

# Pipeline Hazards

**Hakim Weatherspoon**

**CS 3410, Spring 2011**

Computer Science

Cornell University

See P&H Appendix 4.7

# Announcements

---

*PA1 available: mini-MIPS processor*

PA1 due next Friday

Work in **pairs**

Use your ~~own~~ resources

- FAQ, class notes, book, Sections, office hours, newsgroup, CSUGLab

HW1 graded

- Max: 10; Median: 9; Mean: 8.3; Stddev: 1.8
- *Great job!*
- Regrade policy
  - Submit written request to lead TA, 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.

# Announcements

---

## Prelims:

- Thursday, March 10<sup>th</sup> **in class**
- Thursday, April 28<sup>th</sup> Evening

## Late Policy

- 1) Each person has a total of **four “slip days”**
- 2) For projects, slip days are deducted from all partners
- 3) 10% deducted per day late after slip days are exhausted

# 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?



# 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:

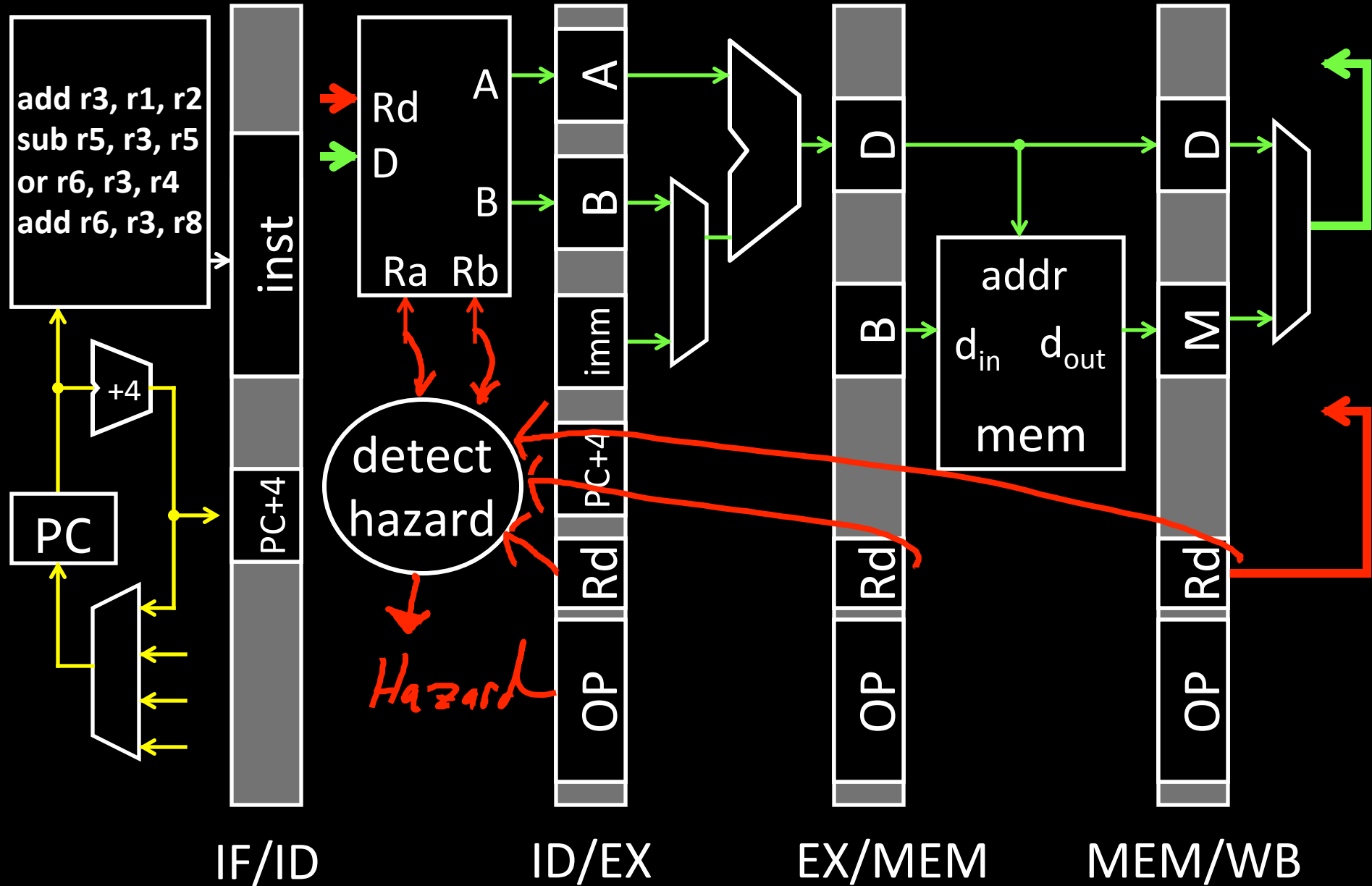
stall = (ID.rA != 0 && (ID.rA == EX.rD ||

*E/ID.rA* ID.rA == M.rD ||

ID.rA == WB.rD))

|| (same for rB)

# Detecting Data Hazards



# Resolving Data Hazards

---

What to do if data hazard detected?

wait/stall

Pause the current inst  
and subsequent inst

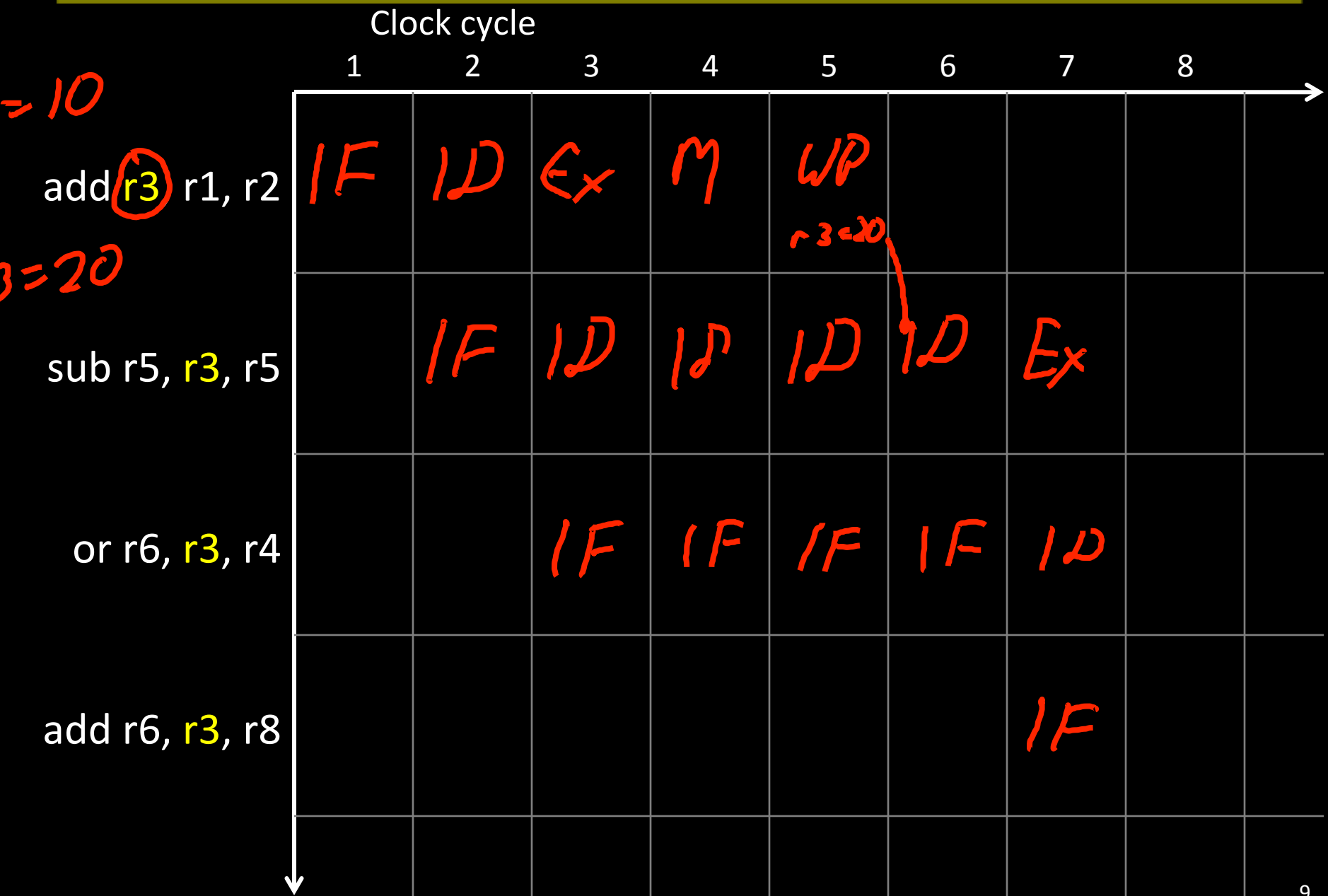
reorder inst

Forward/Bypass

steal the value u  
need from some later



# Stalling





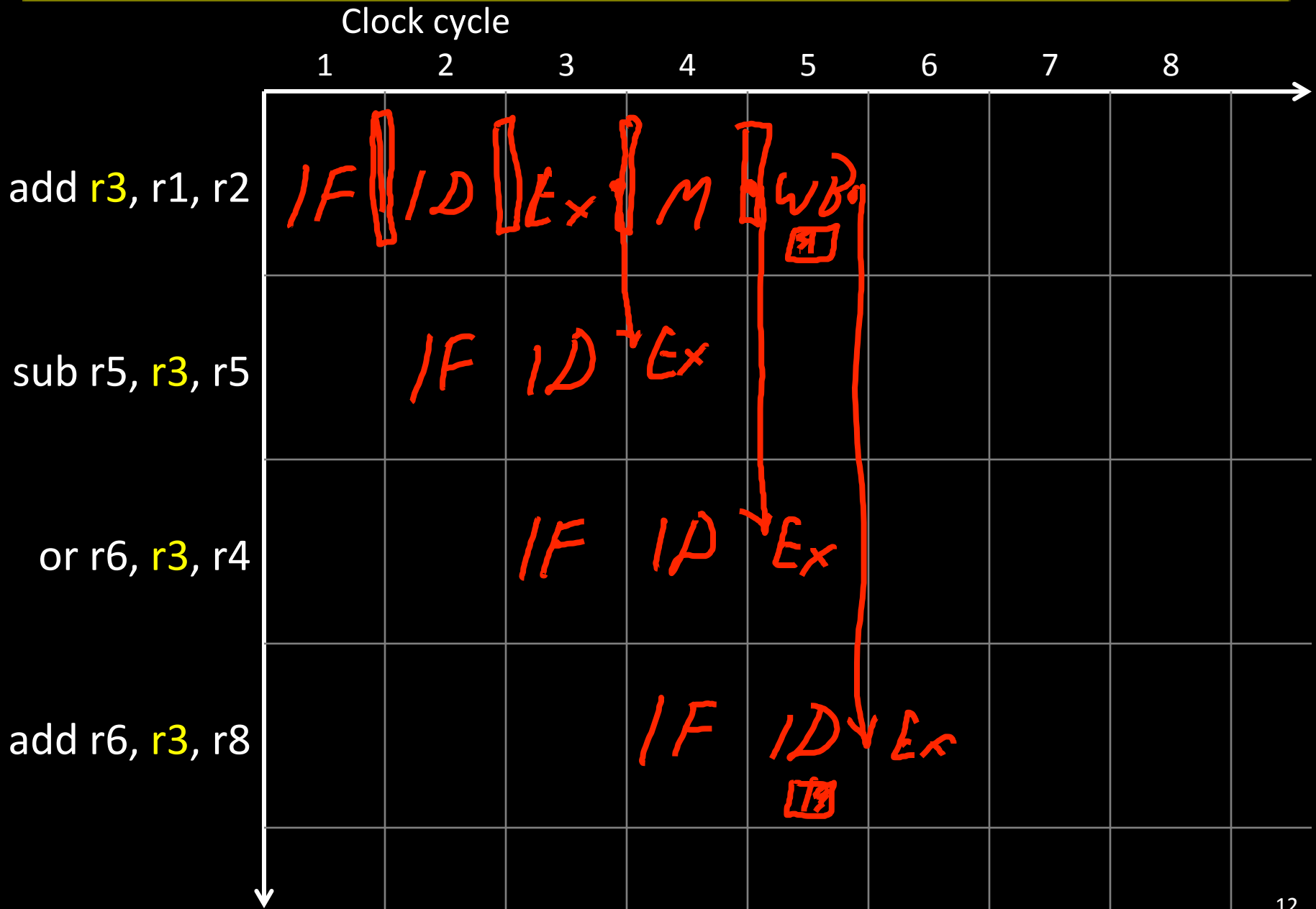
# Stalling

## How to stall an instruction in ID stage

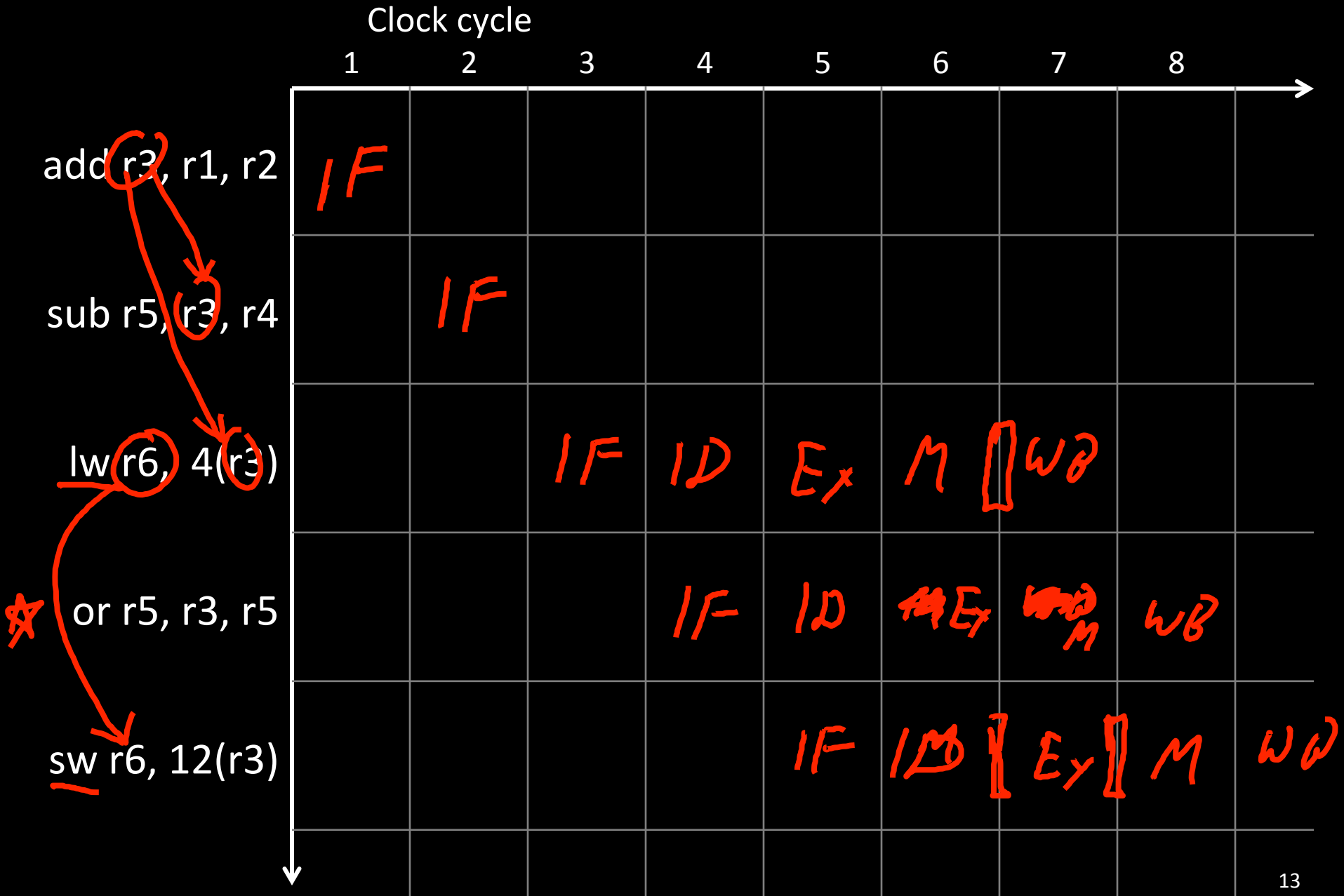
- prevent IF/ID pipeline register update
  - stalls the ID stage instruction
- convert ID stage instr into **nop** for later stages
  - innocuous “bubble” passes through pipeline
- prevent PC update
  - stalls the next (IF stage) instruction

deassert  
all  
control  
signals  
no reg's or  
mem w

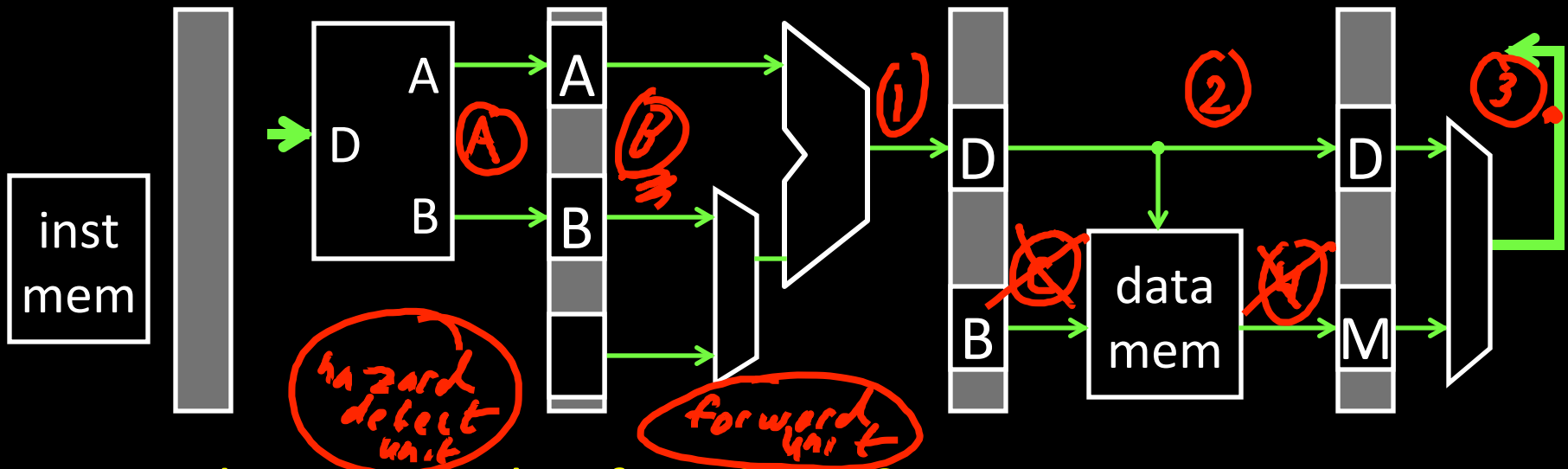
# Forwarding



# Forwarding



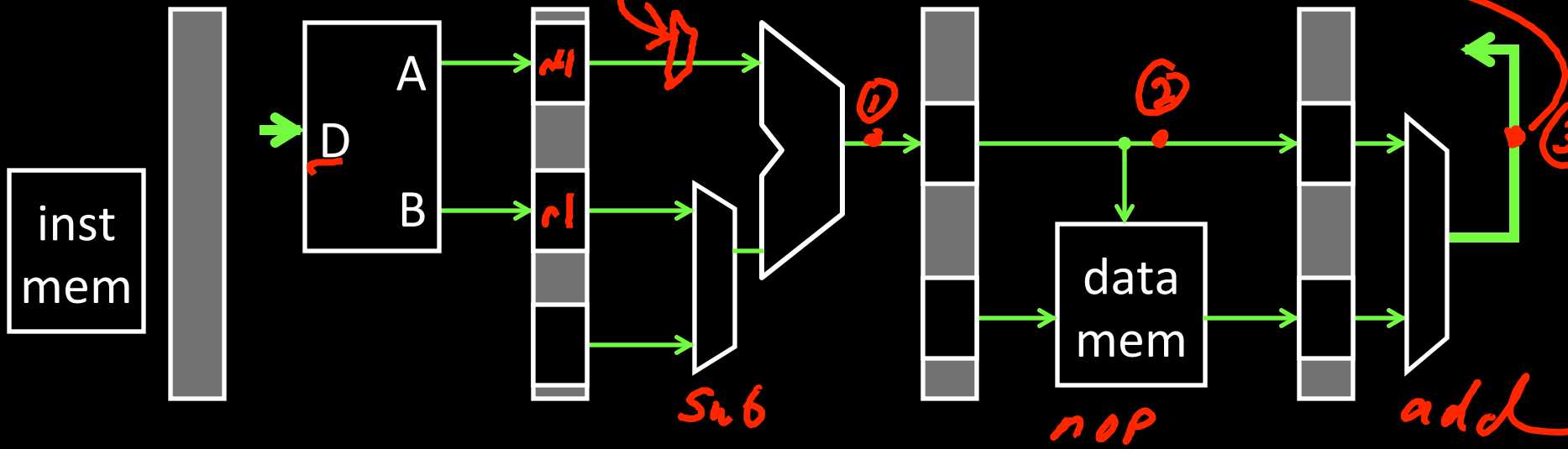
# Forwarding



Forward correct value from? to?

1. ALU output: too late in cycle?
  - 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
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

# Forwarding Path 1



add r4, r1, r2	IF	ID	Ex	M	WB				
nop		IF	ID	Ex					
sub r6, <u>r4</u> , r1			IF	ID	Ex				
				r4					

# WB to EX Bypass

---

## WB to EX Bypass

- EX needs value being written by WB

Resolve:

Add bypass from WB final value to start of EX

Detect:

$(Ex.Ra = WB.Rd \text{ or } Ex.Rb = WB.Rd)$   
 $\text{and } WB.WE \text{ and } WB.Rd \neq 0$   
~~and not (Ex needs A or B)~~





# MEM to EX Bypass

---

## MEM to EX Bypass

- EX needs ALU result that is still in MEM stage

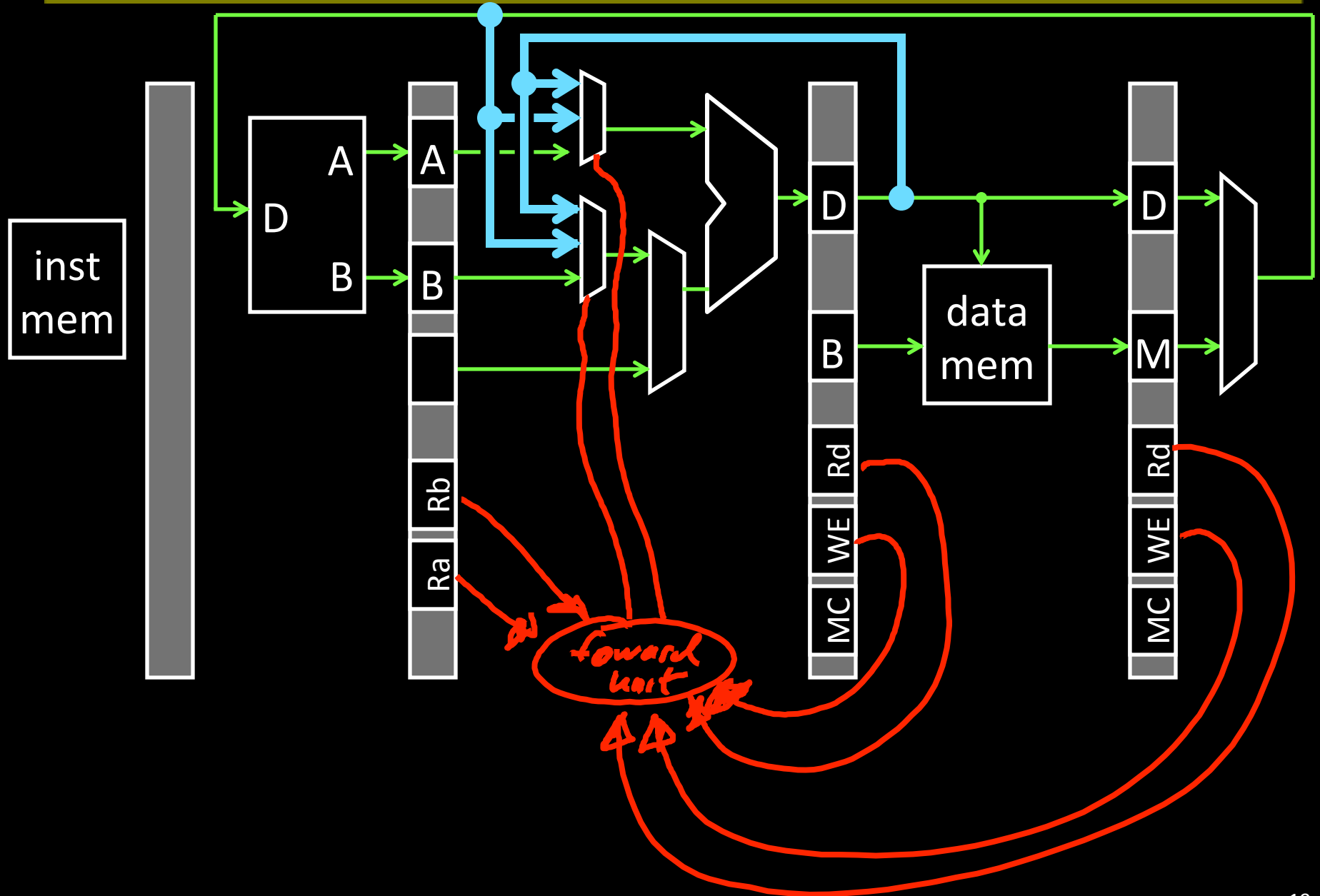
Resolve:

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

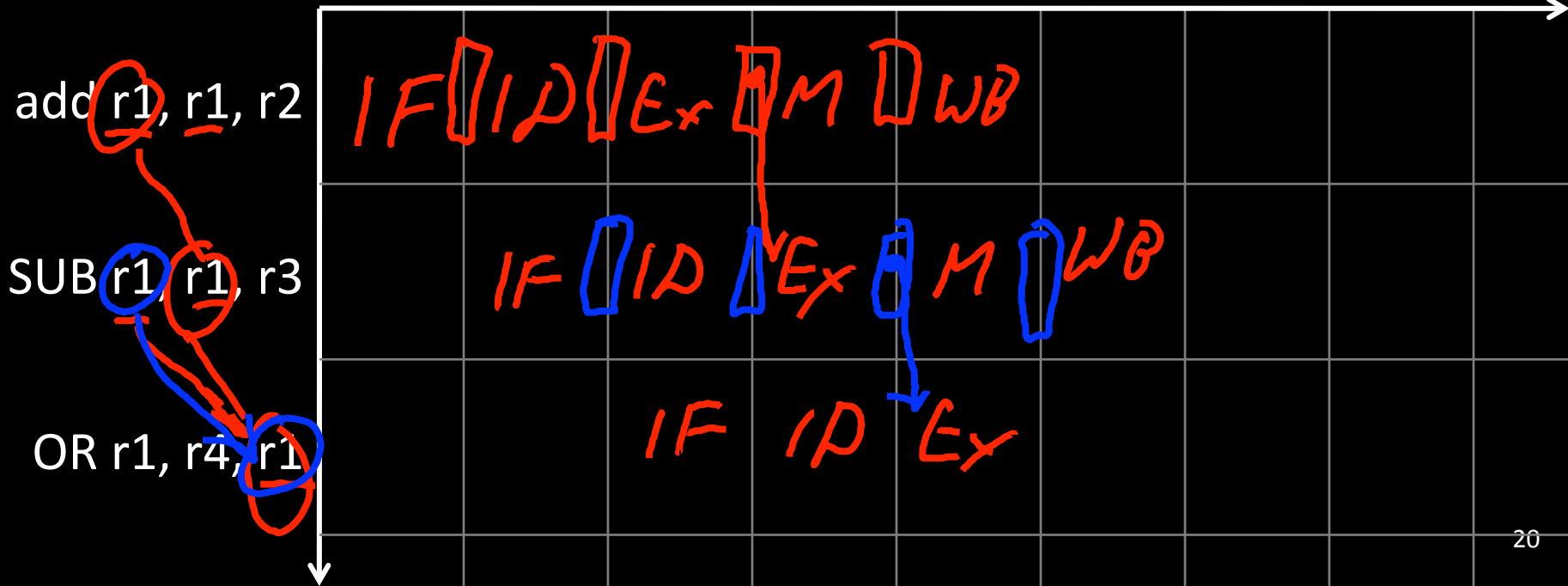
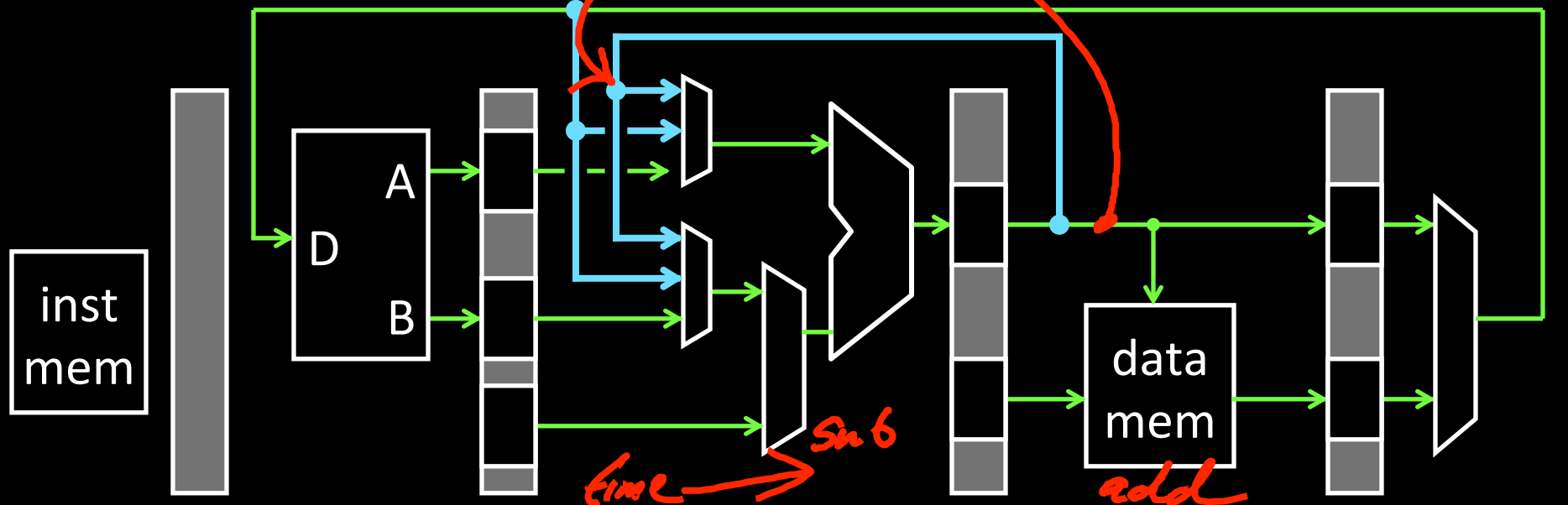
Detect:

$(Ex.Ra = Mem.Rd) \text{ or } (Ex.Rb = Mem.Rd)$   
and  $(Mem.WE \text{ and } Mem.Rd \neq 0)$  // write to  
to  
and

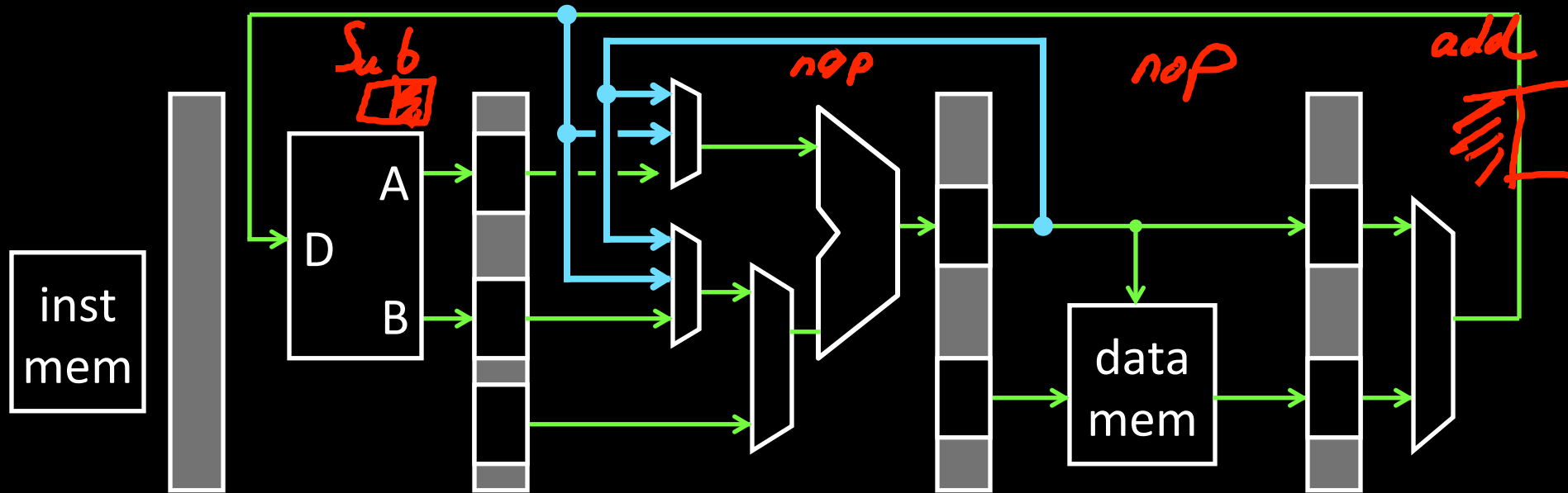
# Forwarding Datapath



# Tricky Example



# More Data Hazards



add r4, r1, r2	IF	ID	EX	M	WB				
<u>nop</u>		IF							
<u>nop</u>			IF						
sub r6, r4, r1				IF	ID				

# Register File Bypass

---

## Register File Bypass

- Reading a value that is currently being written

Detect:

$((Ra == MEM/WB.Rd) \text{ or } (Rb == MEM/WB.Rd))$   
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



# Quiz

Find all hazards, and say how they are resolved:

add r3, r1, r2

sub ~~r3~~, r2, r1

nand ~~r4~~, ~~r3~~, r1

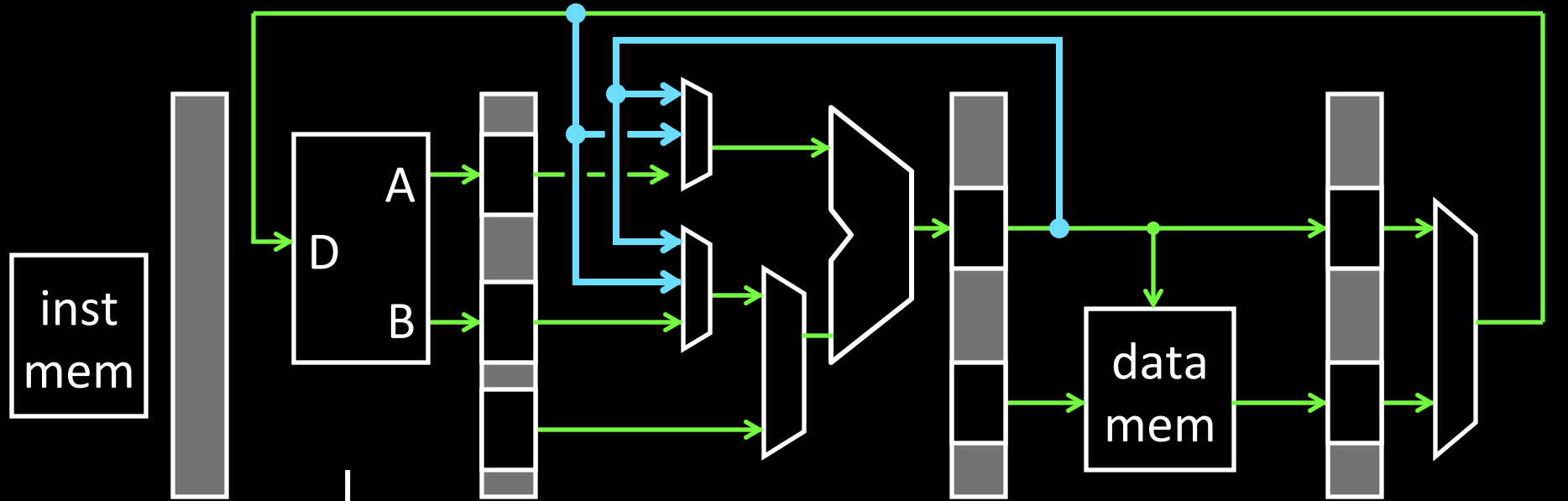
or ~~r0~~, ~~r3~~, ~~r4~~

xor r1, ~~r4~~, ~~r3~~

sb r4, 1(~~r0~~)

hours & hours  
debugging  
r0 = 0

# Memory Load Data Hazard



lw r4, 20(r8)

IF [ ] ID [ ] EX [ ] MEM [ ] WB [ ]

sub r6, r4, r1

IF [ ] ID [ ] EX [ ]

can't do



# Resolving Memory Load Hazard

---

## Load Data Hazard

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

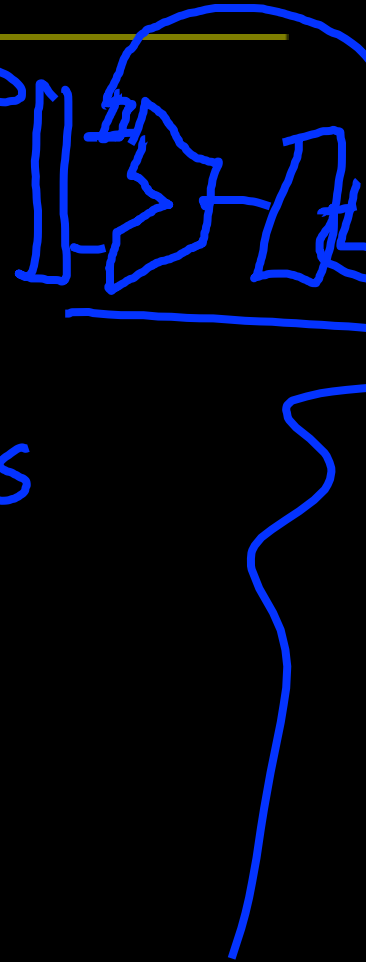
## Resolution:

- 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)

M → Ex  
WB → Ex  
RF Bypass  
WB → Ex



# Data Hazard Recap

---

## Delay Slot(s)

- Modify ISA to match implementation

## Stall

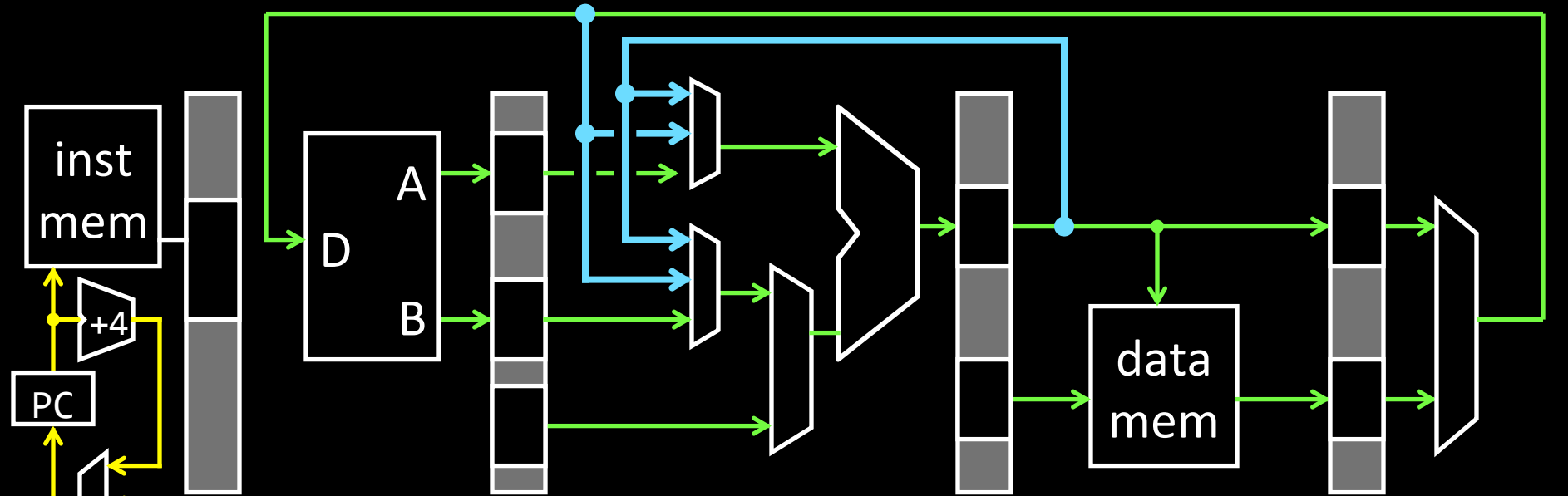
- Pause current and all subsequent instructions

## Forward/Bypass

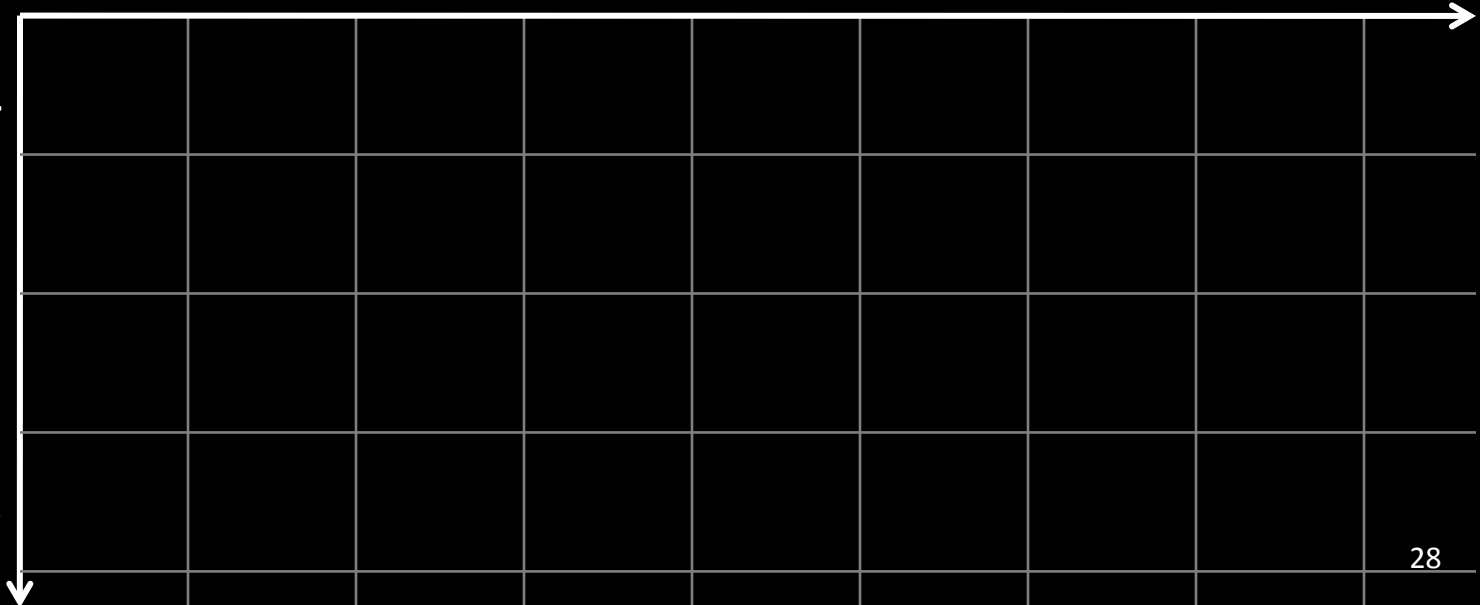
- Try to steal correct value from elsewhere in pipeline
- Otherwise, fall back to stalling or require a delay slot

Tradeoffs?

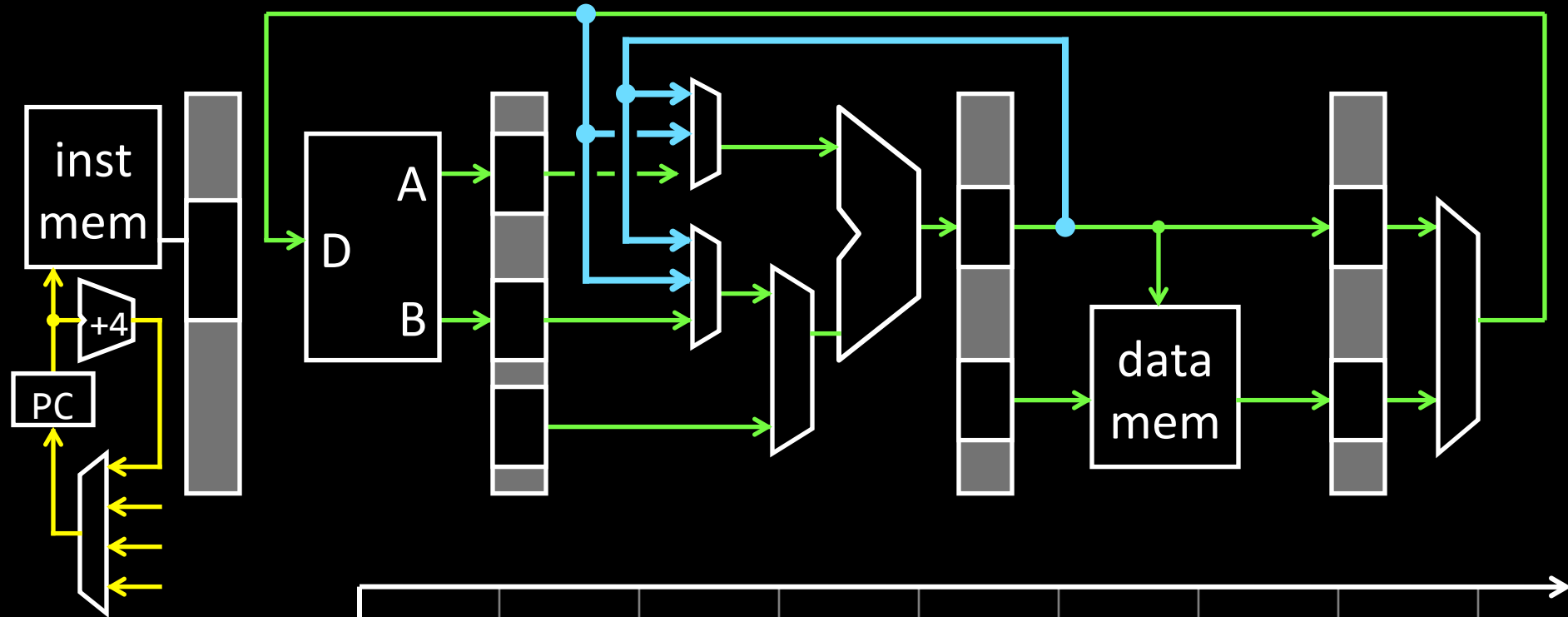
# More Hazards



beq r1, r2, L  
add r3, r0, r3  
sub r5, r4, r6  
L: or r3, r2, r4



# 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

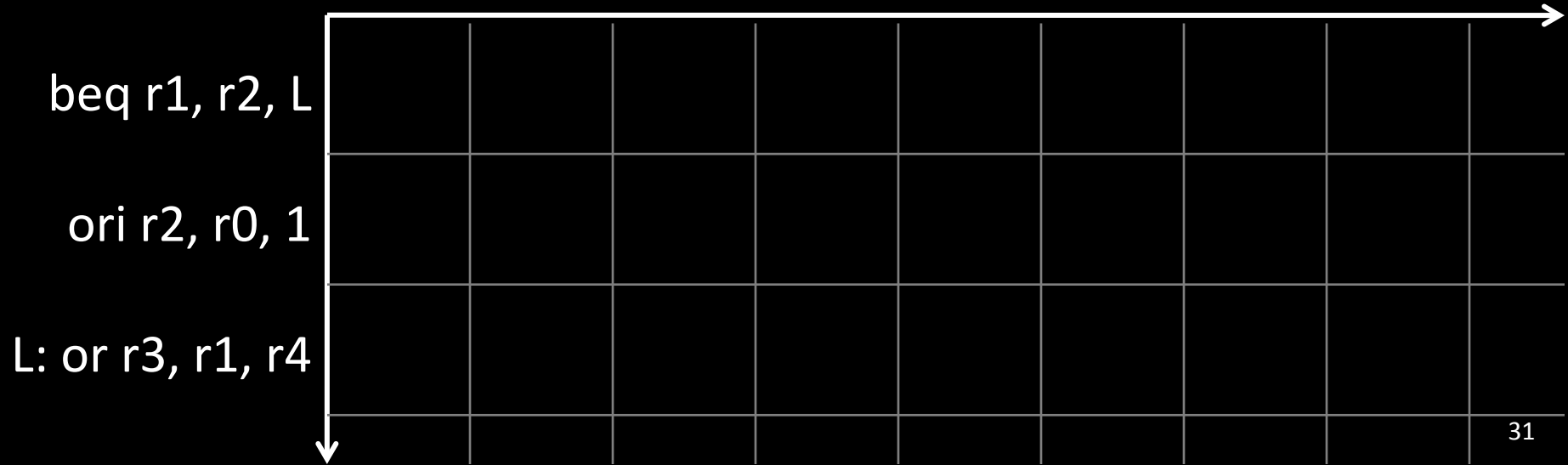
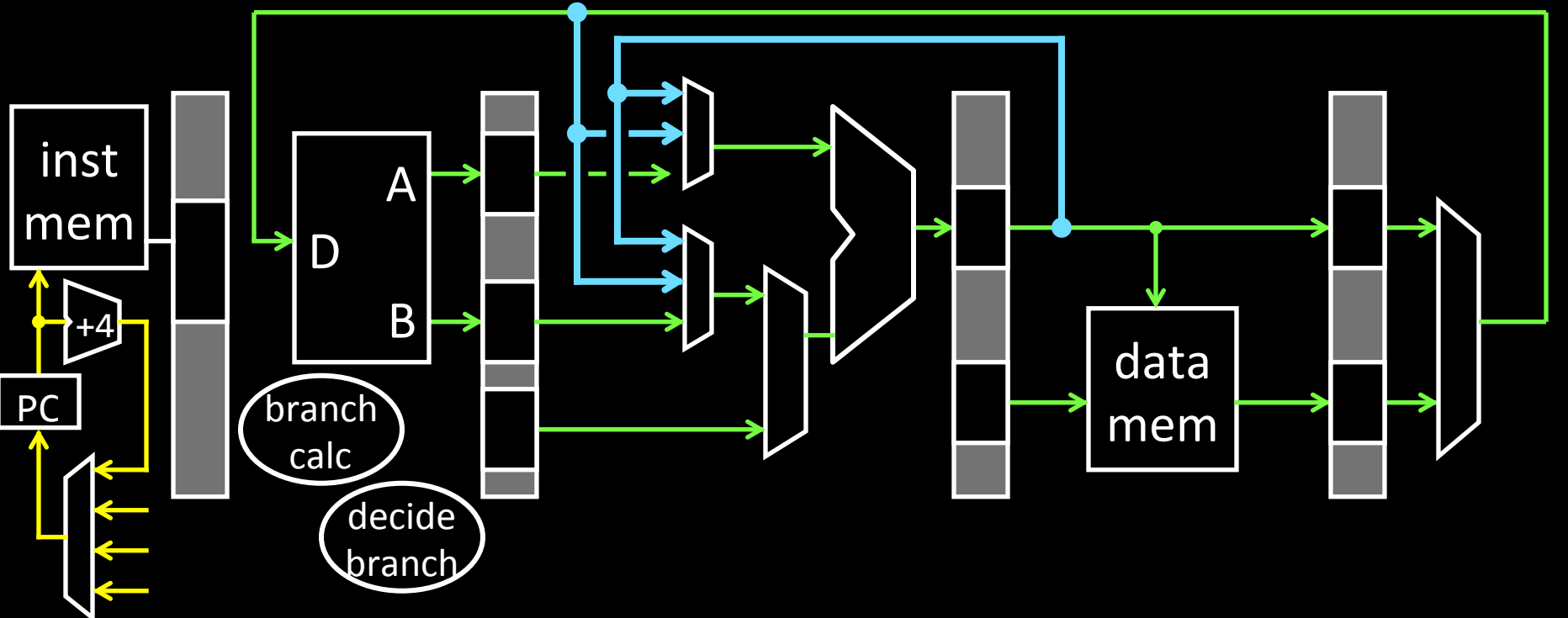
## Delay Slot

- 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



# 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