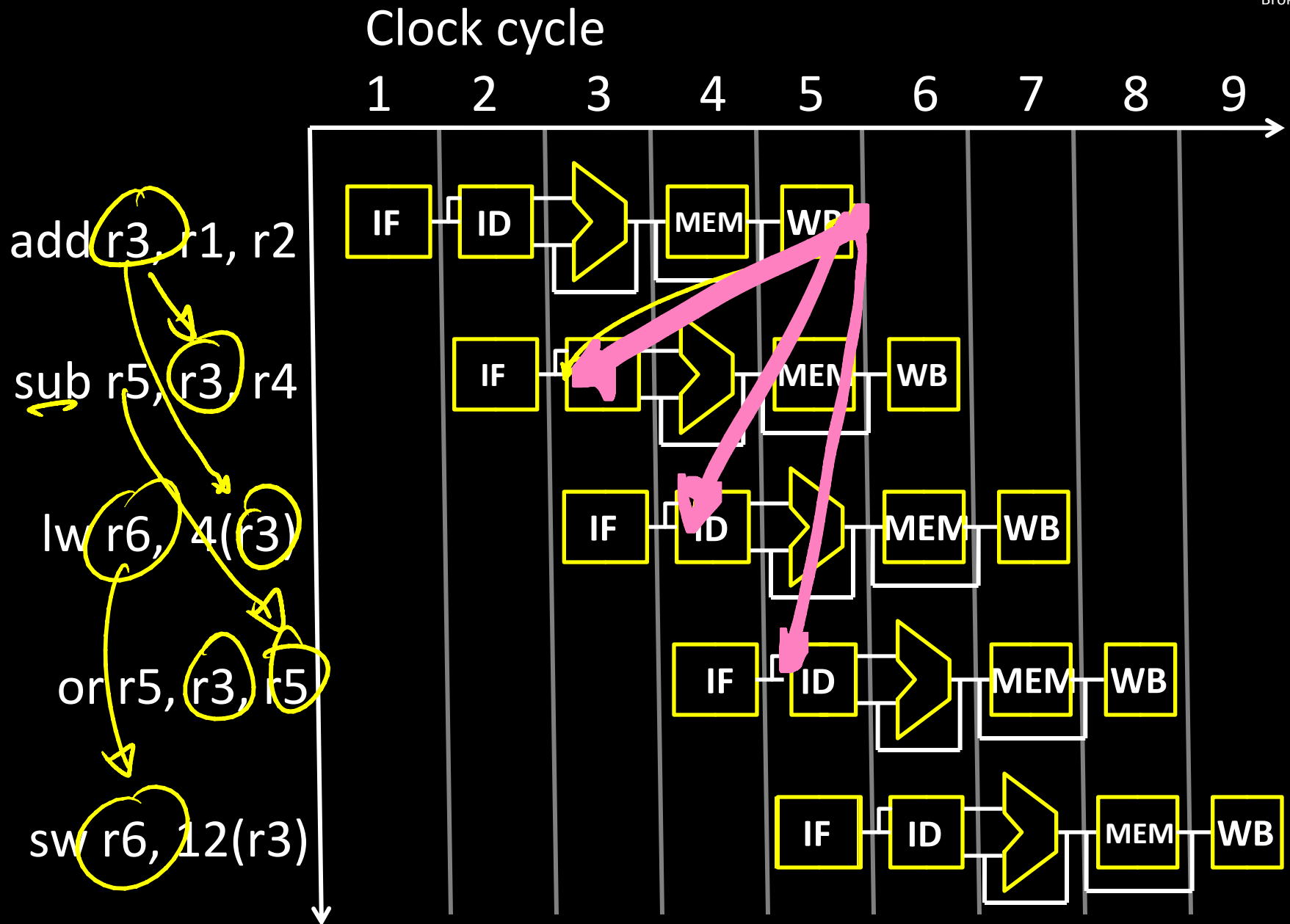# Pipeline Hazards

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H Chapter 4.7
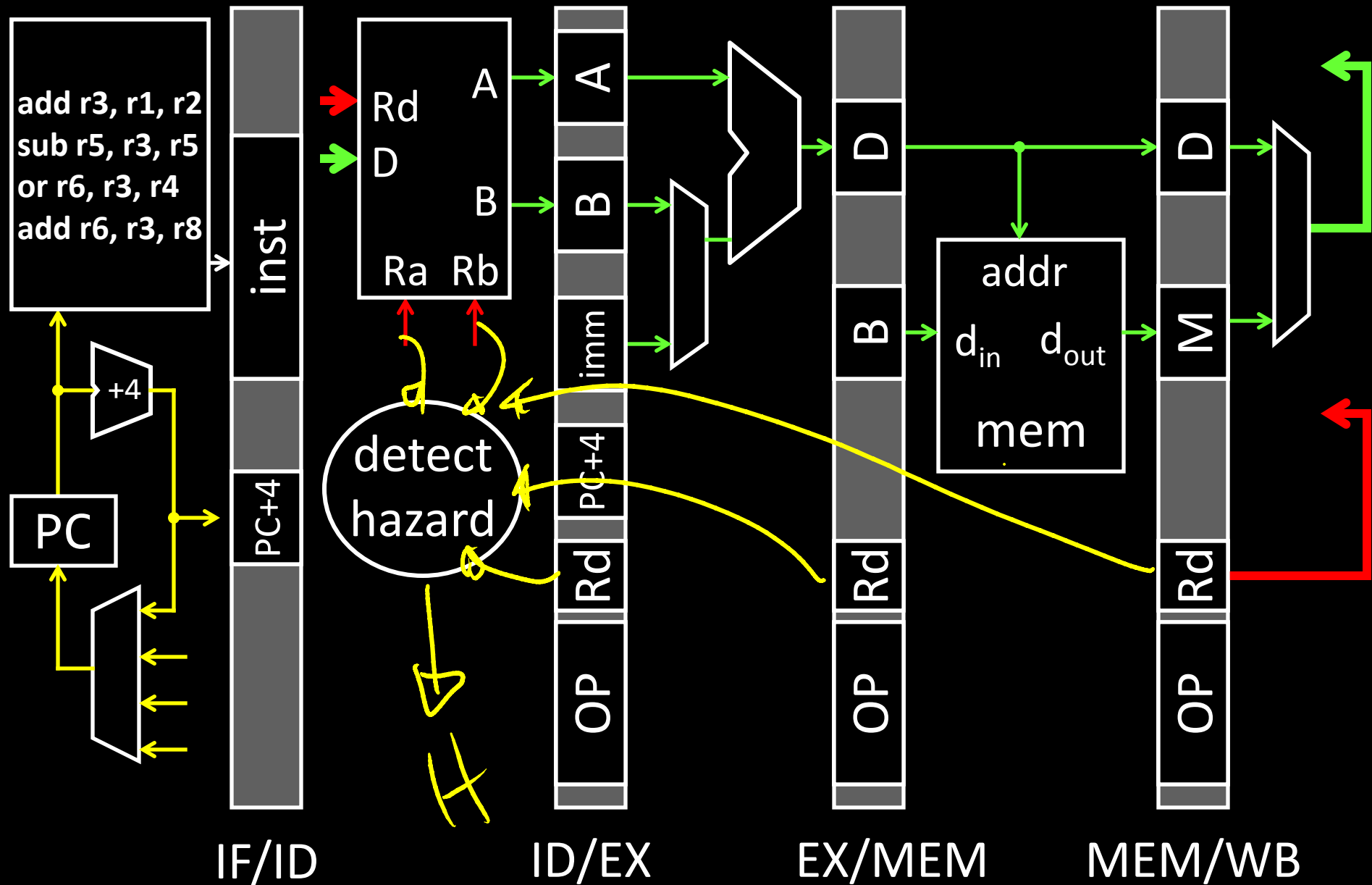
Clock cycle

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

add r3, r1, r2

sub r5, r3, r4

lw r6, 4(r3)

or r5, r3, r5

sw r6, 12(r3)

IF	ID		MEM	WB				
	IF			MEM	WB			
		IF	ID		MEM	WB		
			IF	ID		MEM	WB	
				IF	ID		MEM	WB

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:

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

|| (same for rB)

```
add r3, r1, r2
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8
```

PC

+4

inst

PC+4

Rd

D

A

B

Ra   Rb

detect
hazard

A

B

imm

PC+4

Rd

OP

D

B

Rd

OP

addr

d_in   d_out

mem

D

M

Rd

OP

IF/ID   ID/EX   EX/MEM   MEM/WB

What to do if data hazard detected?

Clock cycle

	1	2	3	4	5	6	7	8
add r3, r1, r2	IF	ID 1,2	EX 3	M	WB			
sub r5, r3, r5		IF	ID	ID	ID	ID	EX	—
or r6, r3, r4			IF	IF	IF	IF	ID	—
add r6, r3, r8							IF	—

6

SUB

NOP

ADD

J

ADD r3, --
J
SUB r4, r3, r0

nop
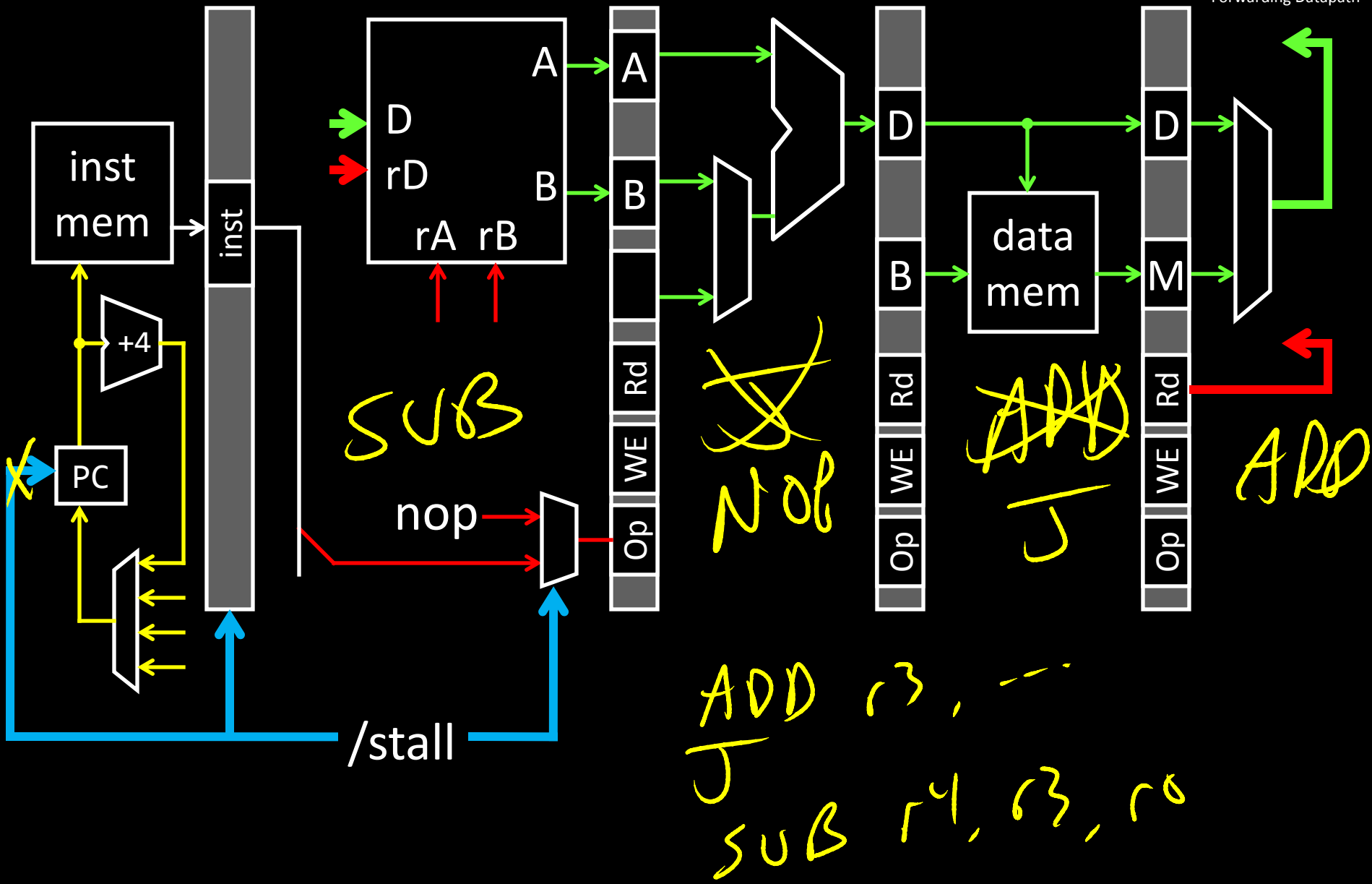
/stall
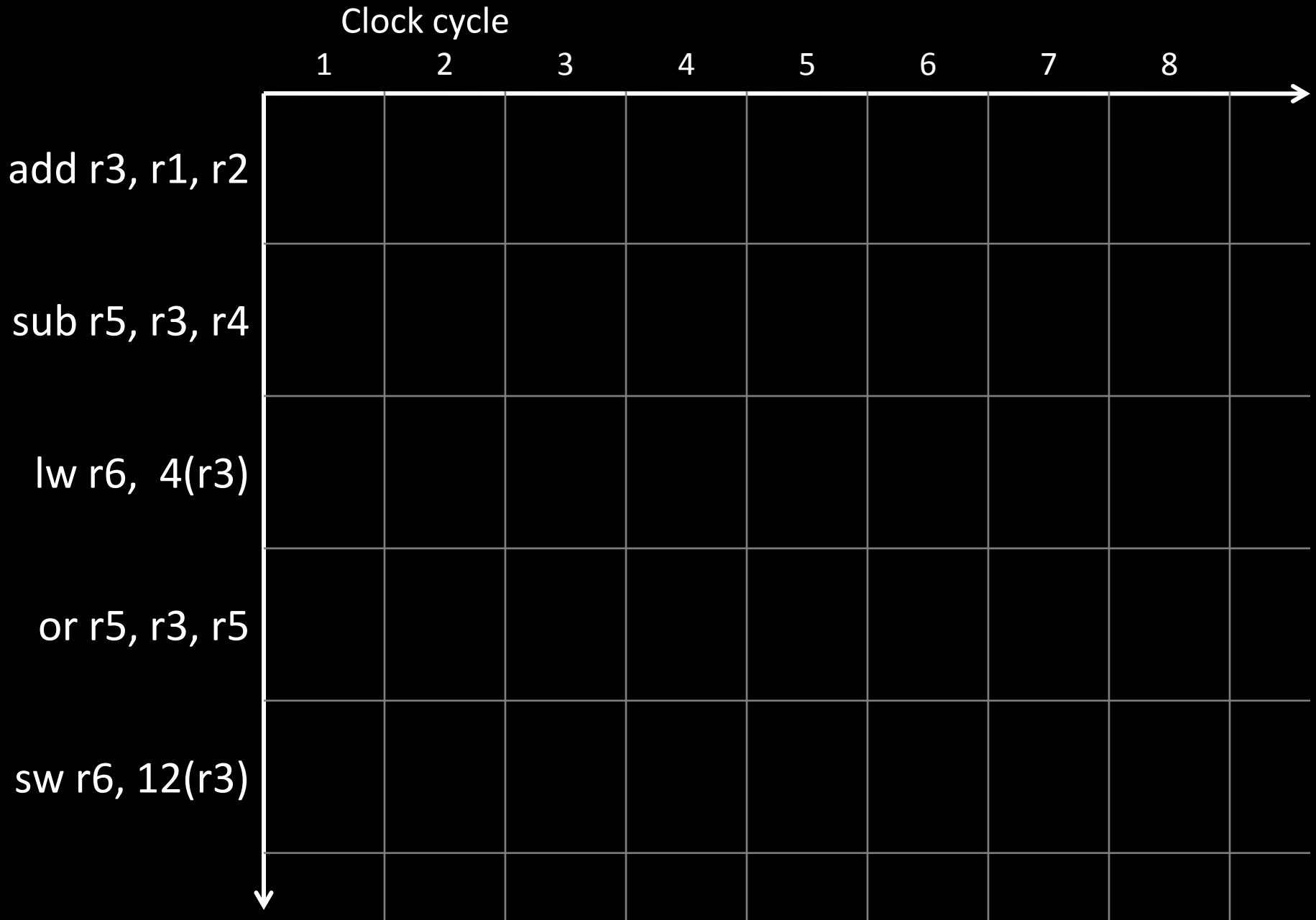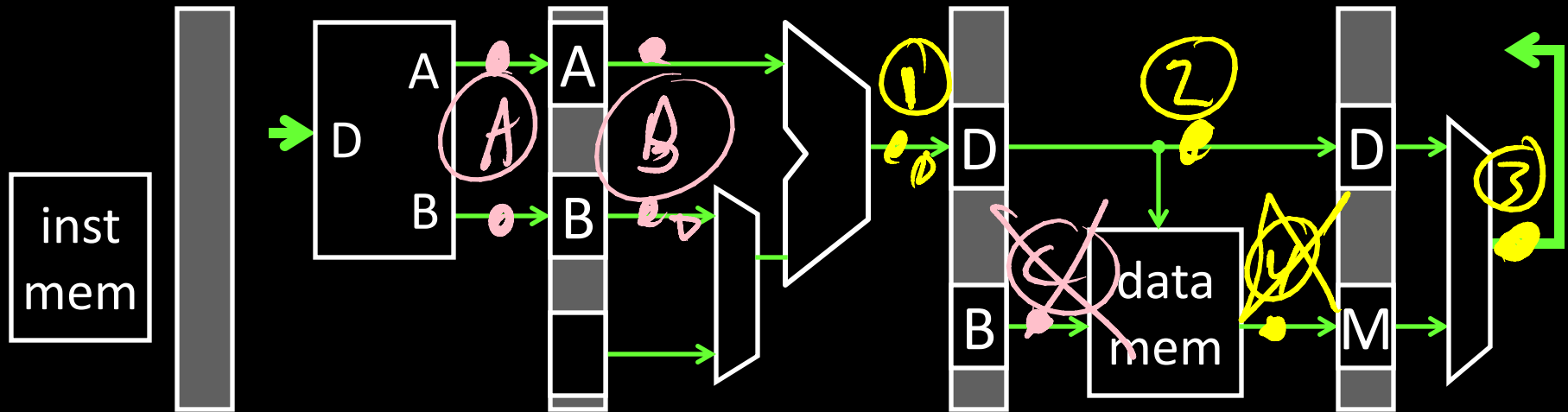
How to stall an instruction in ID stage

- 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

Clock cycle

	1	2	3	4	5	6	7	8

add r3, r1, r2   IF   EX   M

            3

sub r5, r3, r5   IF   ID

or r6, r3, r4   IF   ID

add r6, r3, r8   IF   ID

9

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)								
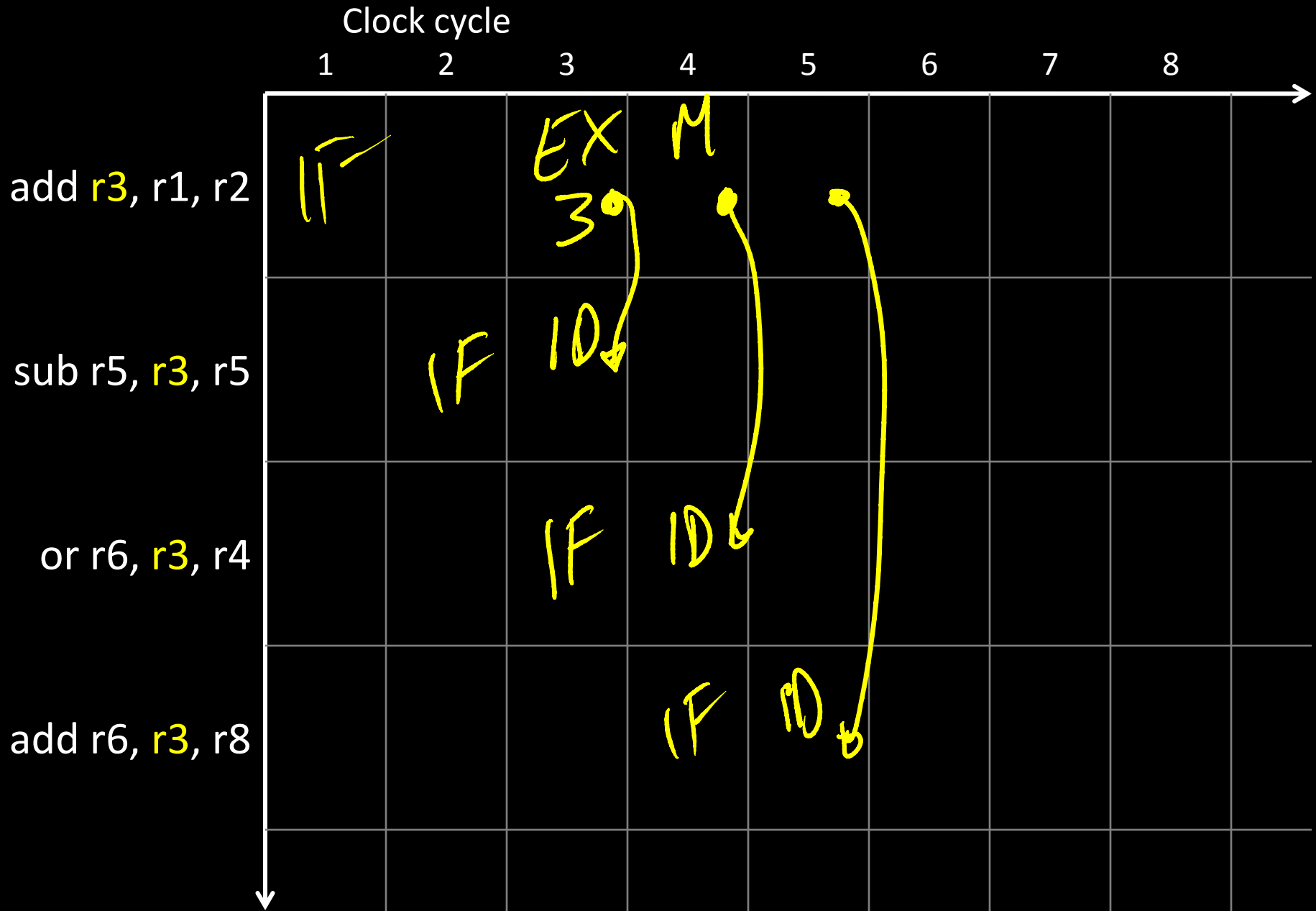
Forward correct value from?   to?

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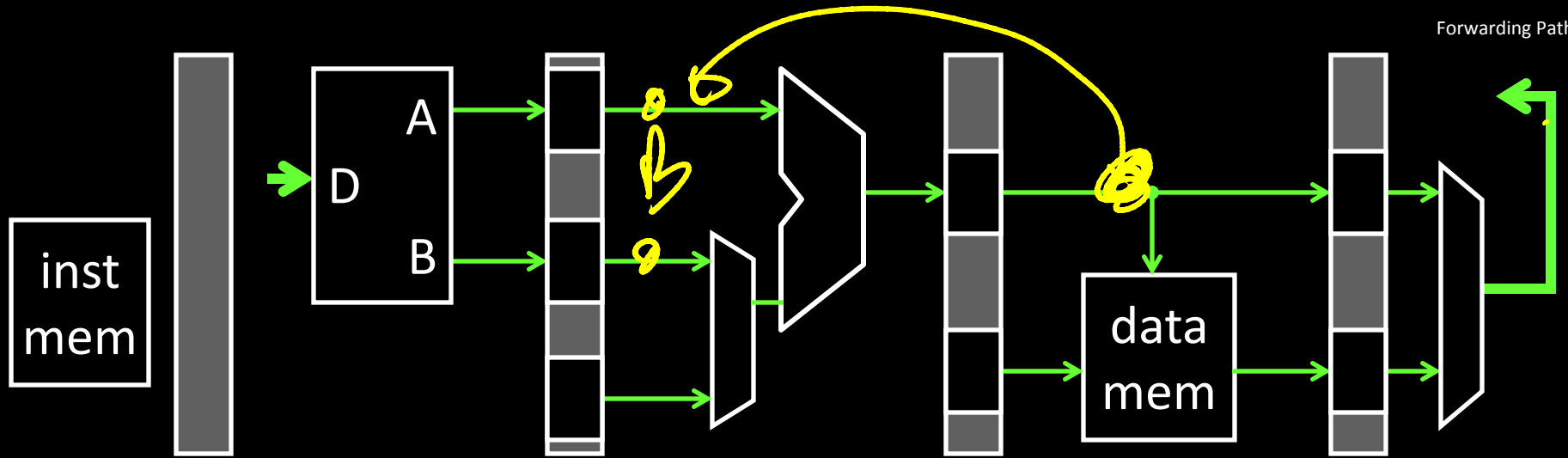c) MEM, just after EX/MEM.B is read: on critical path

WB to EX Bypass

- EX needs value being written by WB

Resolve:

Add bypass from WB final value to start of EX

Detect:

LW
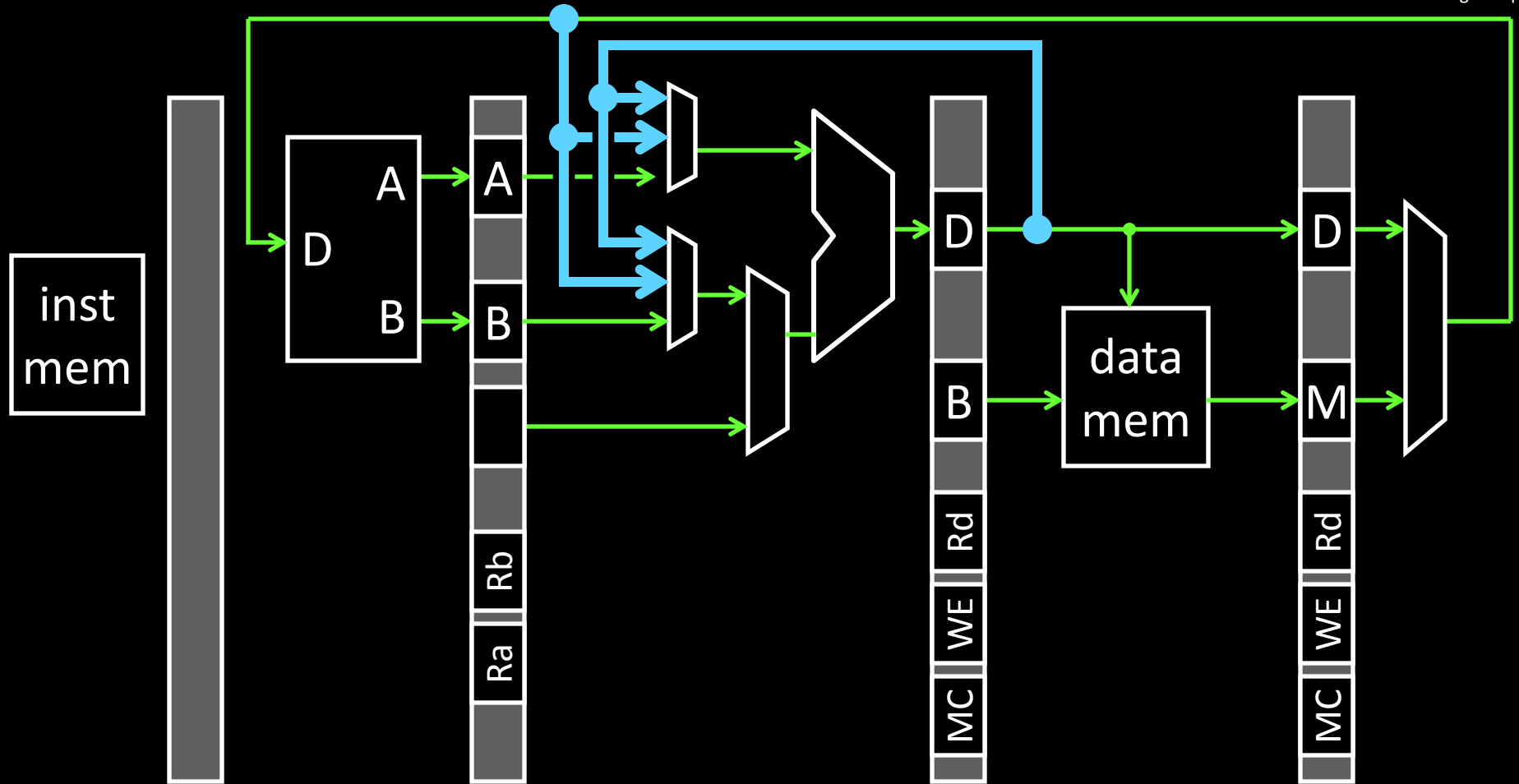
add r4, r1, r2 — IF ID EX M WB

sub r6, r4, r1 — IF ID EX

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:
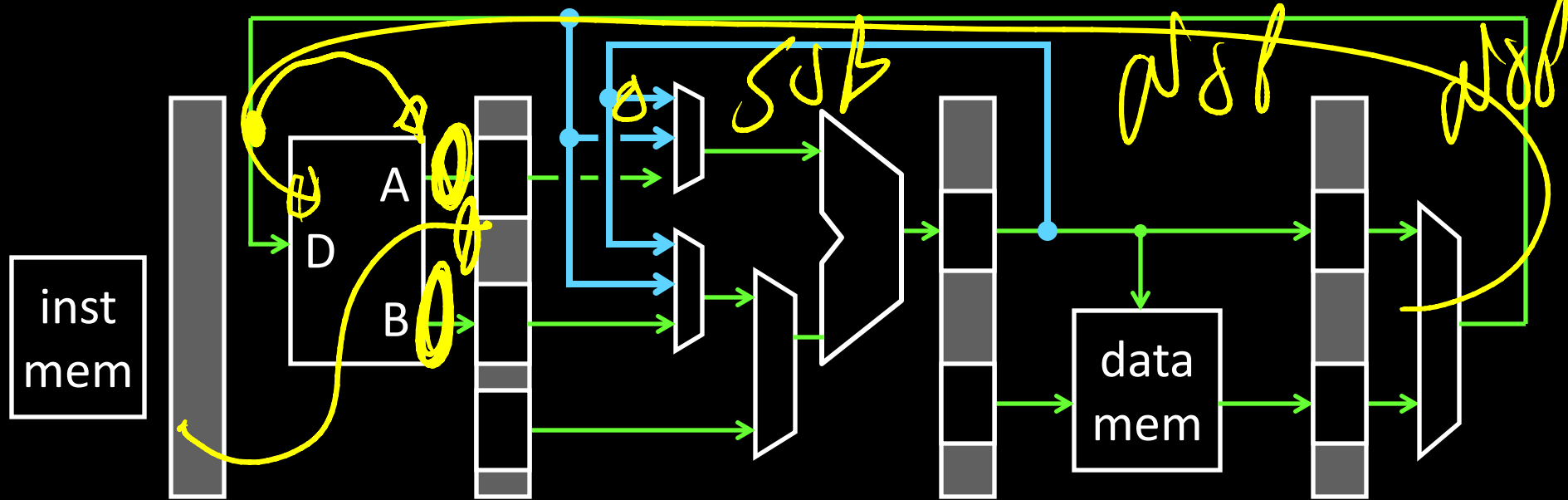
inst mem

D

A
B

A
B

Ra Rb

MC WE Rd

D
B

data mem

MC WE Rd

D
M

OR

SUB

7

ADD

3
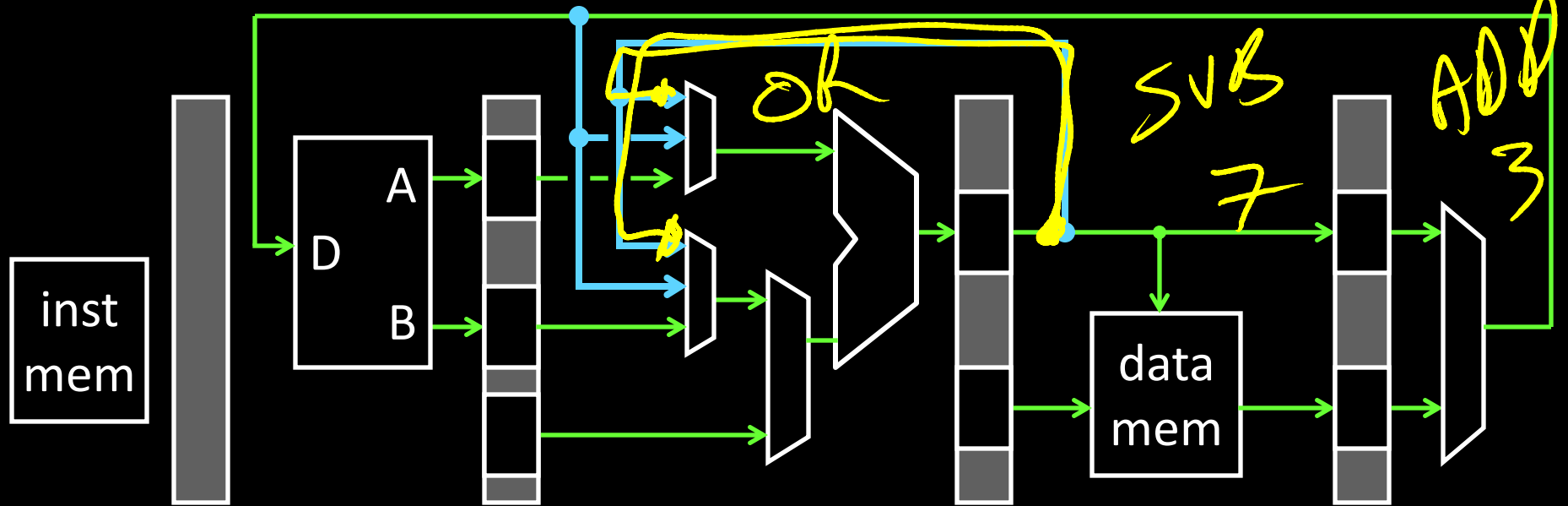
add r1, r1, r2

SUB r1, r1, r3

OR r1, r4, r1

IF	ID	EX	M	WB			

add r4, r1, r2	IF	D	X	M	WB			
nop		IF						
nop			IF					
sub r6, r4, r1				IF	ID	EX		

Register File Bypass

- Reading a value that is currently being written

Detect:

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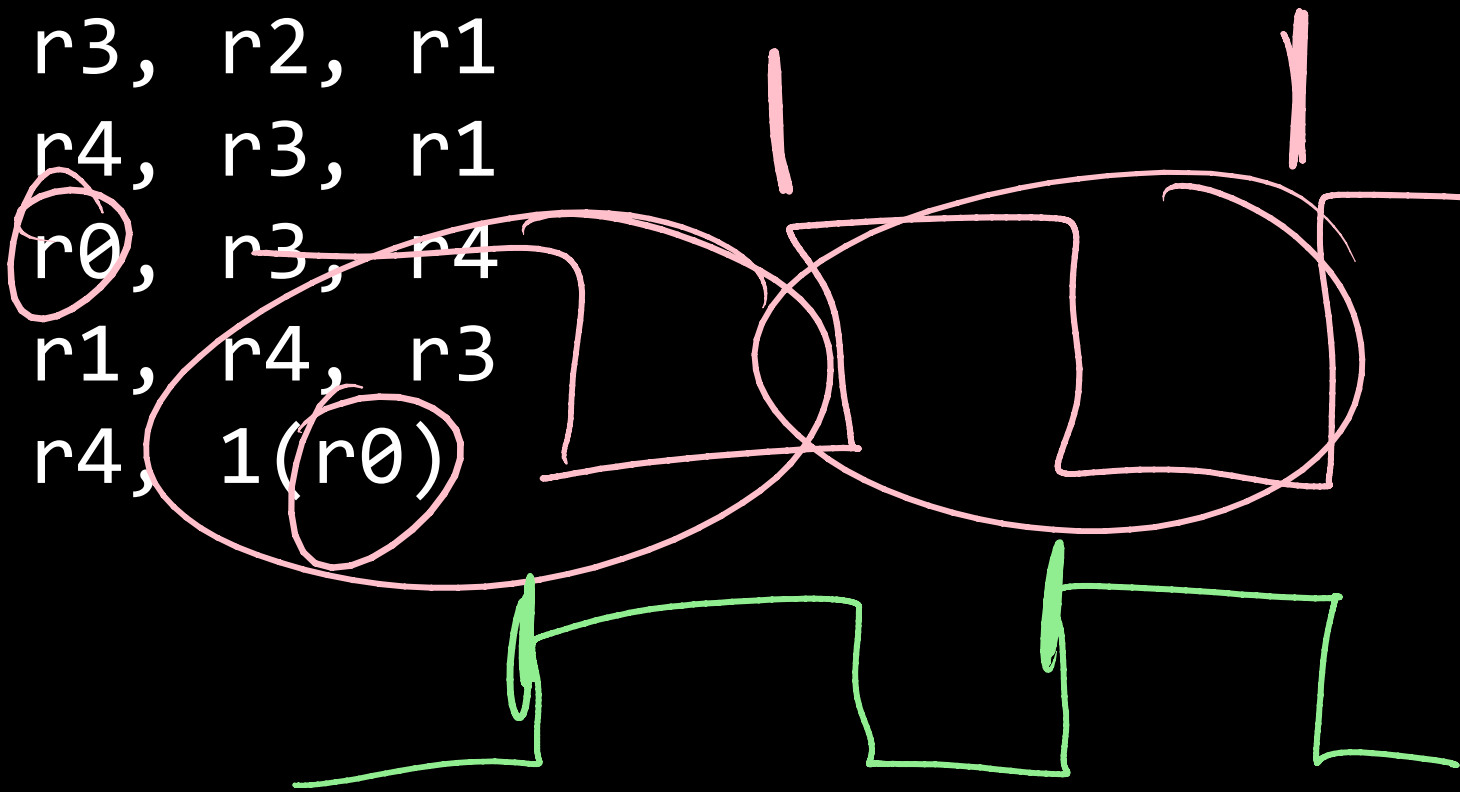   and (WB is writing a register)
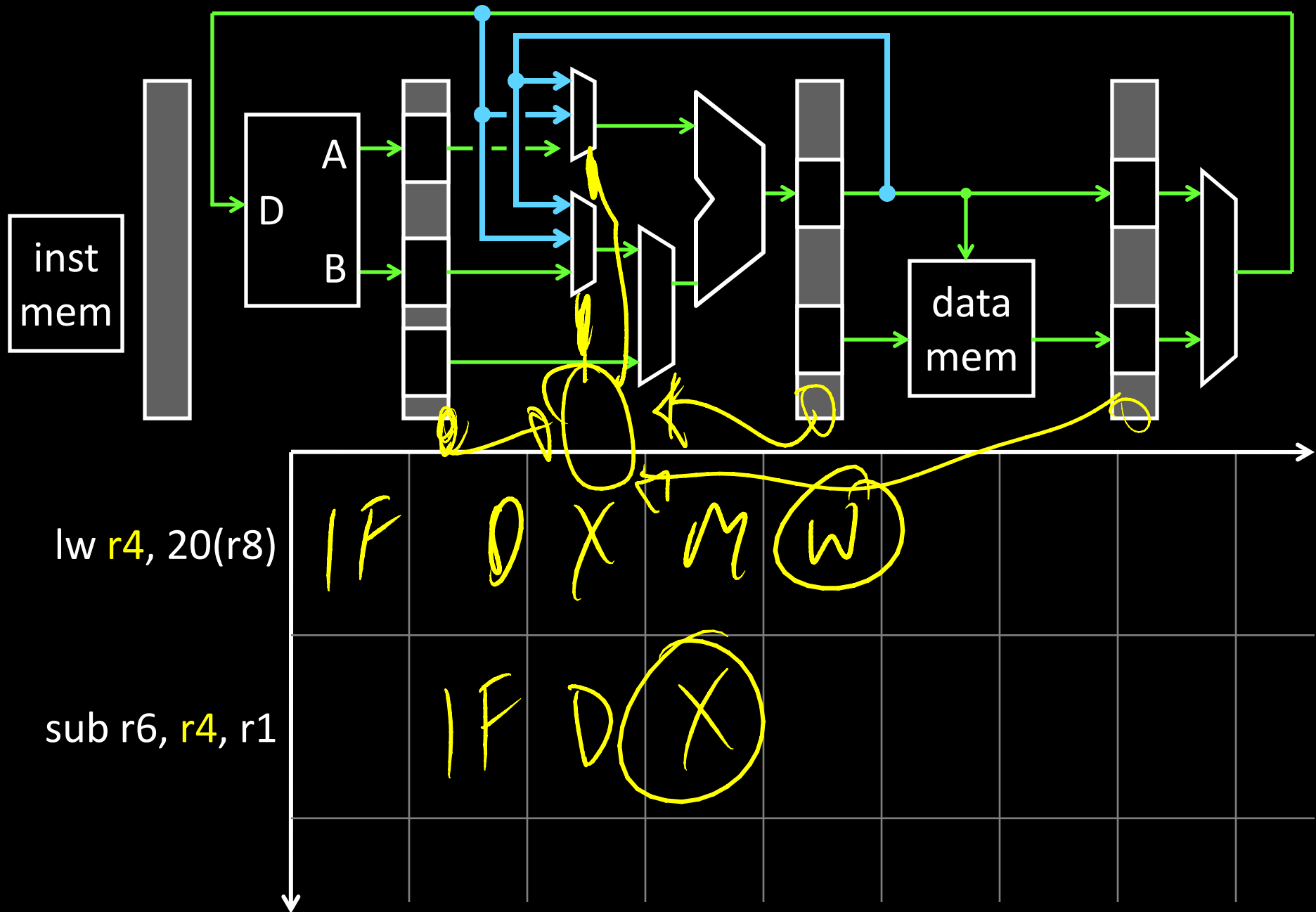
Resolve:

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

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)
```

lw r4, 20(r8)

sub r6, r4, r1

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

```
add     r3, r1, r2
nand    r5, r3, r4        M → X
add     r2, r6, r3        W → X
lw      r6, 24(r3)        RF Bypass
sw      r6, 12(r2)        W → X
```

STALL 1 cycle

+ W → X

Delay Slot

Delay Slot(s)

- Modify ISA to match implementation

Stall

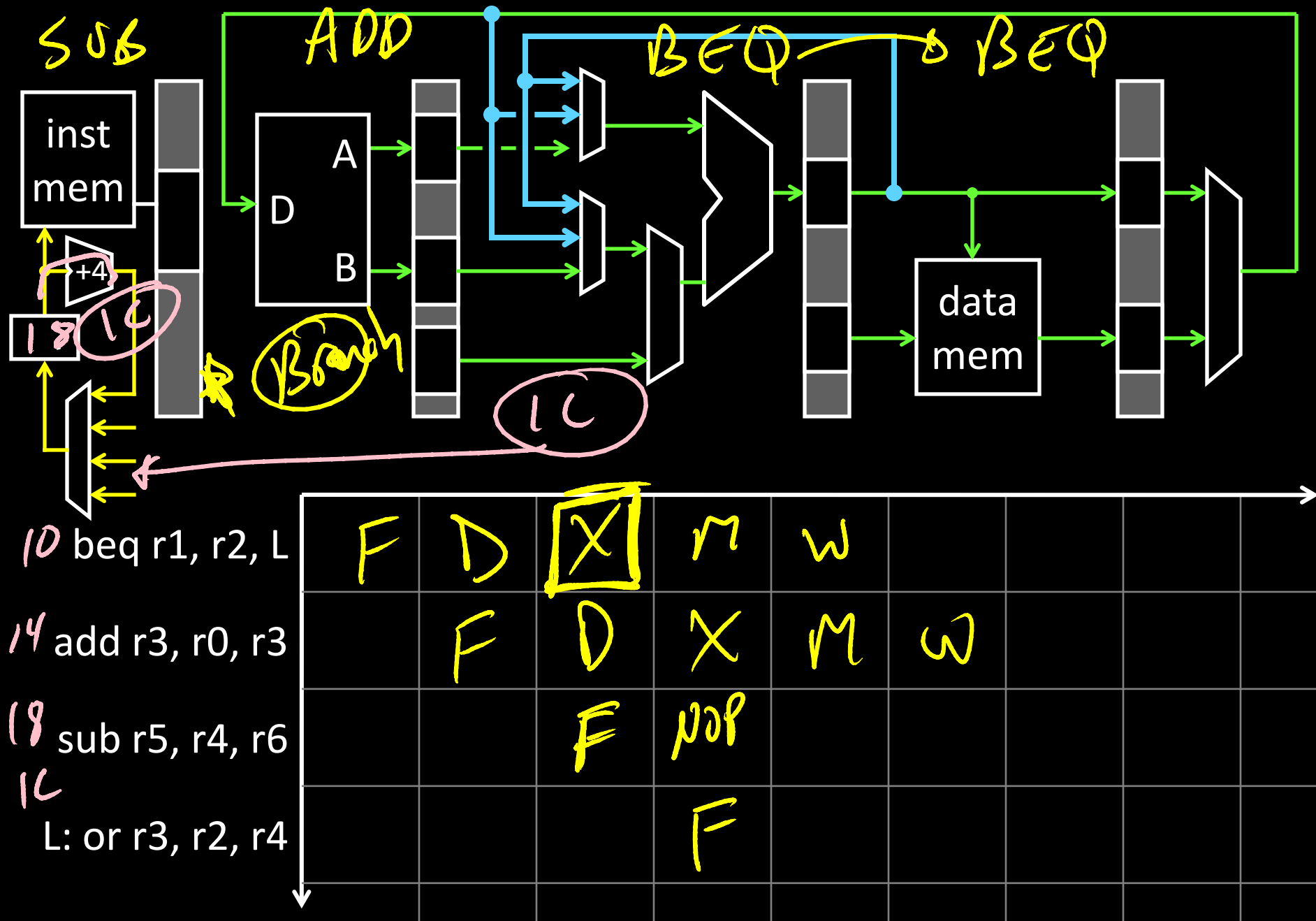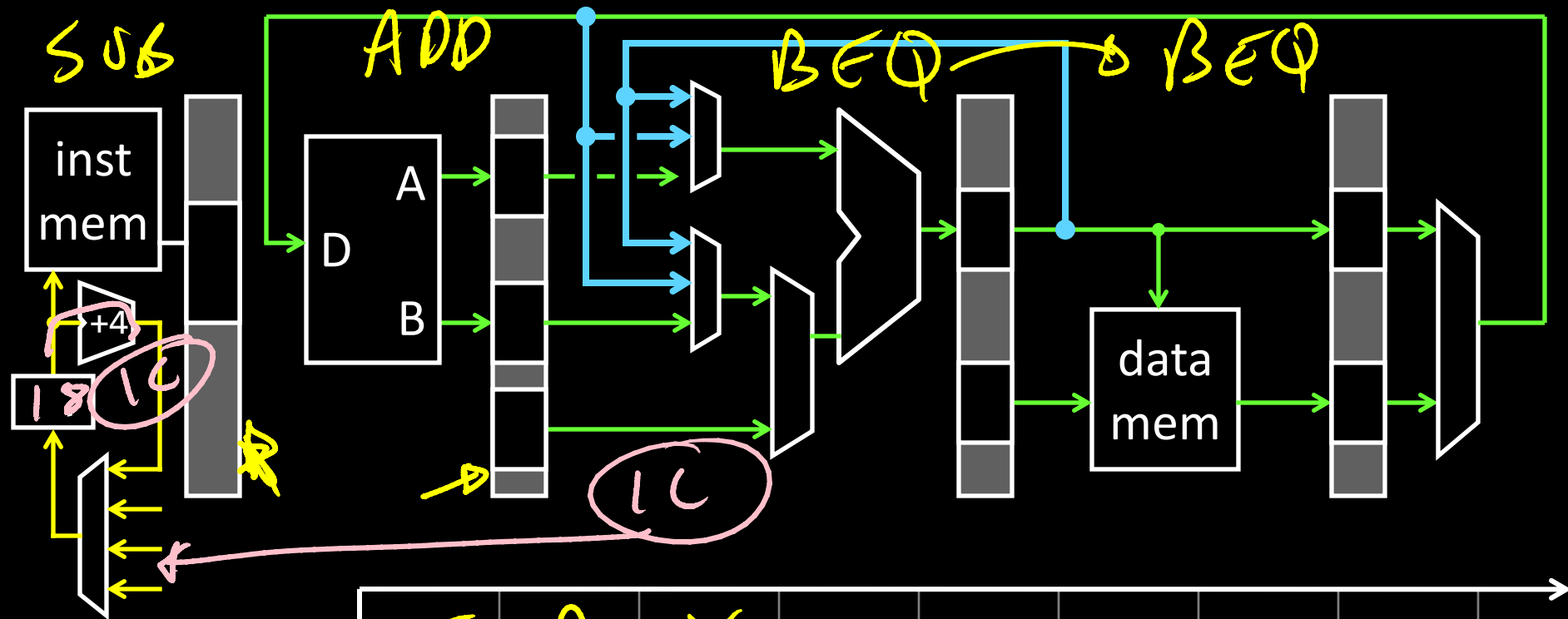- Pause current and all subsequent instructions

Forward/Bypass

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

Tradeoffs?

SUB   ADD   BEQ   BEQ

Branch

IC

10 beq r1, r2, L	F	D	X	M	W			
14 add r3, r0, r3		F	D	X	M	W		
18 sub r5, r4, r6			F	NOP				
1C L: or r3, r2, r4				F				

SUB   ADD   BEQ   BEQ

inst mem

+4

D   A   B

data mem

10 beq r1, r2, L

14 add r3, r0, r3

18 sub r5, r4, r6

1C

L: or r3, r2, r4

F	D	X				
	F	F	Nop			
			F			

Control Hazards

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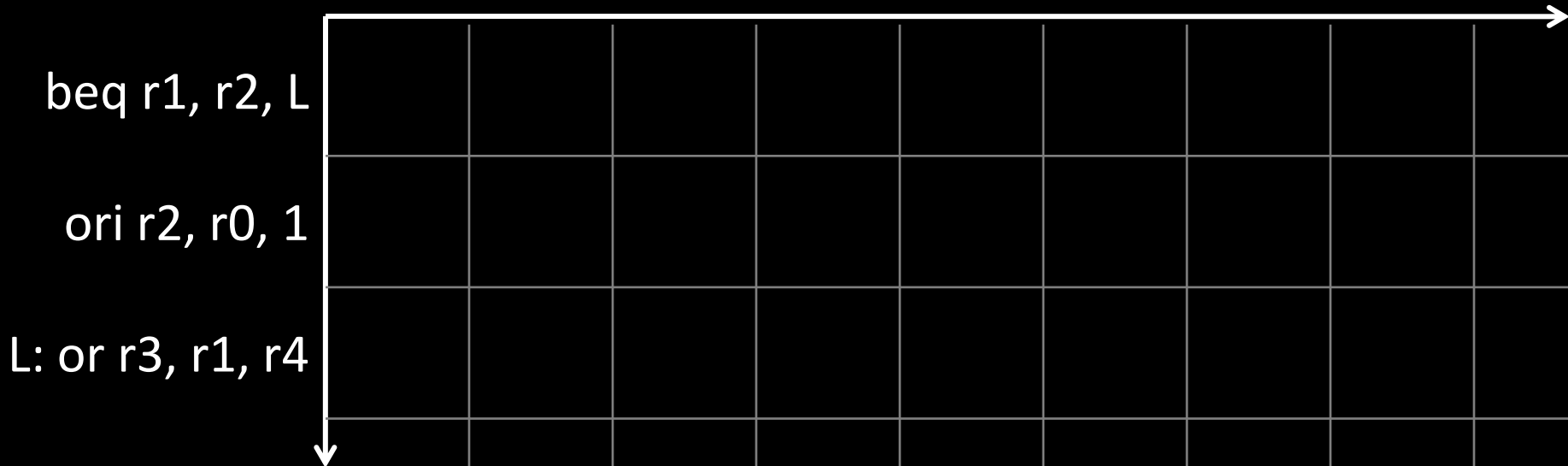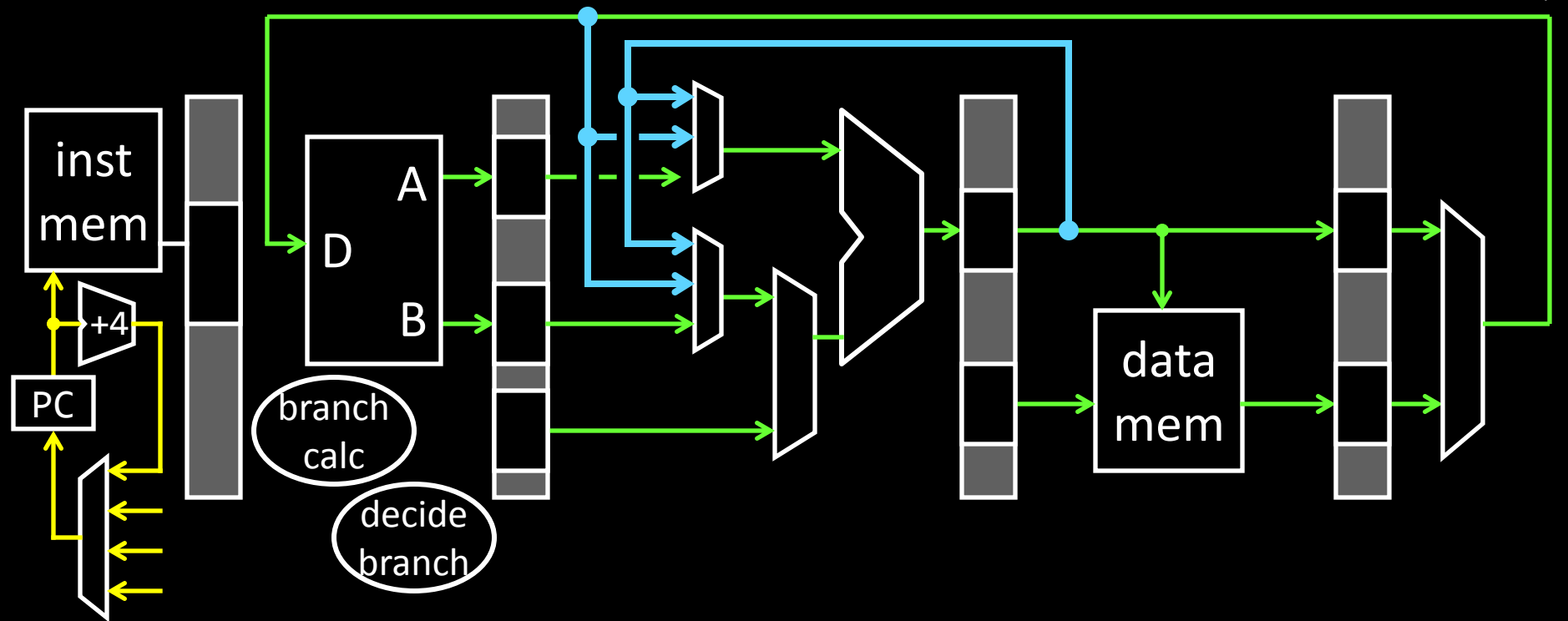- i.e. next PC is not known until 2 cycles after branch/jump

Delay Slot

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

A D D

J

A D D

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

beq r1, r2, L

ori r2, r0, 1

L: or r3, r1, r4

Control Hazards

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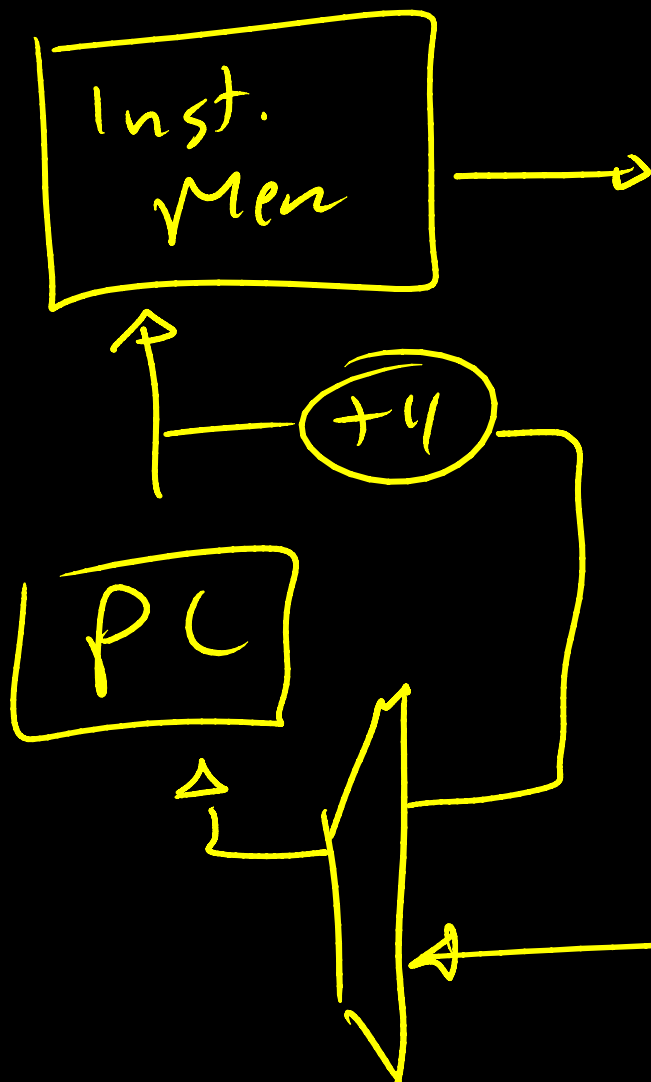- i.e. next PC not known until 2 cycles after branch/jump

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

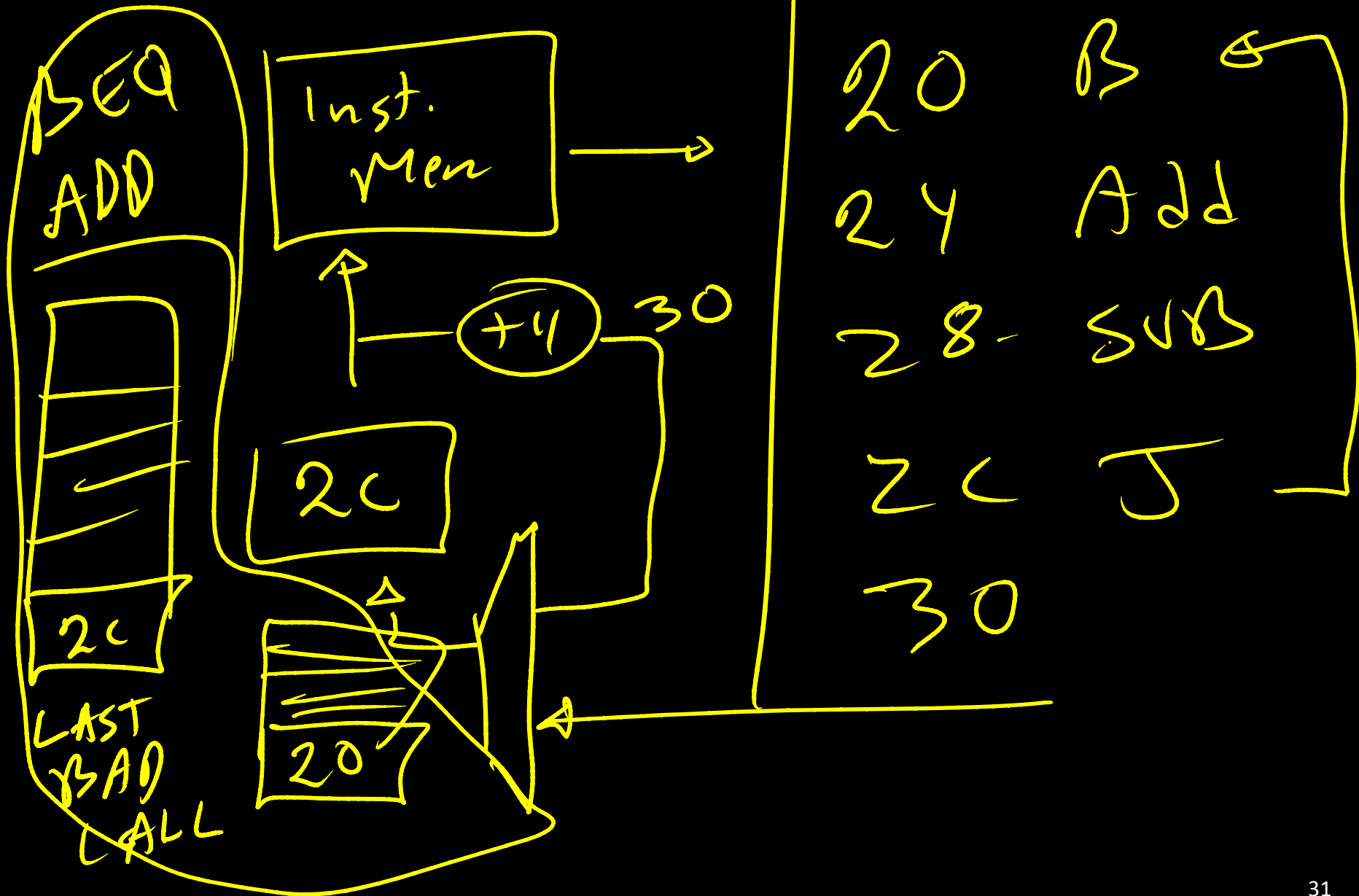BEQ
ADD

Inst.
Mem

+4

PC

while(r3≠0)

J

: BEQ END
NOP
T8P:
~~~
~~~

J T8P

END: BNE T8P

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