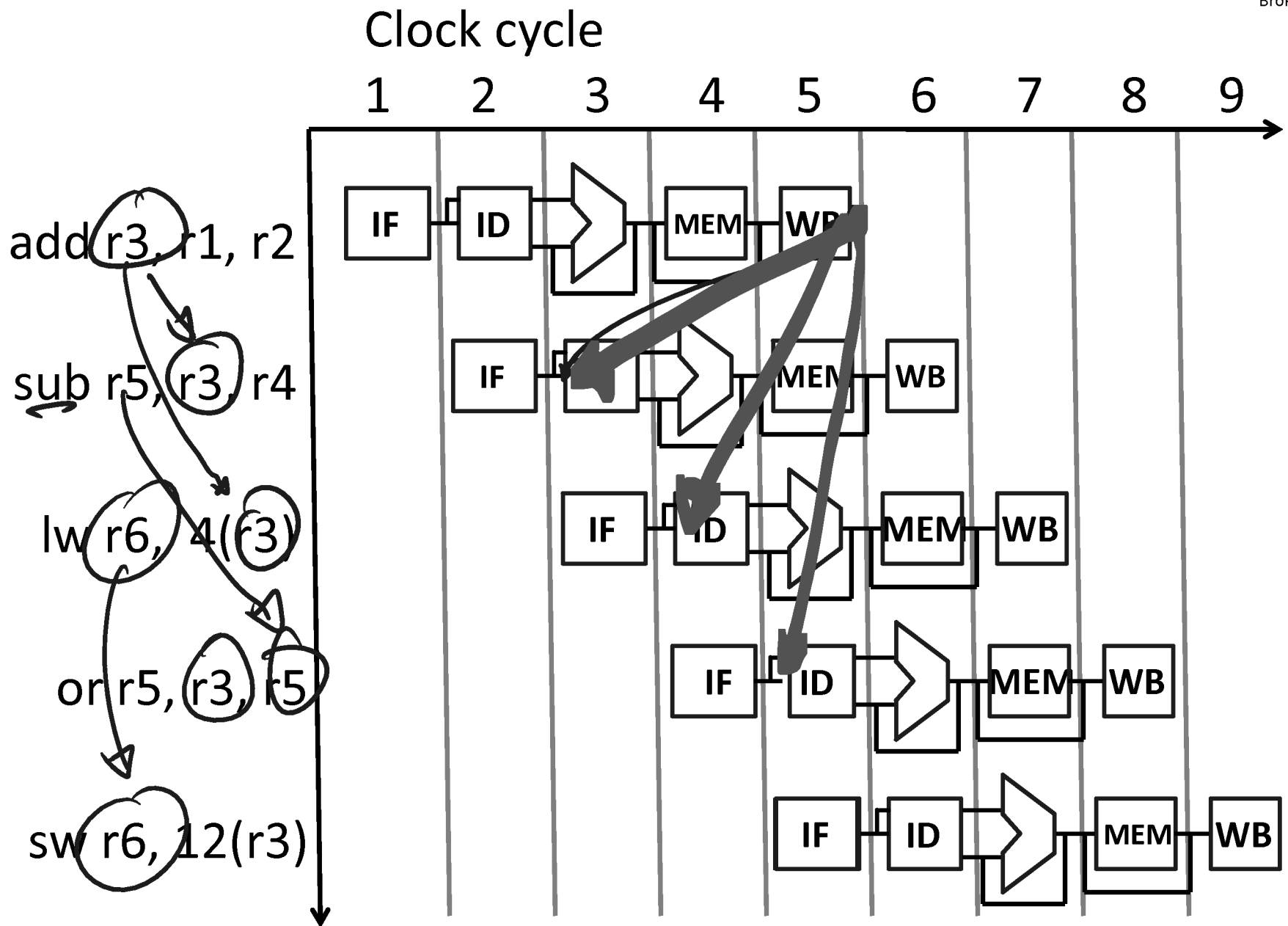


# Pipeline Hazards

**Kevin Walsh**  
**CS 3410, Spring 2010**  
Computer Science  
Cornell University

See: P&H Chapter 4.7

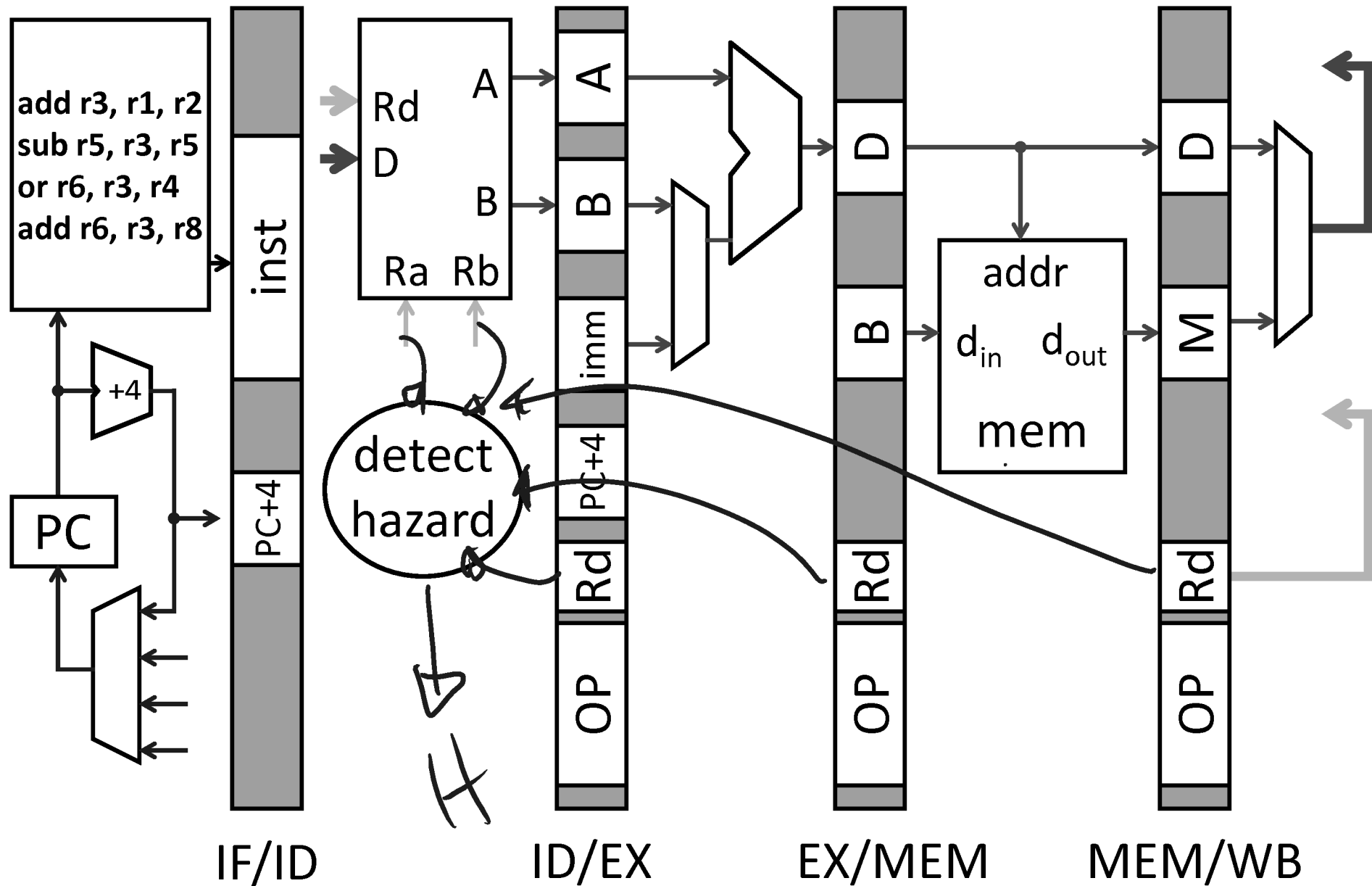


## Data Hazards

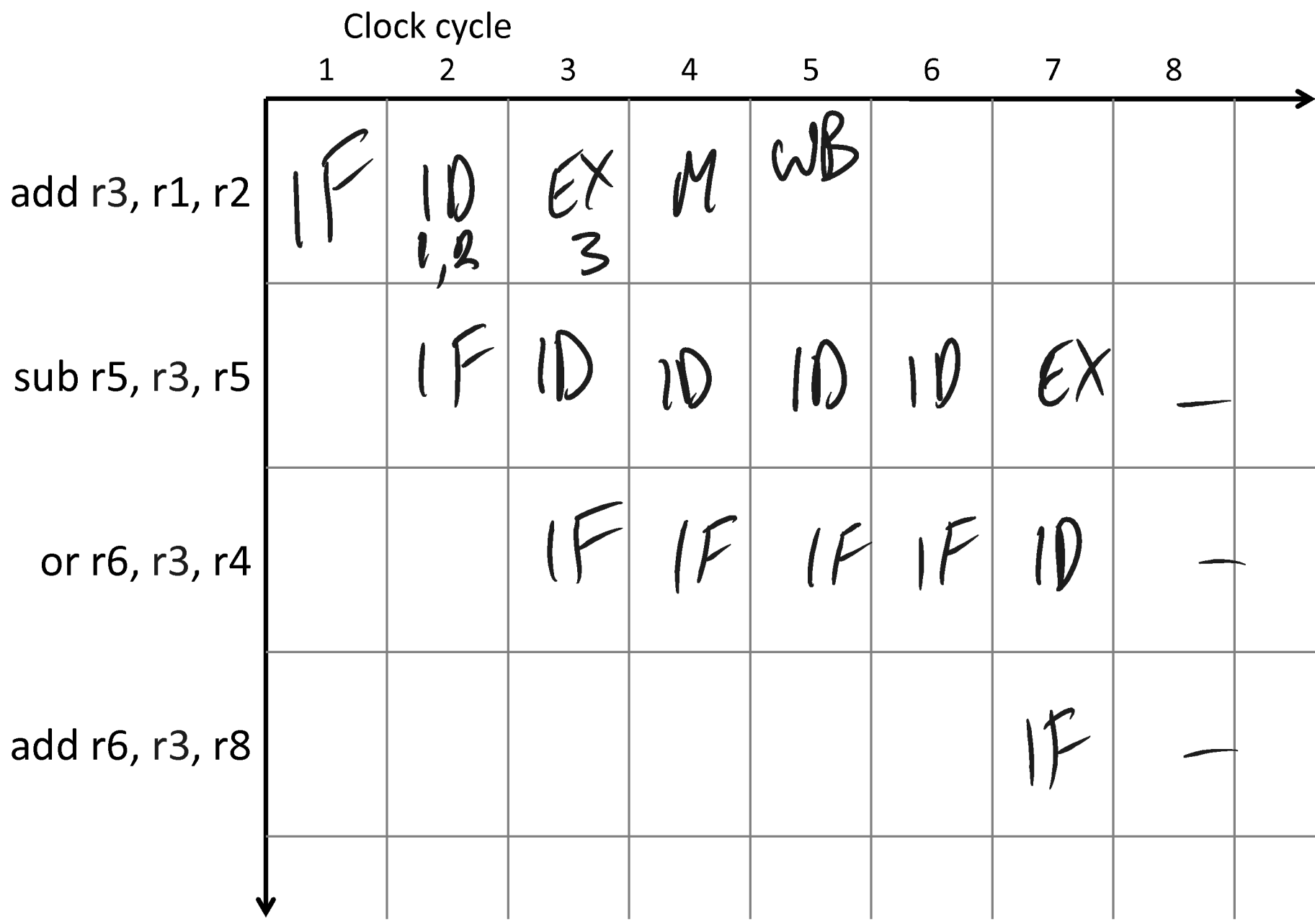
- register file reads occur in stage 2 (IF)
- register file writes occur in stage 5 (WB)
- next instructions may read values about to be written

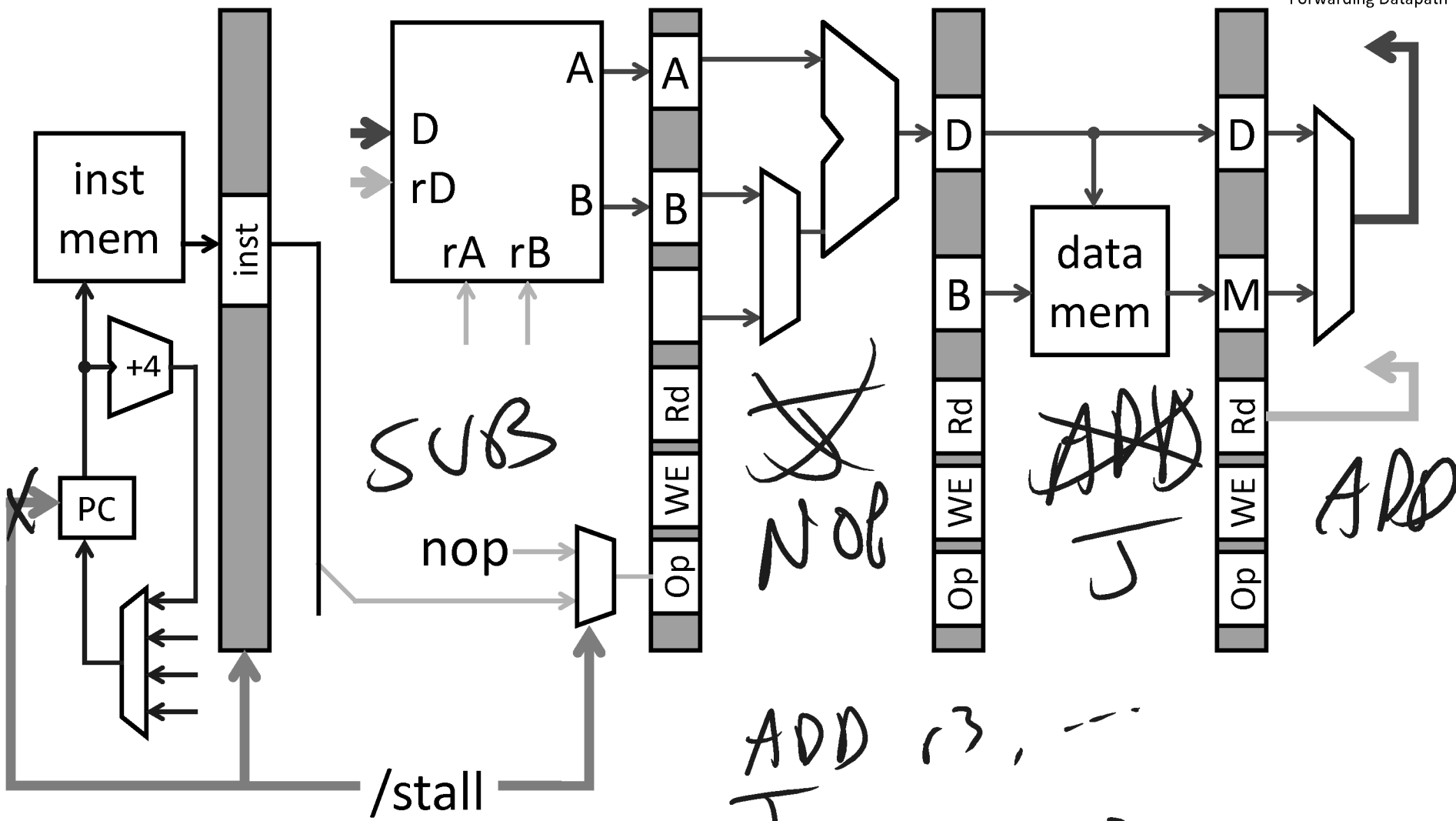
How to detect? Logic in ID stage:

$$\begin{aligned} \text{stall} = & (\text{ID.rA} \neq 0 \ \&\& \ (\text{ID.rA} == \text{EX.rD} \ || \\ & \text{ID.rA} == \text{M.rD} \ || \\ & \text{ID.rA} == \text{WB.rD})) \\ & \ || \ (\text{same for rB}) \end{aligned}$$



# What to do if data hazard detected?



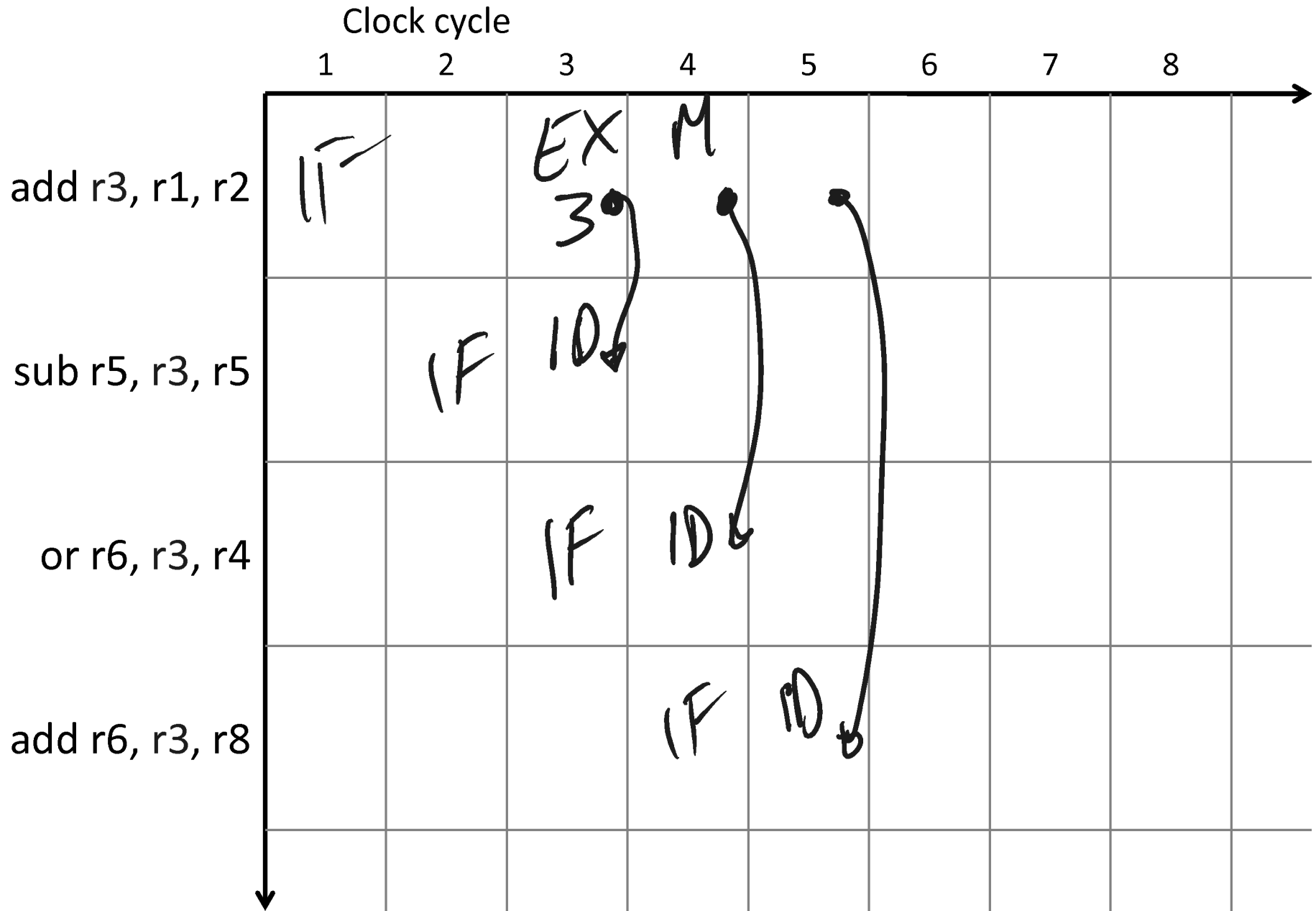


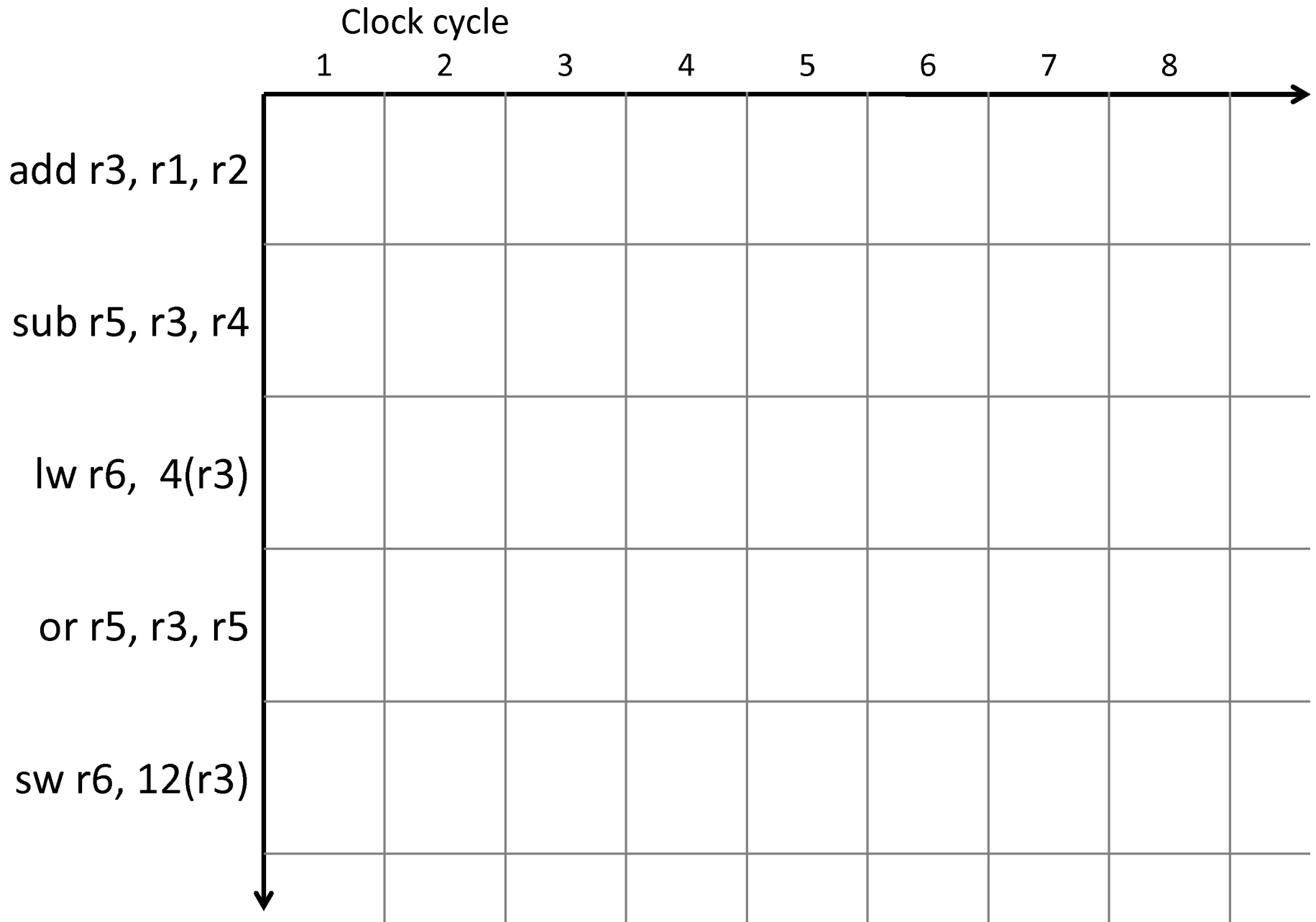
ADD r3, ...  
J  
SUB r4, r3, r0

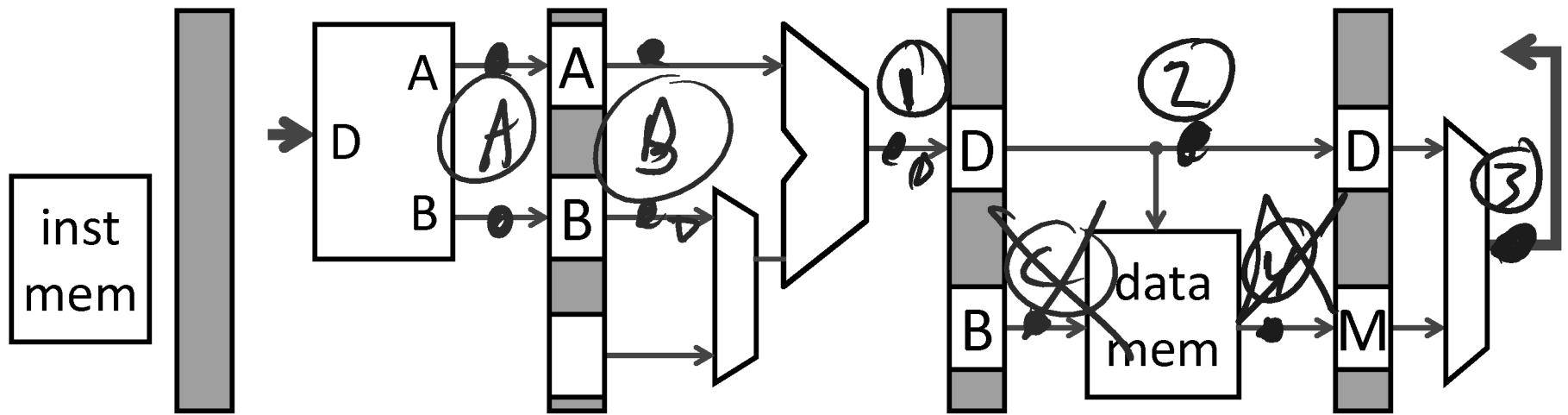
# How to stall an instruction in ID stage

- 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









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)

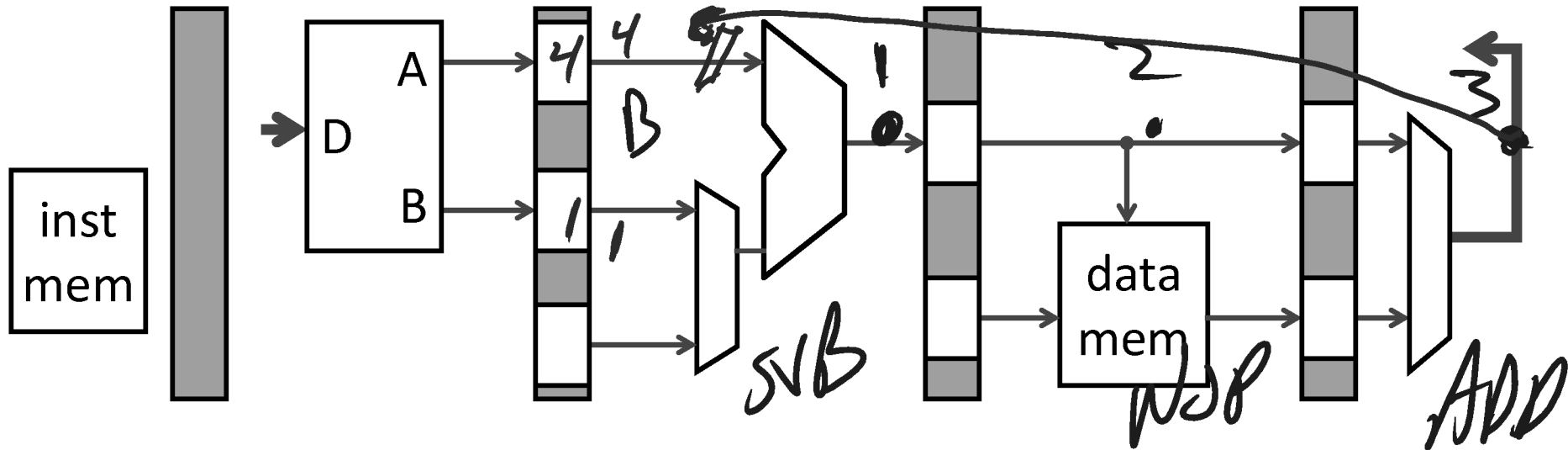
4. MEM output: too late in cycle, on critical path

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



LW

~~add~~ r4, r1, r2

Nop

sub r6, r4, r1

IF	ID	EX	M	WB				
	IF	ID	EX	M				
		IF	ID	EX				

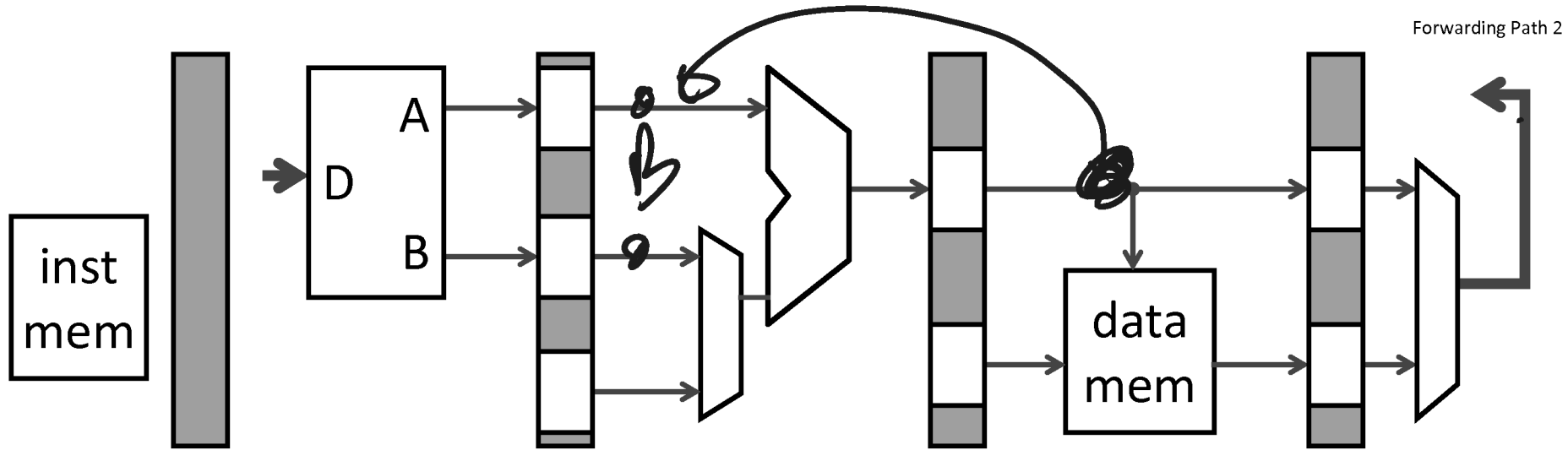
## WB to EX Bypass

- EX needs value being written by WB

Resolve:

Add bypass from WB final value to start of EX

Detect:



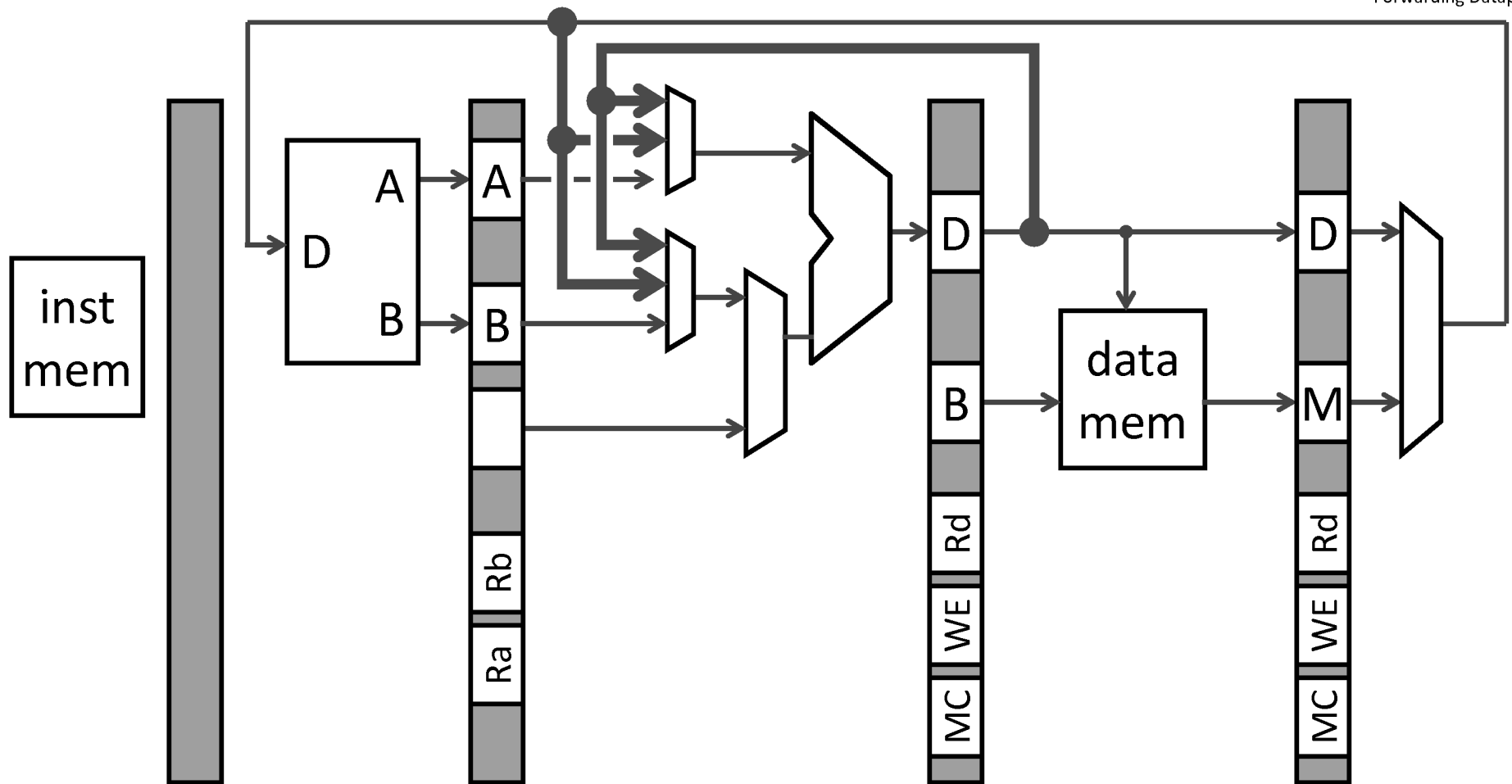
## MEM to EX Bypass

- EX needs ALU result that is still in MEM stage

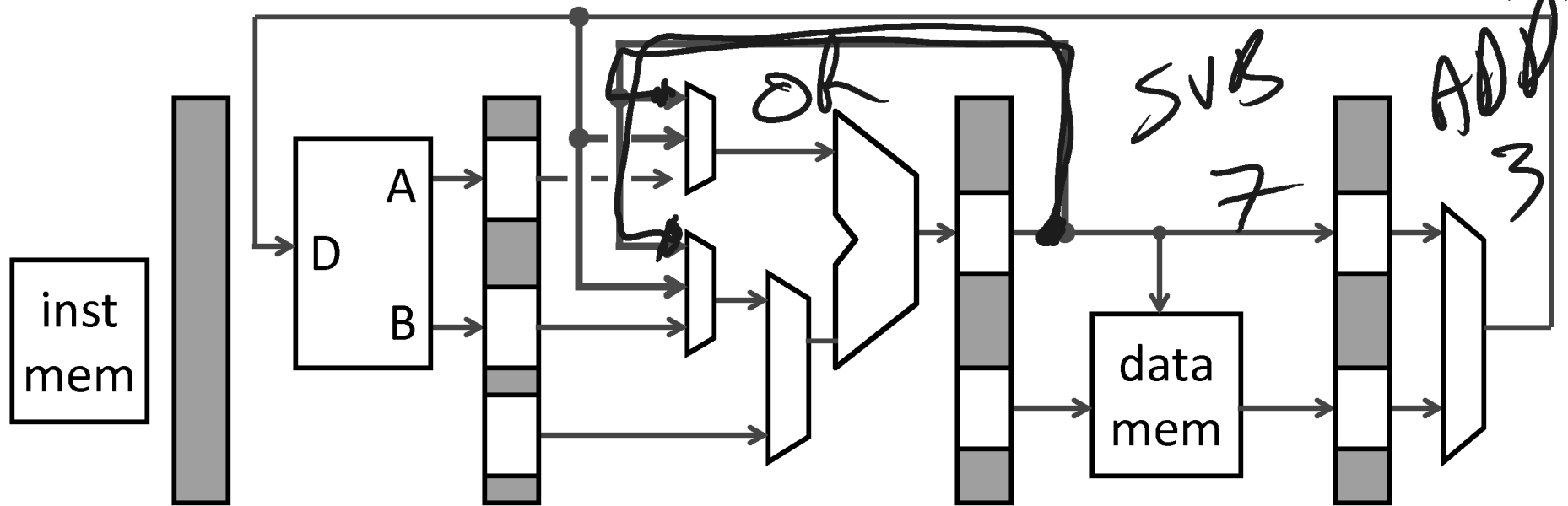
Resolve:

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

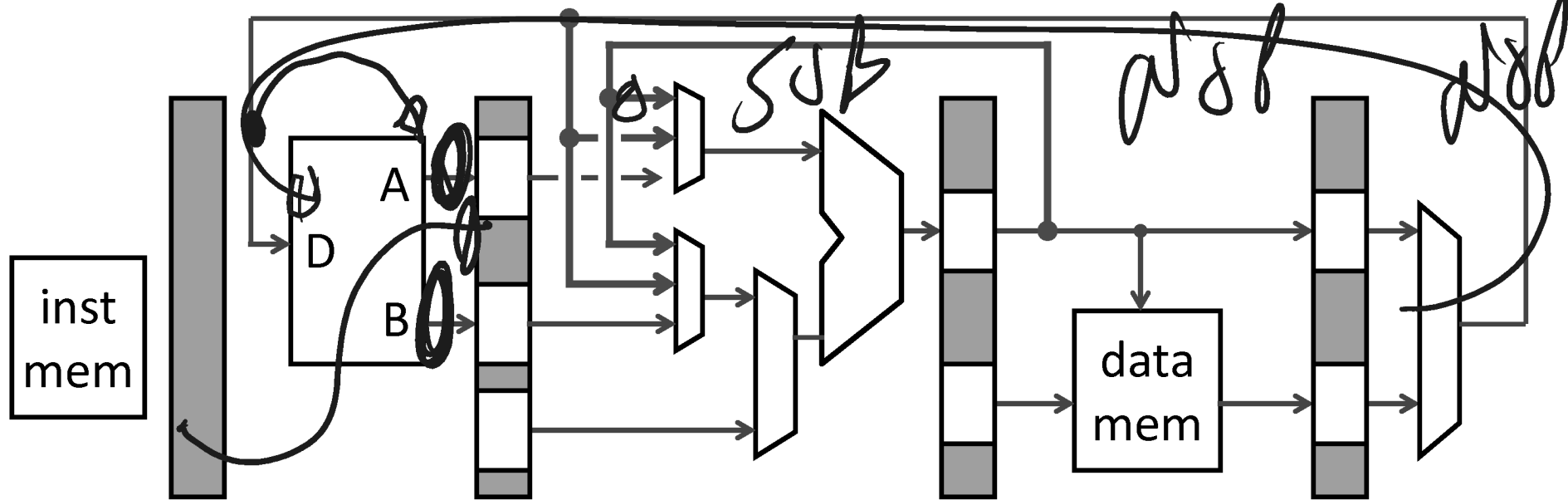
Detect:







	IF	ID	EX	M	WB				
add r1, r1, r2									
SUB r1, r1, r3									
OR r1, r4, r1									



add r4, r1, r2

IF ID X M WB

nop  
nop

IF

IF

sub r6, r4, r1

IF ID EX

# 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:

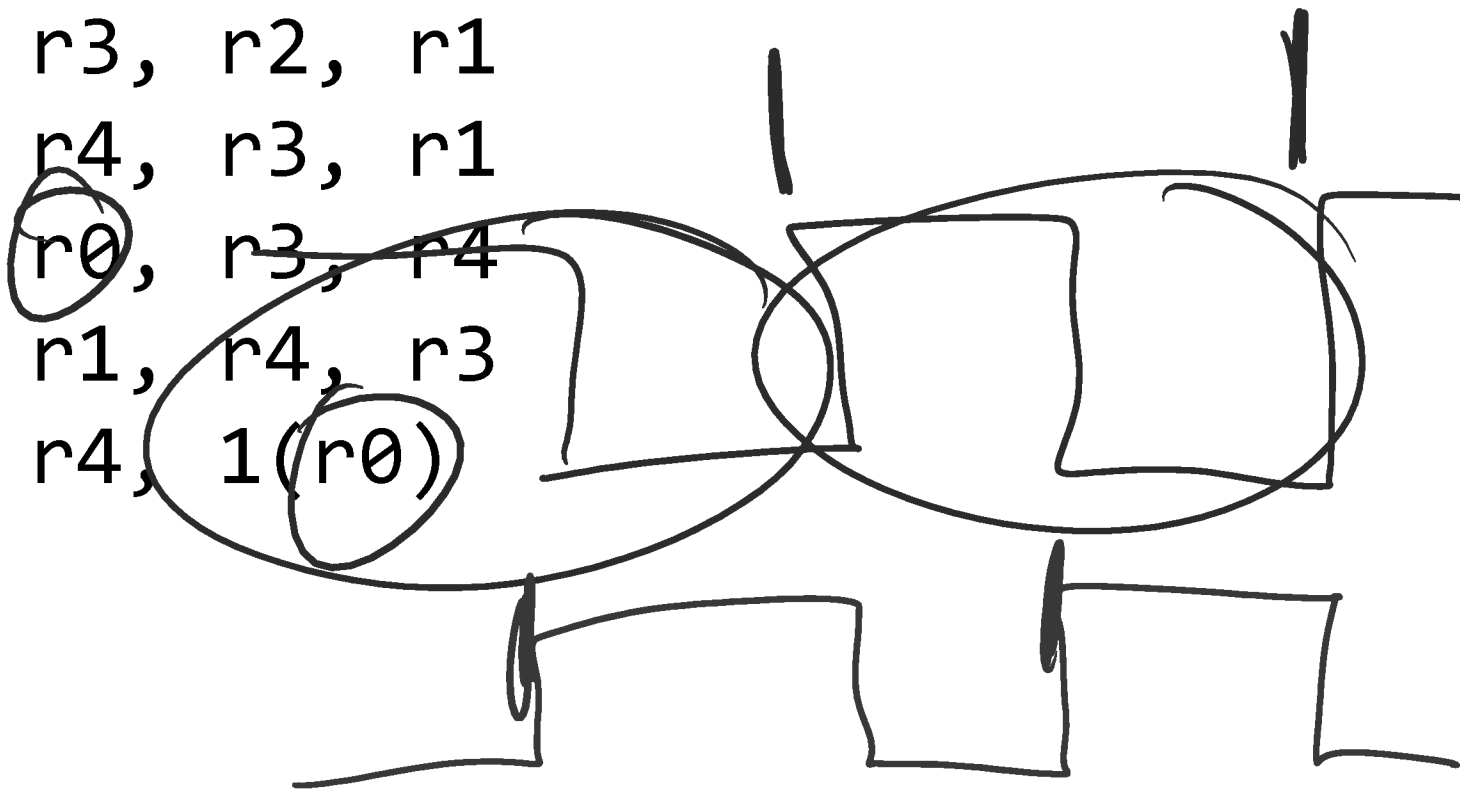
Add a bypass around register file (WB to ID)

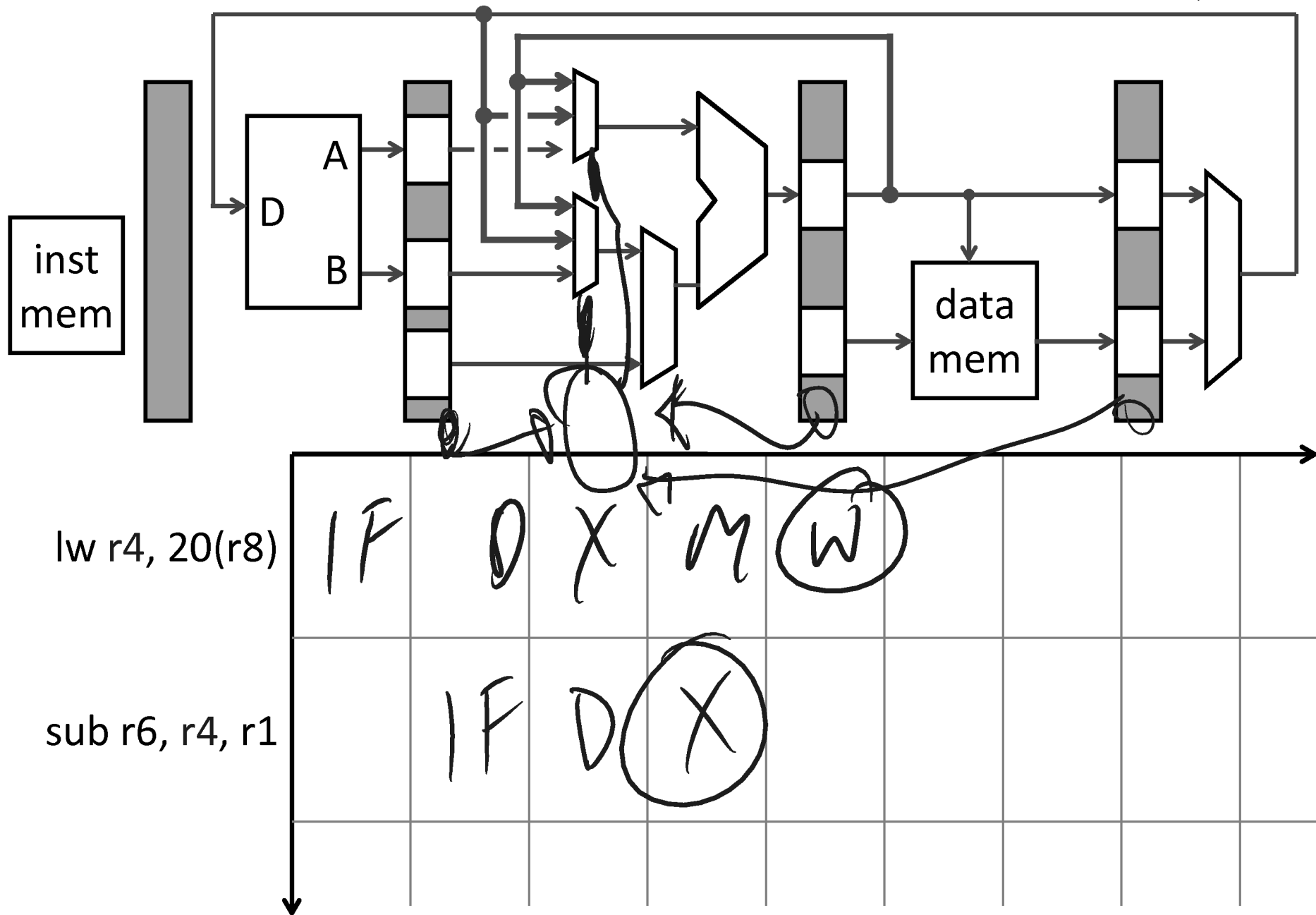
Better: (Hack) just negate register file clock

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

Find all hazards, and say how they are resolved:

add	r3, r1, r2
sub	r3, r2, r1
nand	r4, r3, r1
or	<u>r0</u> , r3, r4
xor	r1, r4, r3
sb	r4, 1( <u>r0</u> )





# 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

LW  
NOP

add    r3, r1, r2  
 nand   r5, r3, r4     $M \rightarrow X$   
 add    r2, r6, r3     $w \rightarrow X$   
 lw    r6, 24(r3)    RF Bypass  
 sw    r6, 12(r2)     $w \rightarrow X$

~~STALL 1 cycle~~  
~~+ w → X~~  
 Delay slot

## Delay Slot(s)

- Modify ISA to match implementation

## Stall

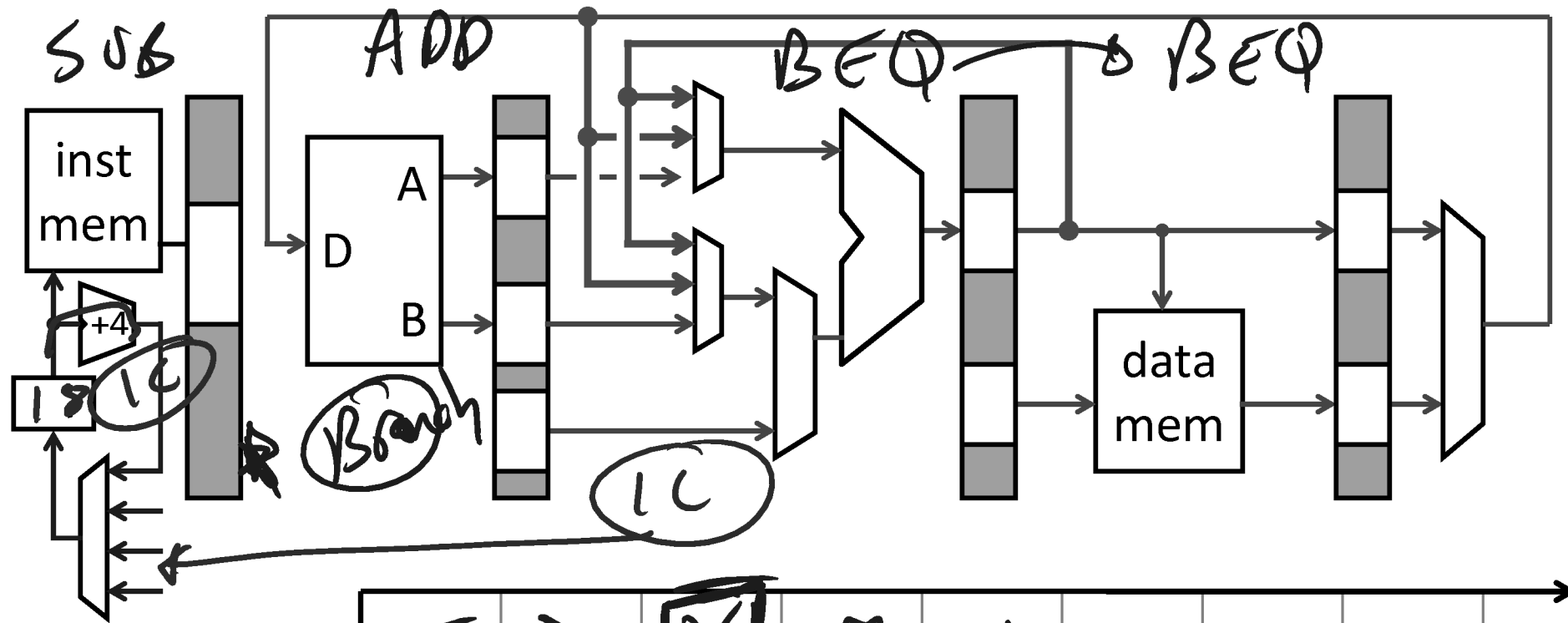
- Pause current and all subsequent instructions

## Forward/Bypass

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

## Tradeoffs?





10 beq r1, r2, L

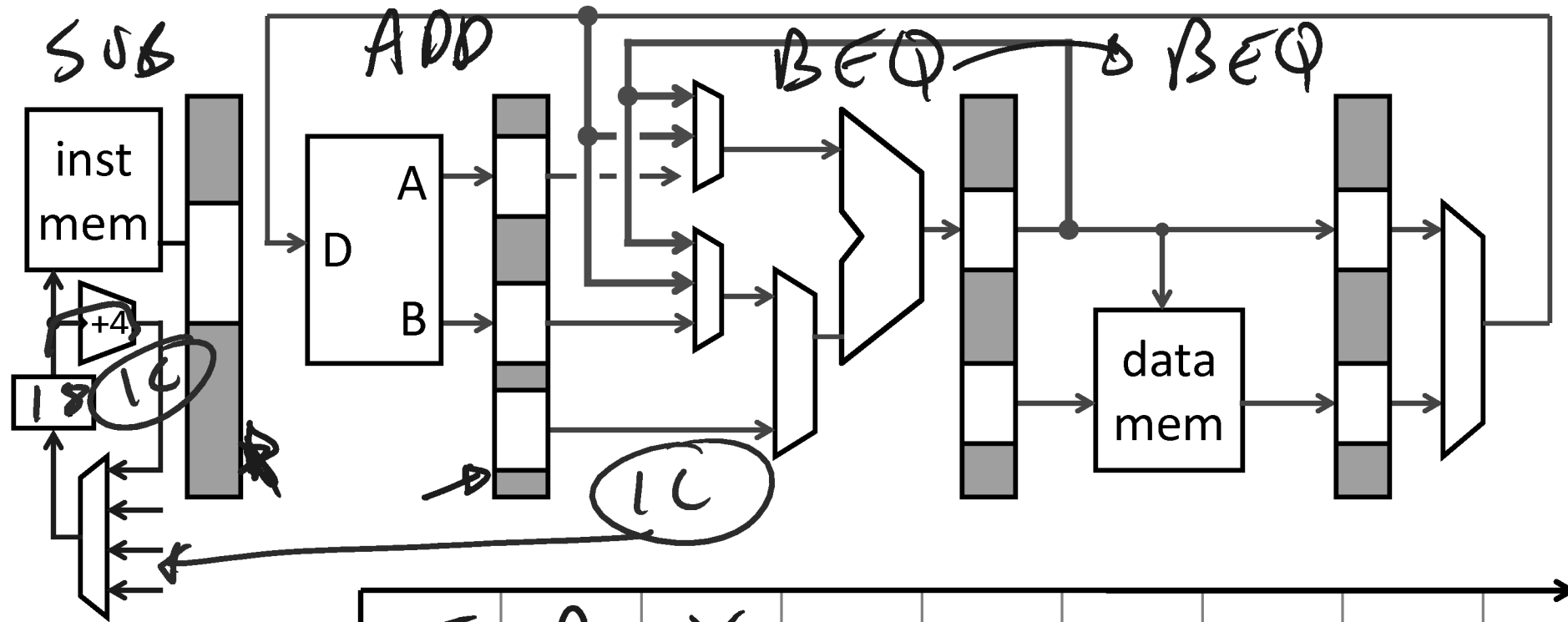
14 add r3, r0, r3

18 sub r5, r4, r6

LC

L: or r3, r2, r4

F	D	<b>X</b>	m	w				
	F	D	X	m	w			
		F	nop					
			F					



10	beq r1, r2, L	F	D	X						
14	add r3, r0, r3		F	F	No					
18	sub r5, r4, r6									
1C	L: or r3, r2, r4				F					

# 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

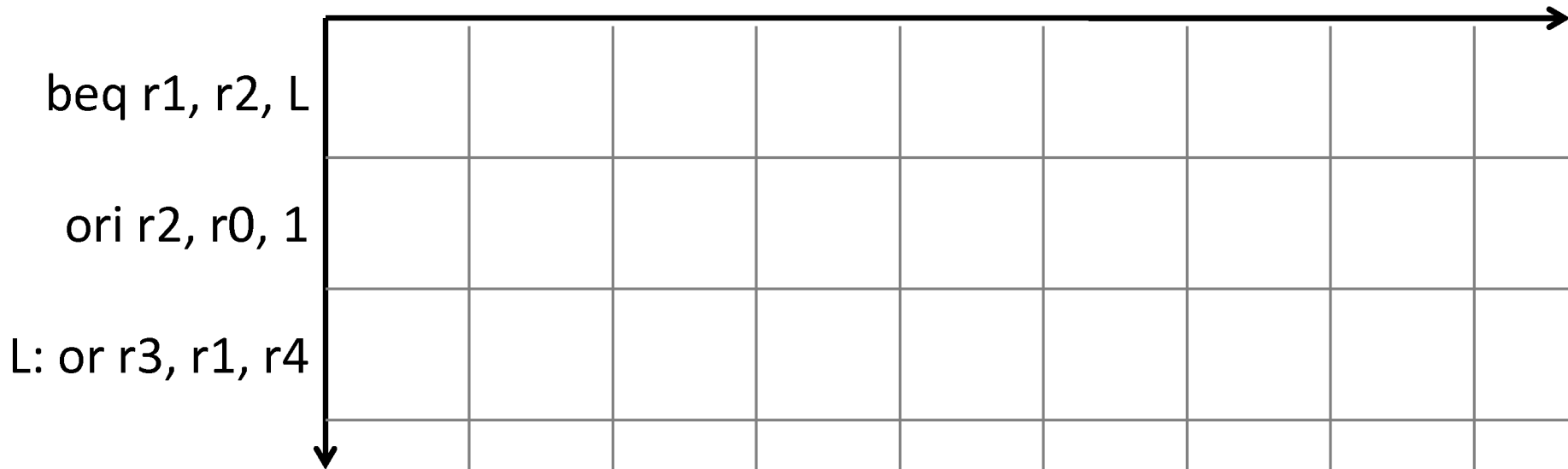
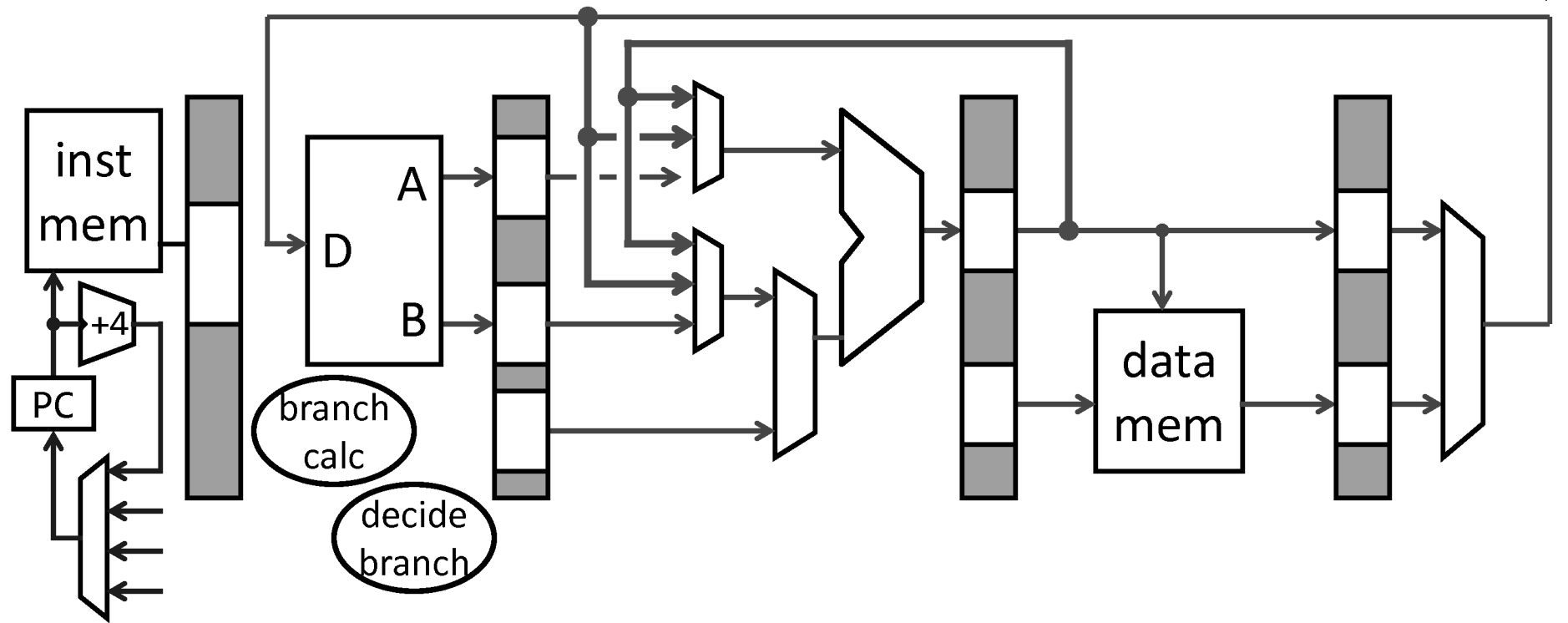
## Delay Slot

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

ADD  
J  
ADD

## 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



# 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

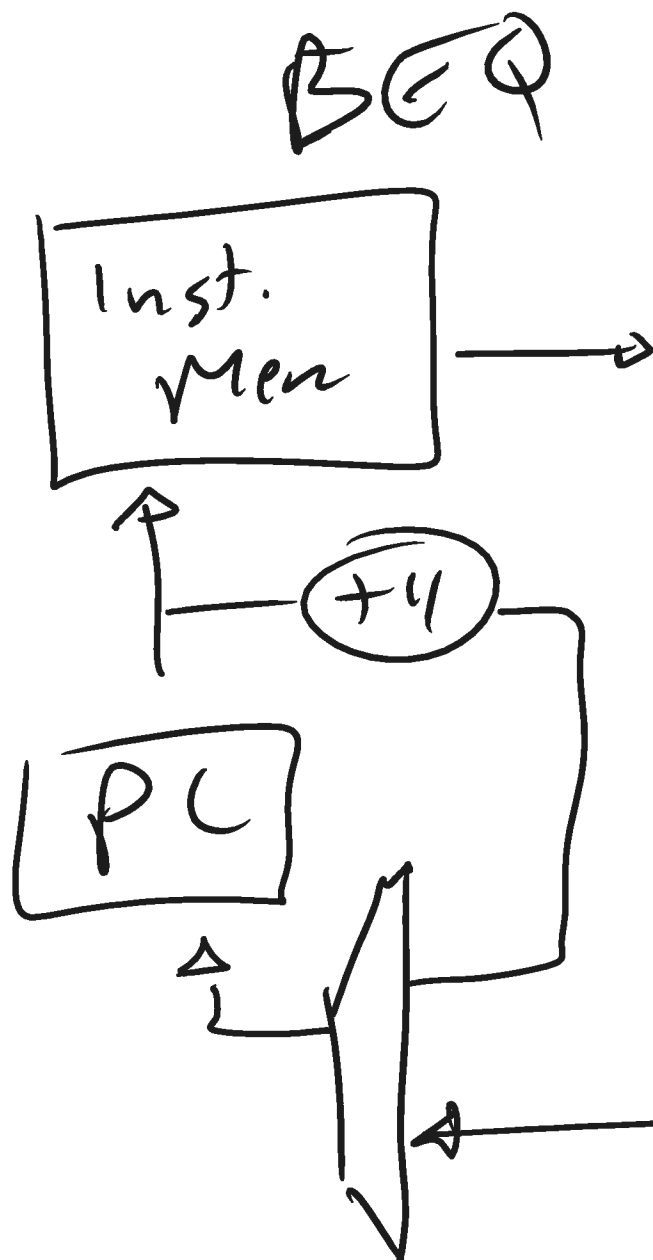
## Delay Slot

## Speculative Execution

Not Taken

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

BEQ  
ADD



while( $r3 \neq 0$ )

---

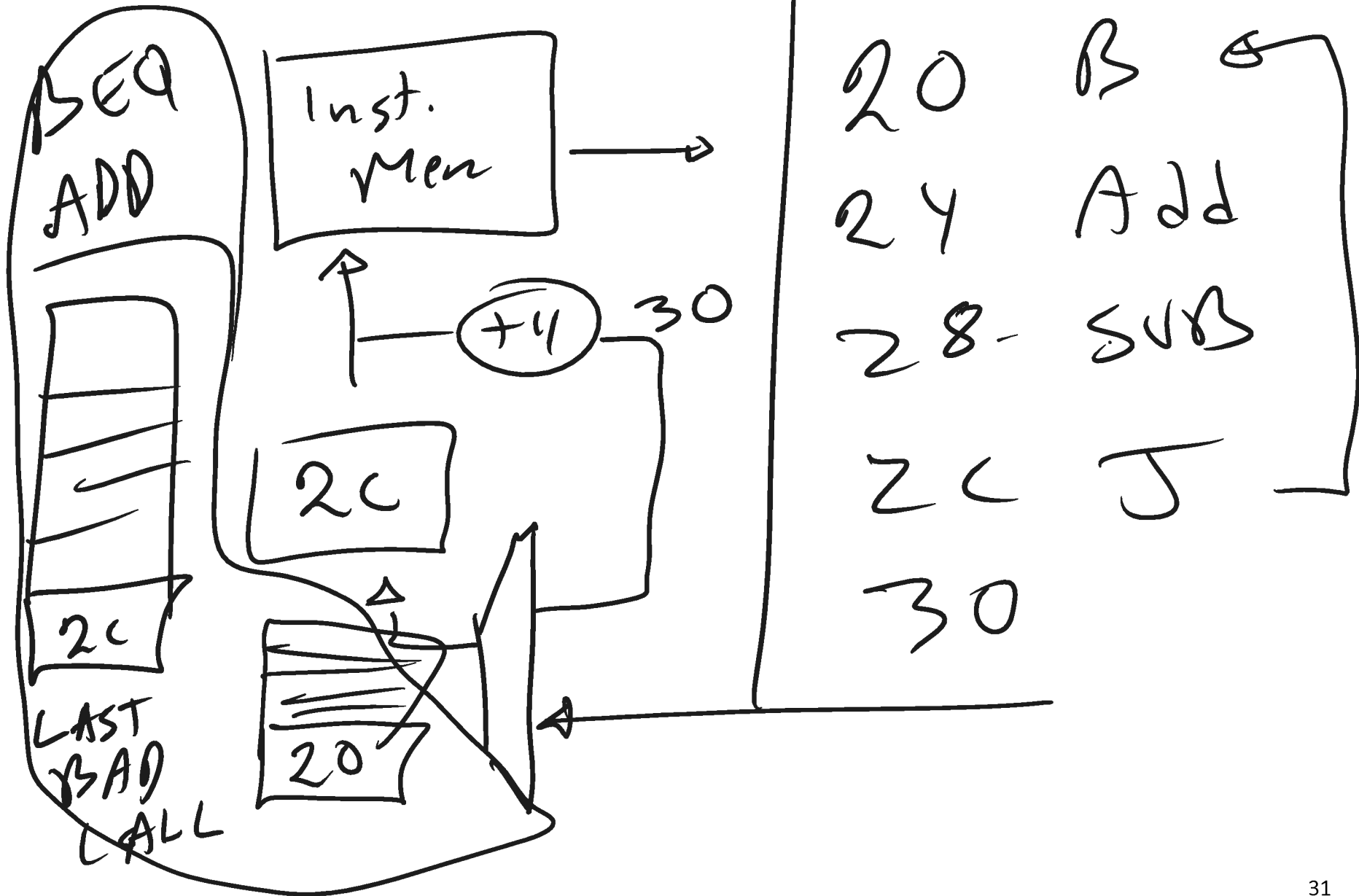
: ~~BEQ~~ END  
           NOP

Top: ~~~~~

~~Top~~

END: BNE Top

dest



## 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