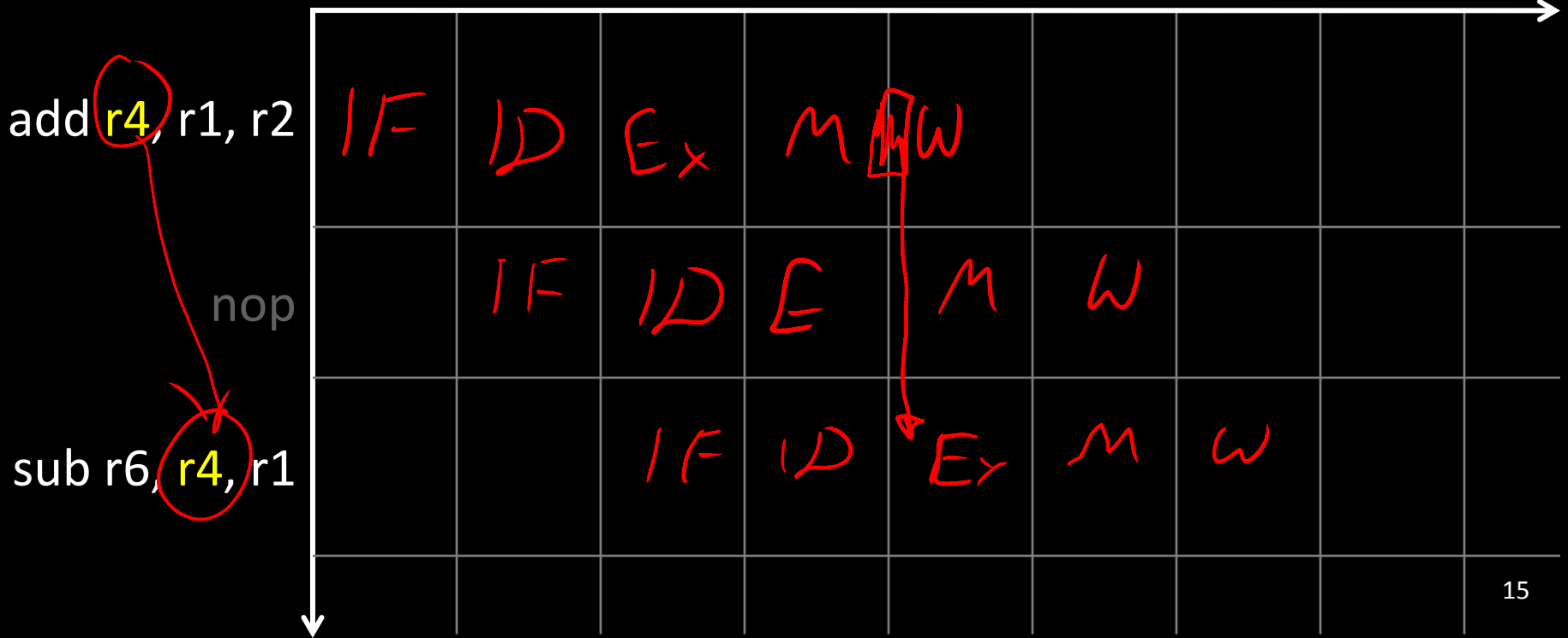
a) ID (just after register file)
   - maybe pointless?

b) EX, just after ID/EX.A and ID/EX.B are read

c) MEM, just after EX/MEM.B is read: on critical path

# Forwarding Path 1



# WB to EX Bypass

## WB to EX Bypass

Check pg 369

- EX needs value being written by WB

Resolve:

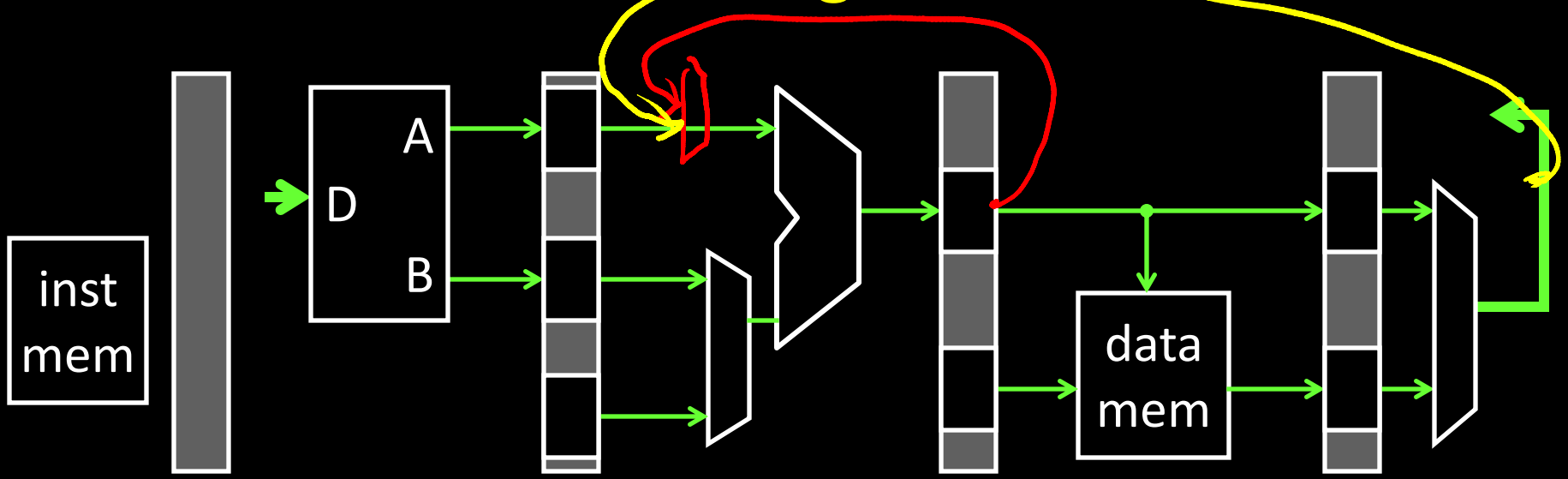
Add bypass from WB final value to start of EX

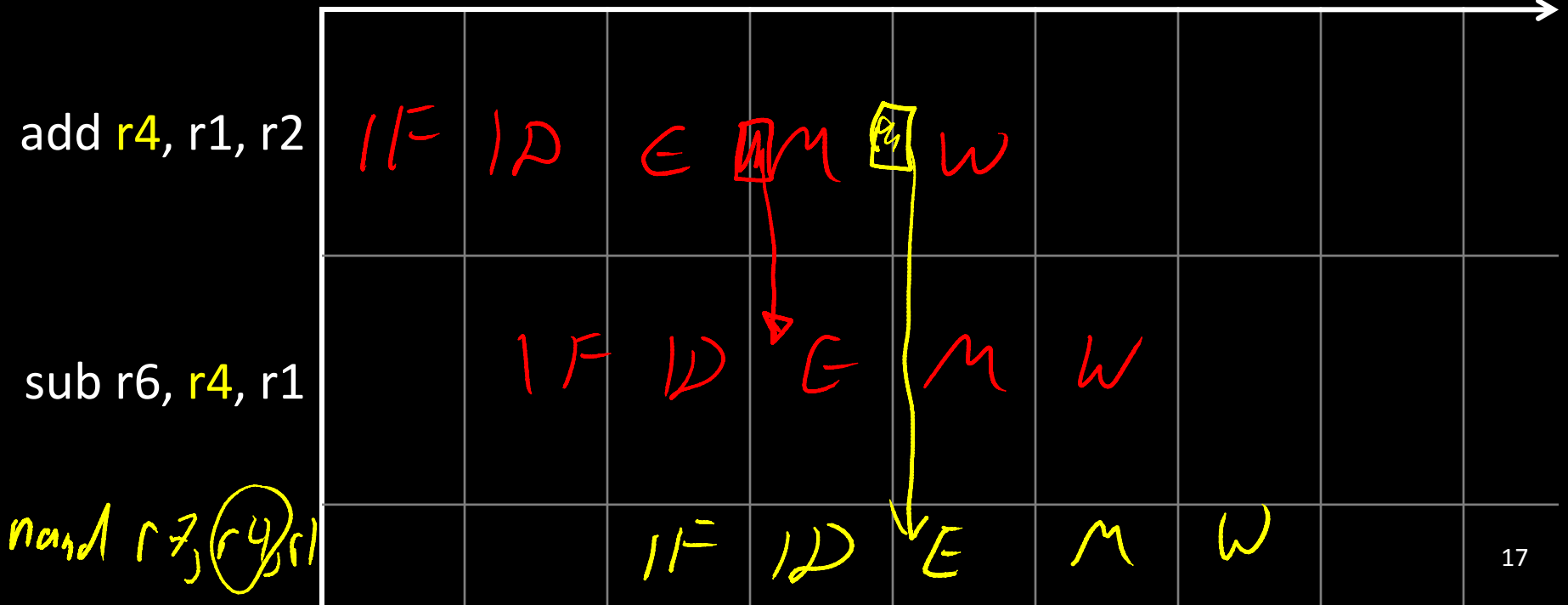
Detect:

Mem/WB.WE and mem/W.Rd≠0
and ID/EX.Ra == Mem/WB.Rd
and not ID/Ex.WE and Ex/M.Rd≠0
and ID/Ex.Ra≠ Ex/Mem.Rd

Same for B

# Forwarding Path 2



# MEM to EX Bypass

## MEM to EX Bypass

- EX needs ALU result that is still in MEM stage

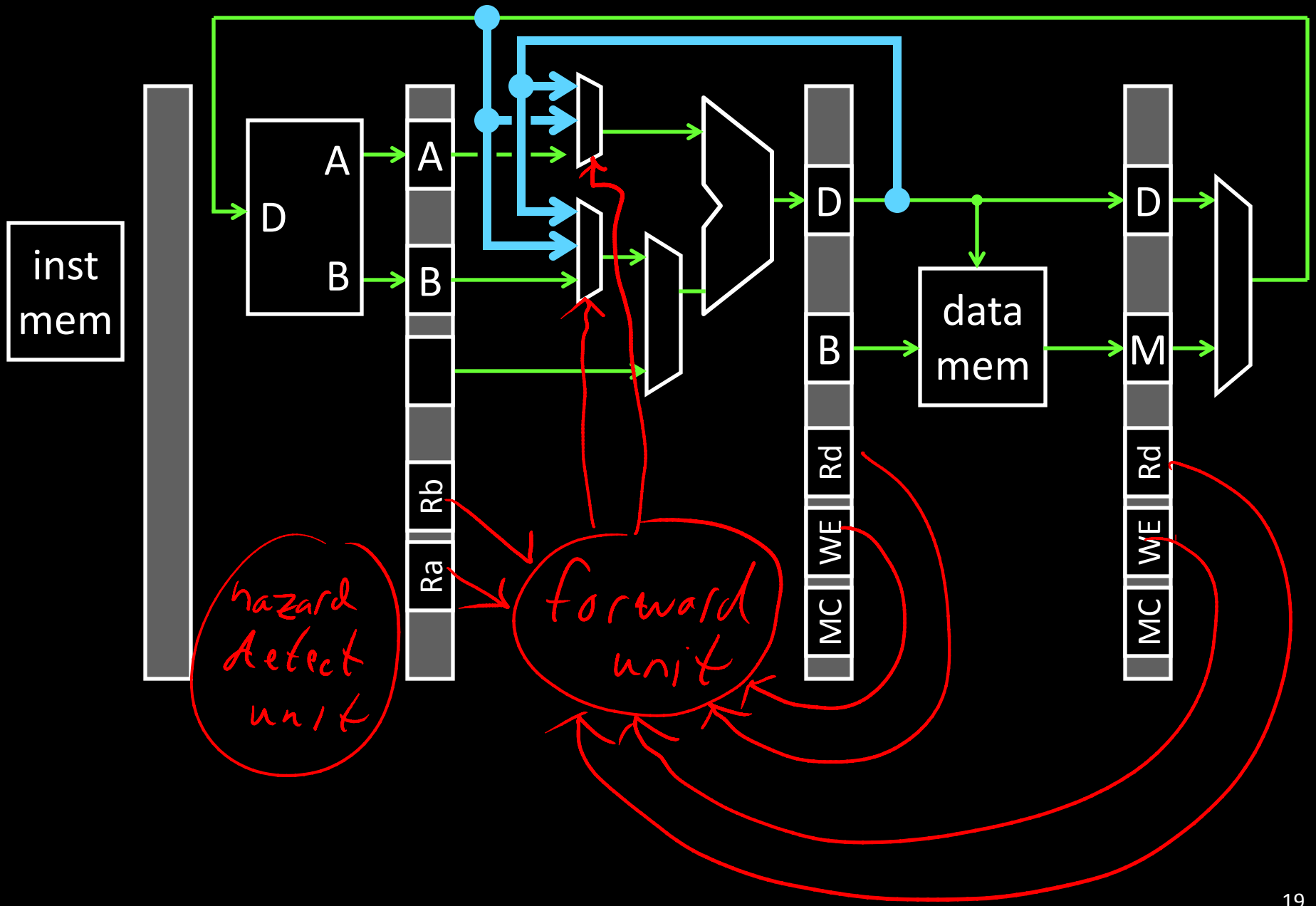
Resolve:

Add a bypass from EX/MEM.D to start of EX

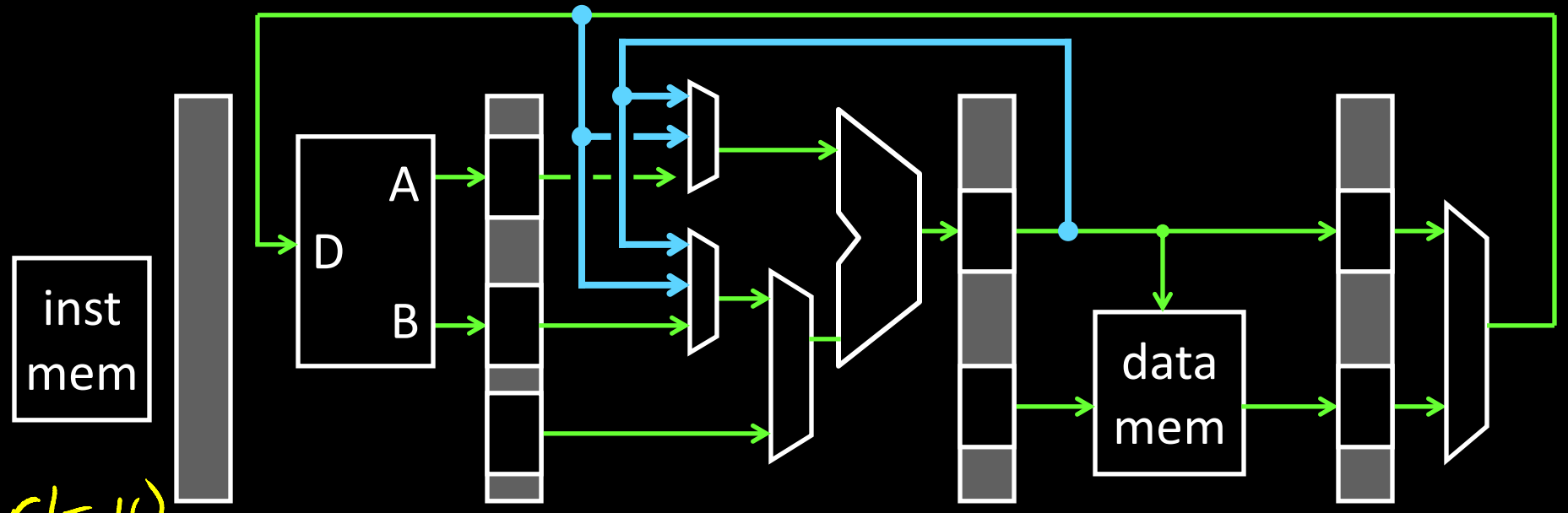
Detect:

Ex/Mem.WE and Ex/Mem.Rd≠0
and D/Ex.Ra == Ex/Mem.Rd

# Forwarding Datapath



# Tricky Example




r1=10

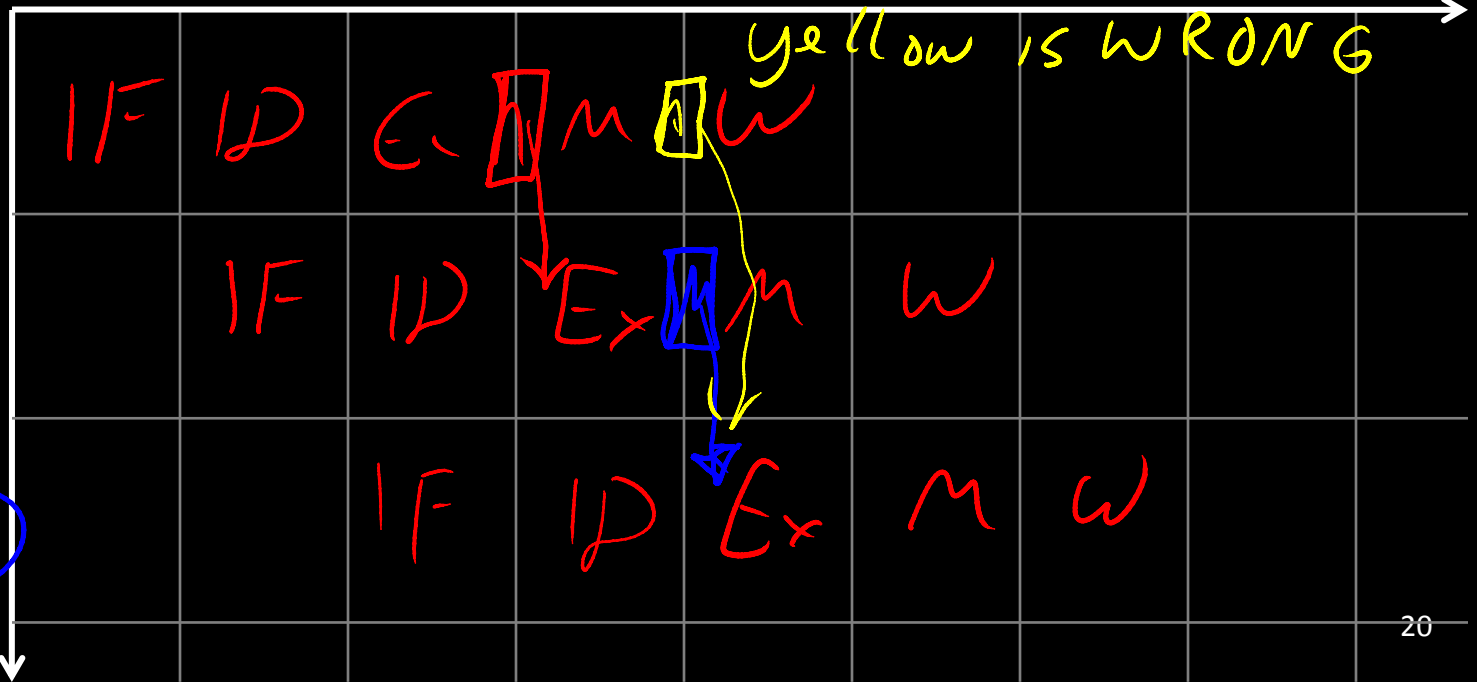
add r1, r1, r2

r1=20

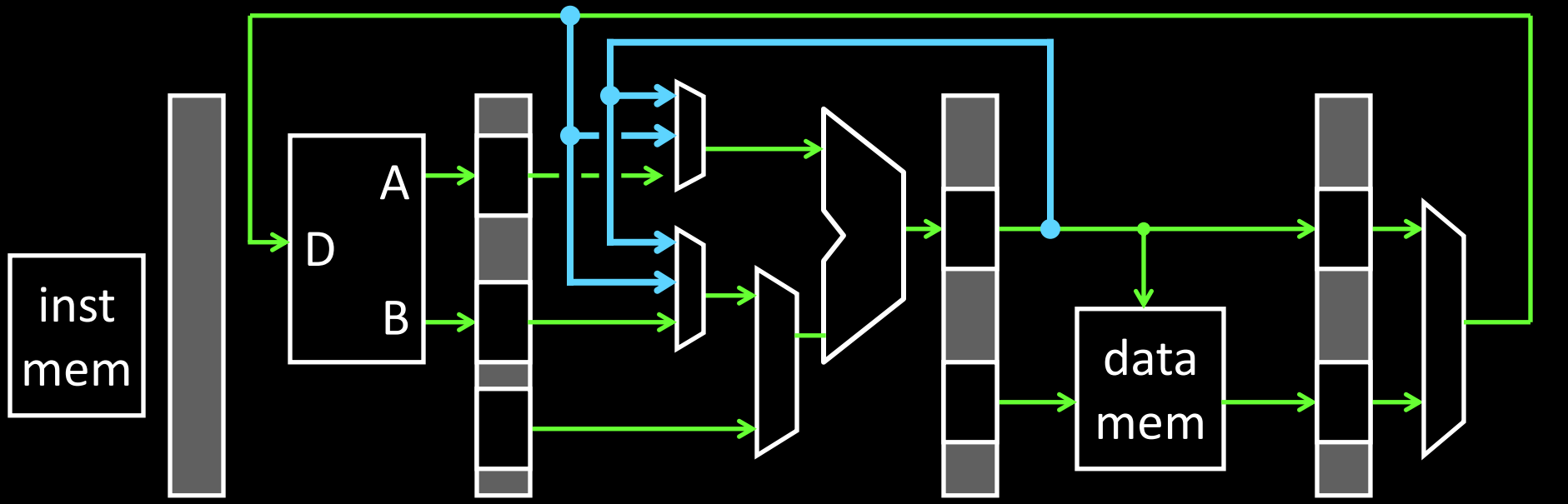
SUB r1, r1, r3

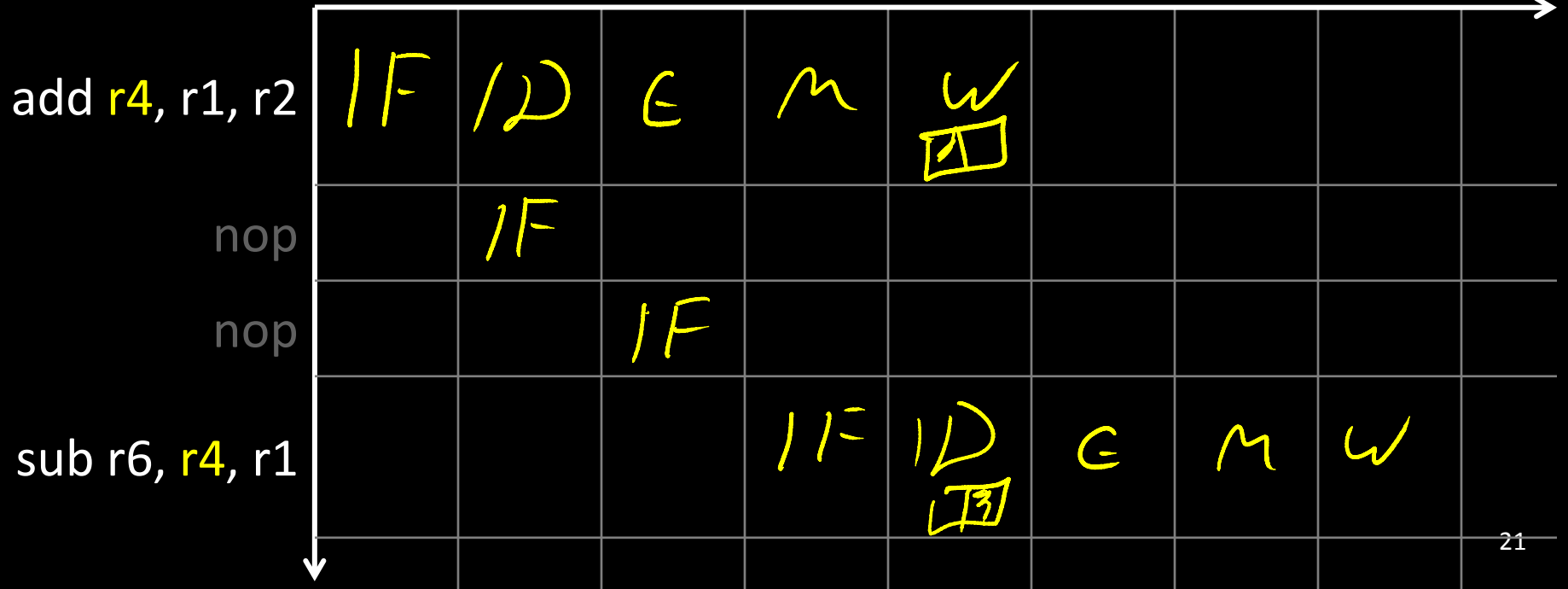
r1=15

OR r1, r4, r1



# More Data Hazards



# Register File Bypass

## Register File Bypass

- Reading a value that is currently being written

Detect:

((Ra == MEM/WB.Rd) or (Rb == MEM/WB.Rd))
and (WB is writing a register)

Resolve:

Add a bypass around register file (WB to ID)

Better: (Hack) just negate register file clock

- writes happen at end of first half of each clock cycle
- reads happen during second half of each clock cycle

# Administrivia

Prelim1: next Tuesday, February 28th in evening

- Location: GSH132: Goldwin Smith Hall room 132
- Time: We will start at 7:30pm sharp, so come early
- Prelim Review: This Wed / Fri, 3:30-5:30pm, in 155 Olin

- Closed Book
  - Cannot use electronic device or outside material
- Practice prelims are online in CMS
- Material covered everything up to end of this week
  - Appendix C (logic, gates, FSMs, memory, ALUs)
  - Chapter 4 (pipelined [and non-pipeline] MIPS processor with hazards)
  - Chapters 2 (Numbers / Arithmetic, simple MIPS instructions)
  - Chapter 1 (Performance)
  - HW1, HW2, Lab0, Lab1, Lab2

# Administrivia

HW2 was due two days ago!

- ***Fill out Survey online***. Receive credit/points on homework for survey:
- https://cornell.qualtrics.com/SE/?SID=SV_5olFfZiXoWz6pKI
- Survey is anonymous

Project1 (PA1) due week after prelim

- Continue working diligently. Use design doc momentum

Save your work!

- ***Save often***. Verify file is non-zero. Periodically save to Dropbox, email.
- Beware of MacOSX 10.5 (leopard) and 10.6 (snow-leopard)

Use your resources

- Lab Section, Piazza.com, Office Hours, Homework Help Session,
- Class notes, book, Sections, CSUGLab

# Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

- Tuesday, February 28th
- Thursday, March 29th
- Thursday, April 26th

Schedule is subject to change

# Collaboration, Late, Re-grading Policies

## "Black Board" Collaboration Policy

- Can discuss approach together on a "black board"
- Leave and write up solution independently
- Do not copy solutions

## Late Policy

- Each person has a total of ***four*** "slip days"
- Max of ***two*** slip days for any individual assignment
- Slip days deducted first for *any* late assignment, cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- 20% deducted per day late after slip days are exhausted

## Regrade policy

- Submit written request to lead TA, and lead TA will pick a different grader
- Submit another written request, lead TA will regrade directly
- Submit yet another written request for professor to regrade.

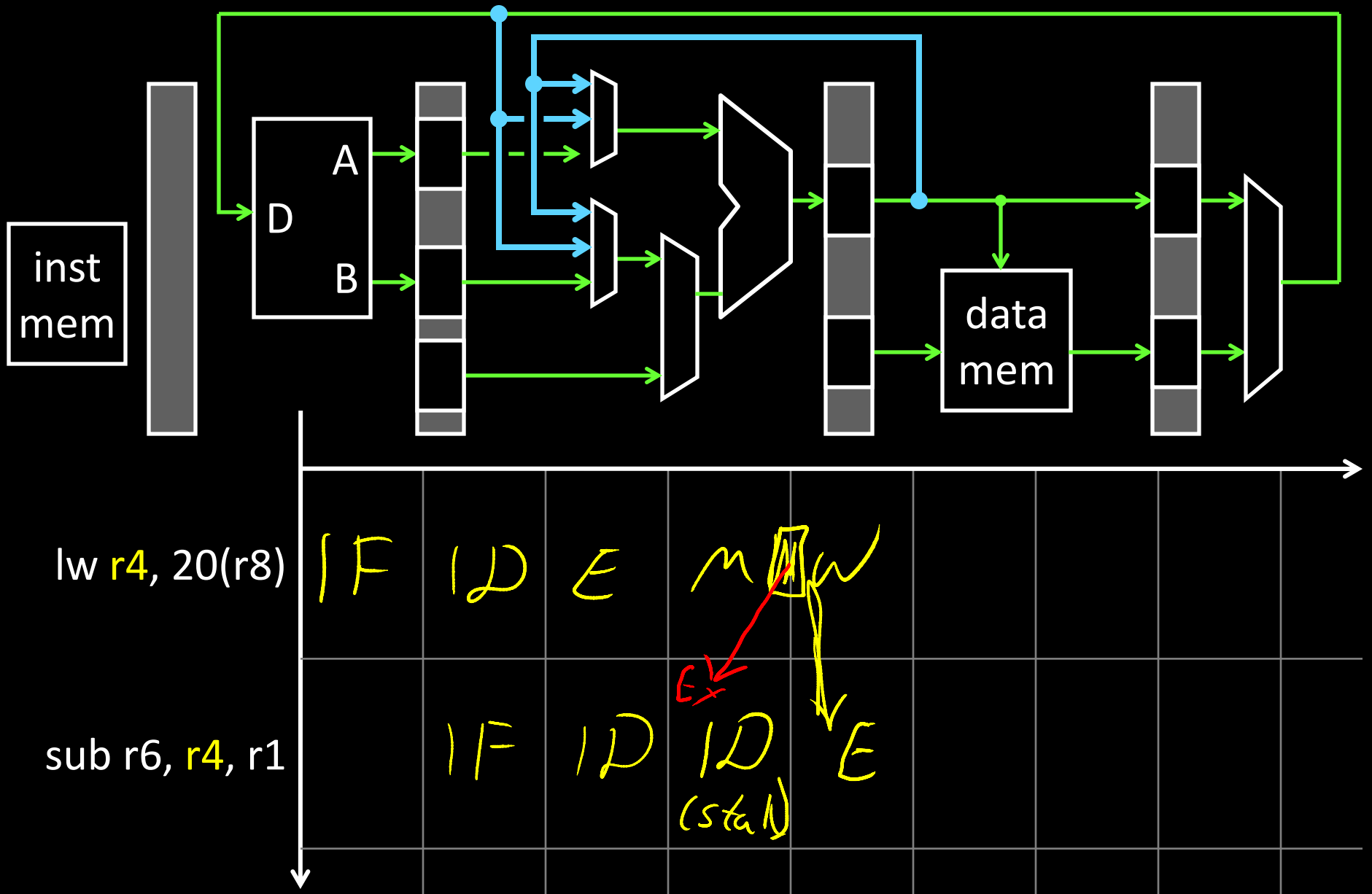
# Quiz

Find all hazards, and say how they are resolved:

```
add    r3, r1, r2
sub    r3, r2, r1
nand   r4, r3, r1
or     r0, r3, r4
xor    r1, r4, r3
sb     r4, 1(r0)
```

hours & hours of debugging

# Memory Load Data Hazard



# Resolving Memory Load Hazard

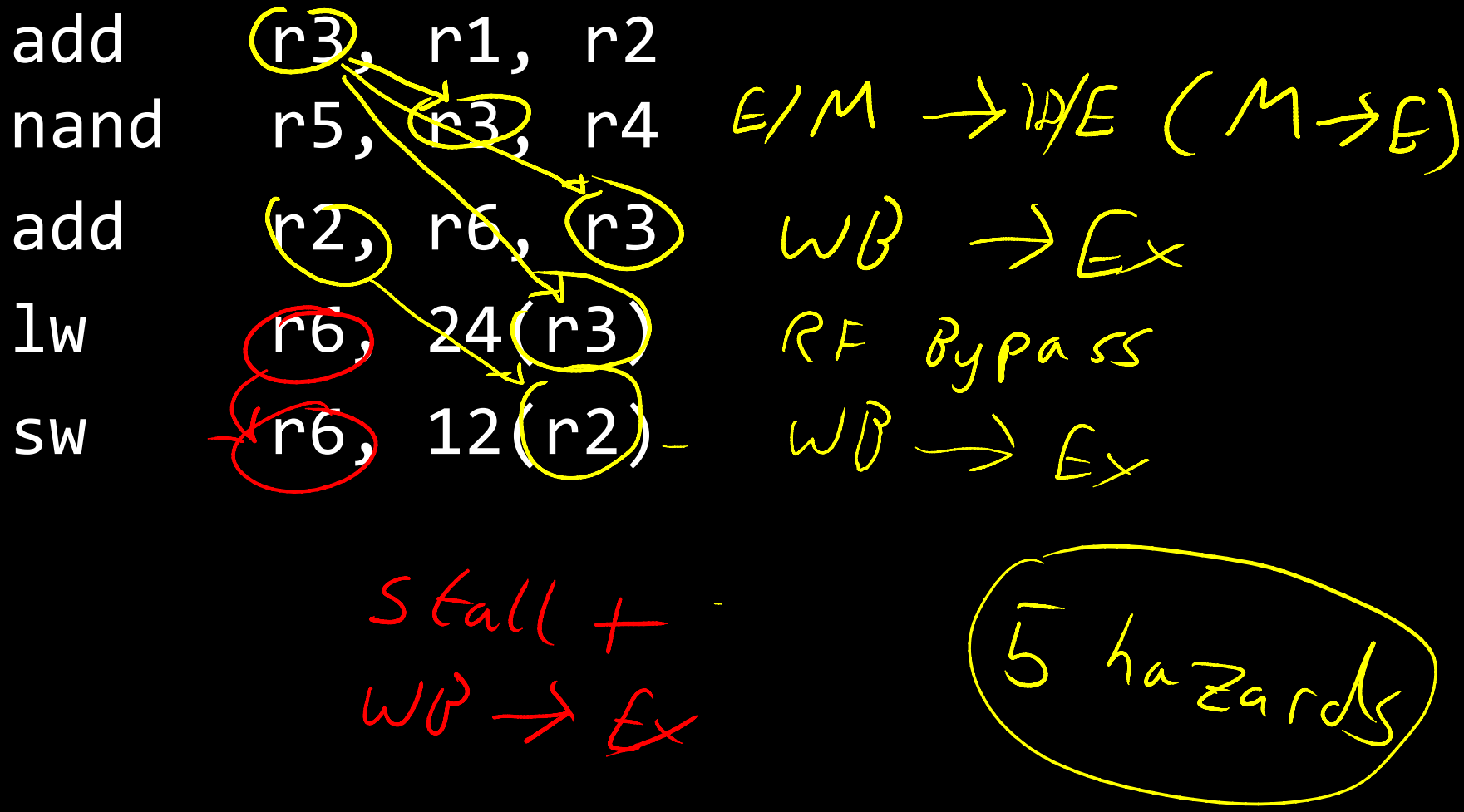
Load Data Hazard

- Value not available until WB stage
- So: next instruction can't proceed if hazard detected

Resolution:

- MIPS 2000/3000: one delay slot
  - ISA says results of loads are not available until one cycle later
  - Assembler inserts nop, or reorders to fill delay slot
- MIPS 4000 onwards: stall
  - But really, programmer/compiler reorders to avoid stalling in the load delay slot

# Quiz 2


add r3, r1, r2
nand r5, r3, r4
add r2, r6, r3
lw r6, 24(r3)
sw r6, 12(r2)
E/M → D/E (M → E)
WB → Ex
RF Bypass
WB → Ex
Stall +
WB → Ex
5 hazards

# Data Hazard Recap

## Delay Slot(s)

- Modify ISA to match implementation

## Stall

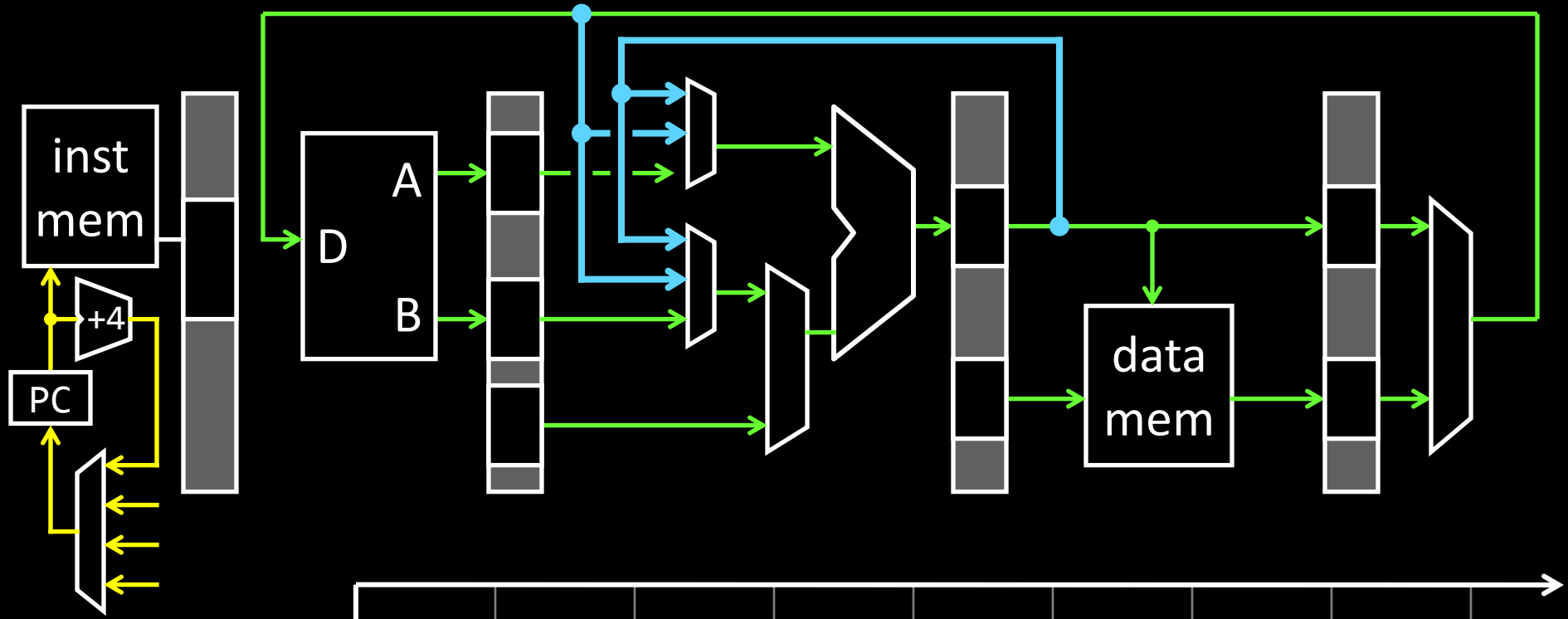
- Pause current and all subsequent instructions

## Forward/Bypass

- Try to steal correct value from elsewhere in pipeline
- Otherwise, fall back to stalling or require a delay slot

Tradeoffs?

# More Hazards




beq r1, r2, L

add r3, r0, r3

sub r5, r4, r6

L: or r3, r2, r4

# Control Hazards

## Control Hazards

- instructions are fetched in stage 1 (IF)
- branch and jump decisions occur in stage 3 (EX)
- i.e. next PC is not known until 2 cycles after branch/jump

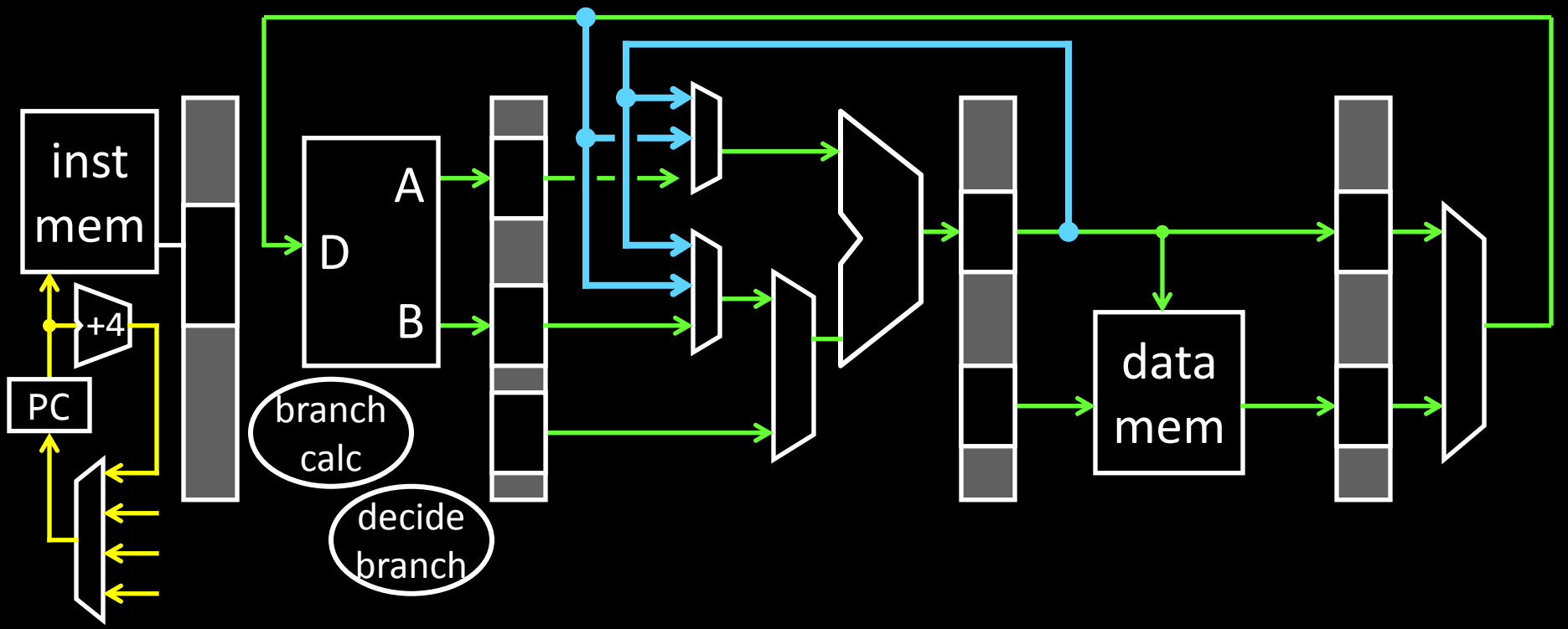
## Delay Slot

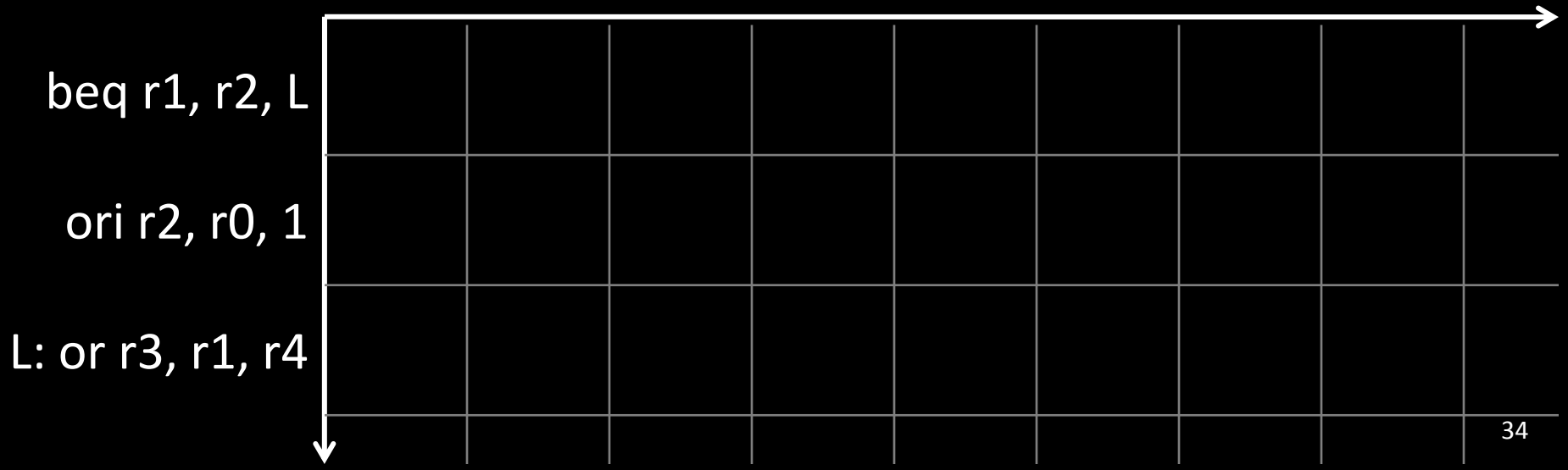
- ISA says N instructions after branch/jump always executed
  - MIPS has 1 branch delay slot

## Stall (+ Zap)

- prevent PC update
- clear IF/ID pipeline register
  - instruction just fetched might be wrong one, so convert to nop
- allow branch to continue into EX stage

# Delay Slot


inst
mem
+4
PC
A
D
B
branch
calc
decide
branch
data
mem


beq r1, r2, L
ori r2, r0, 1
L: or r3, r1, r4

# Control Hazards: Speculative Execution

## Control Hazards

- instructions are fetched in stage 1 (IF)
- branch and jump decisions occur in stage 3 (EX)
- i.e. next PC not known until 2 cycles after branch/jump

## Stall

## Delay Slot

## Speculative Execution

- Guess direction of the branch
  - Allow instructions to move through pipeline
  - Zap them later if wrong guess
- Useful for long pipelines

# Loops

# Branch Prediction

# Pipelining: What Could Possibly Go Wrong?

## Data hazards

- register file reads occur in stage 2 (IF)
- register file writes occur in stage 5 (WB)
- next instructions may read values soon to be written

## Control hazards

- branch instruction may change the PC in stage 3 (EX)
- next instructions have already started executing

## Structural hazards

- resource contention
- so far: impossible because of ISA and pipeline design