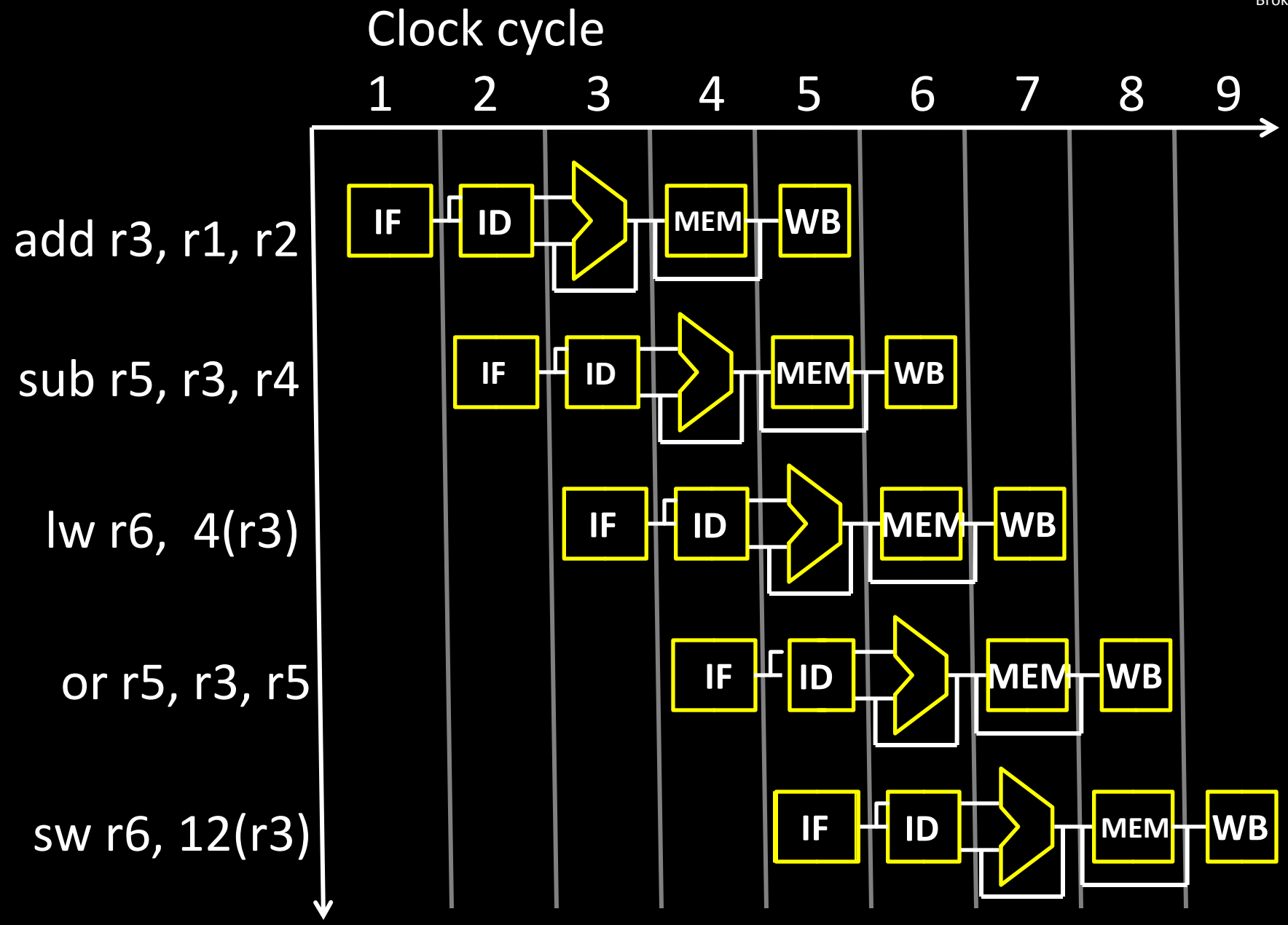
Pipeline Hazards

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H Chapter 4.7

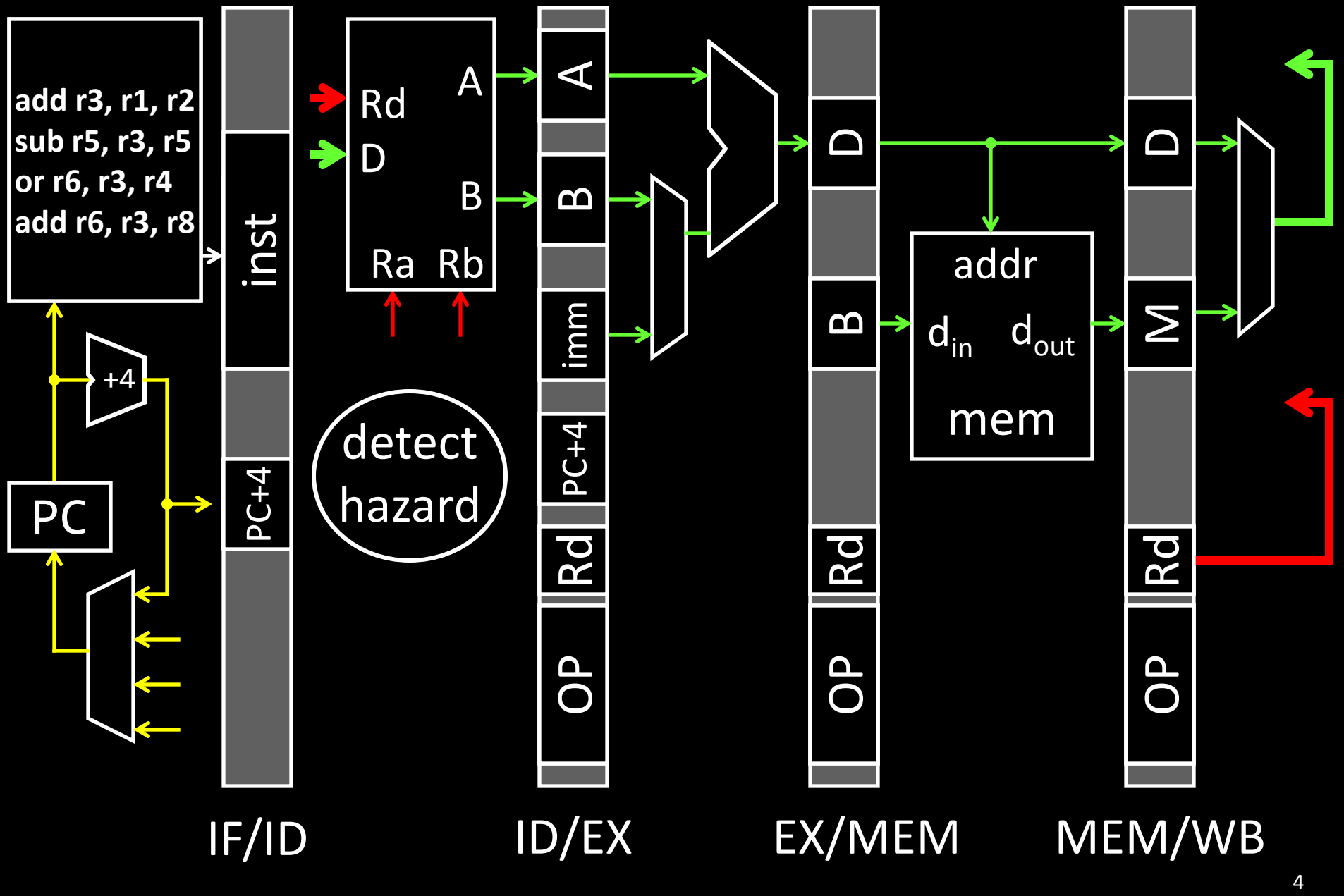

Clock cycle
1
2
3
4
5
6
7
8
9
add r3, r1, r2
sub r5, r3, r4
lw r6, 4(r3)
or r5, r3, r5
sw r6, 12(r3)
IF
ID
MEM
WB

Data Hazards

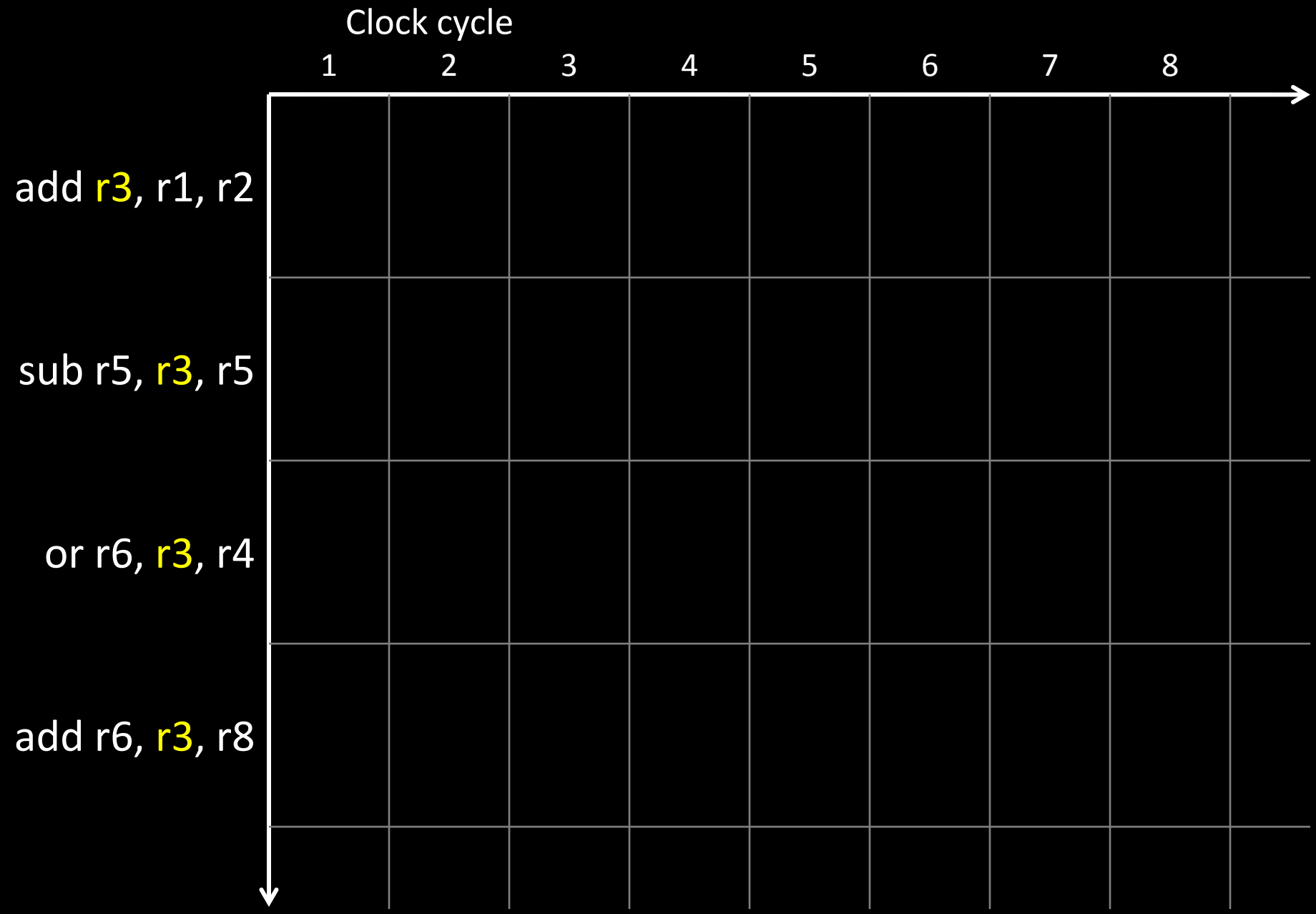
- register file reads occur in stage 2 (IF)
- register file writes occur in stage 5 (WB)
- next instructions may read values about to be written

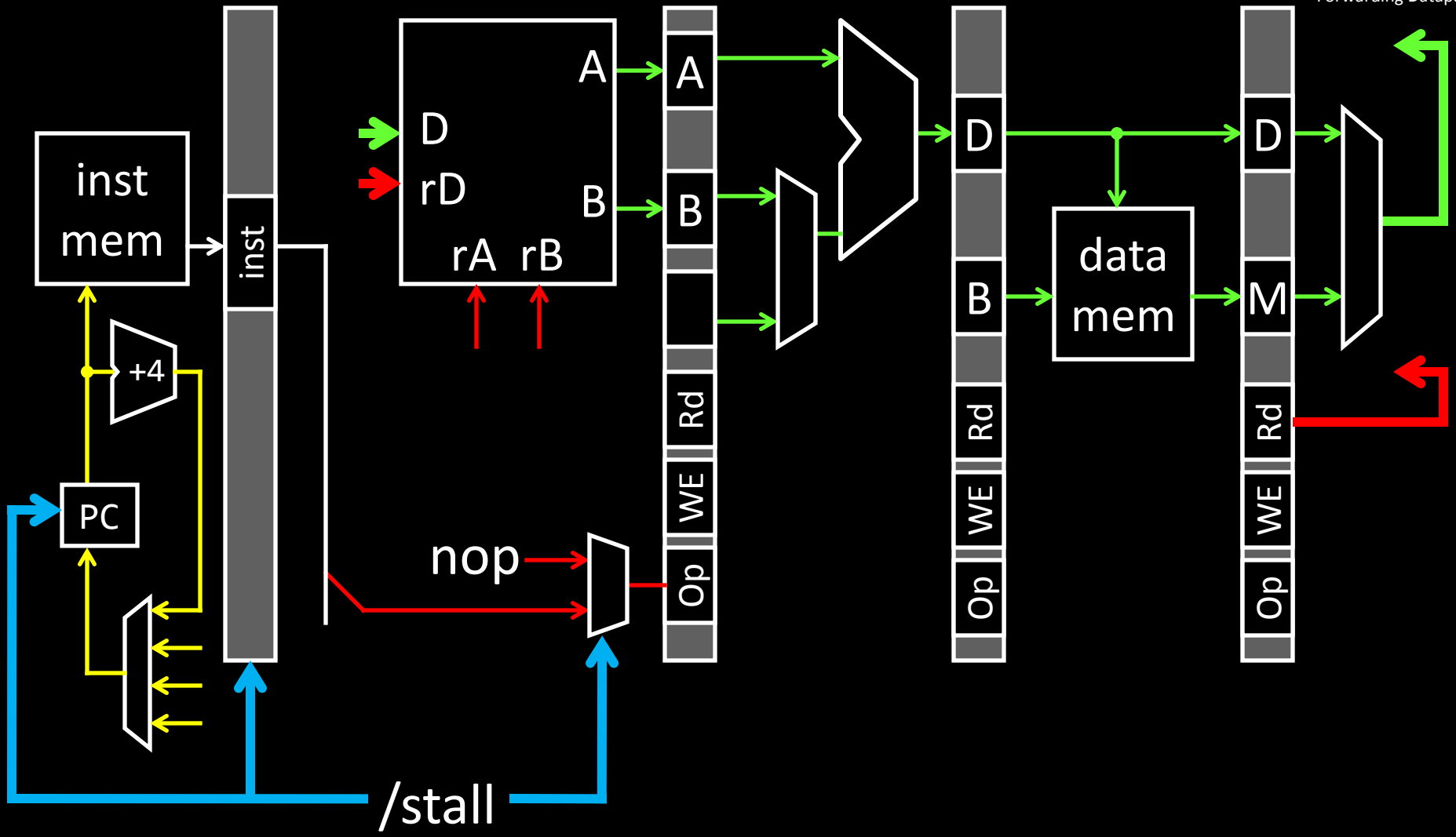
How to detect? Logic in ID stage:

```
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
inst
PC+4
+4
PC
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

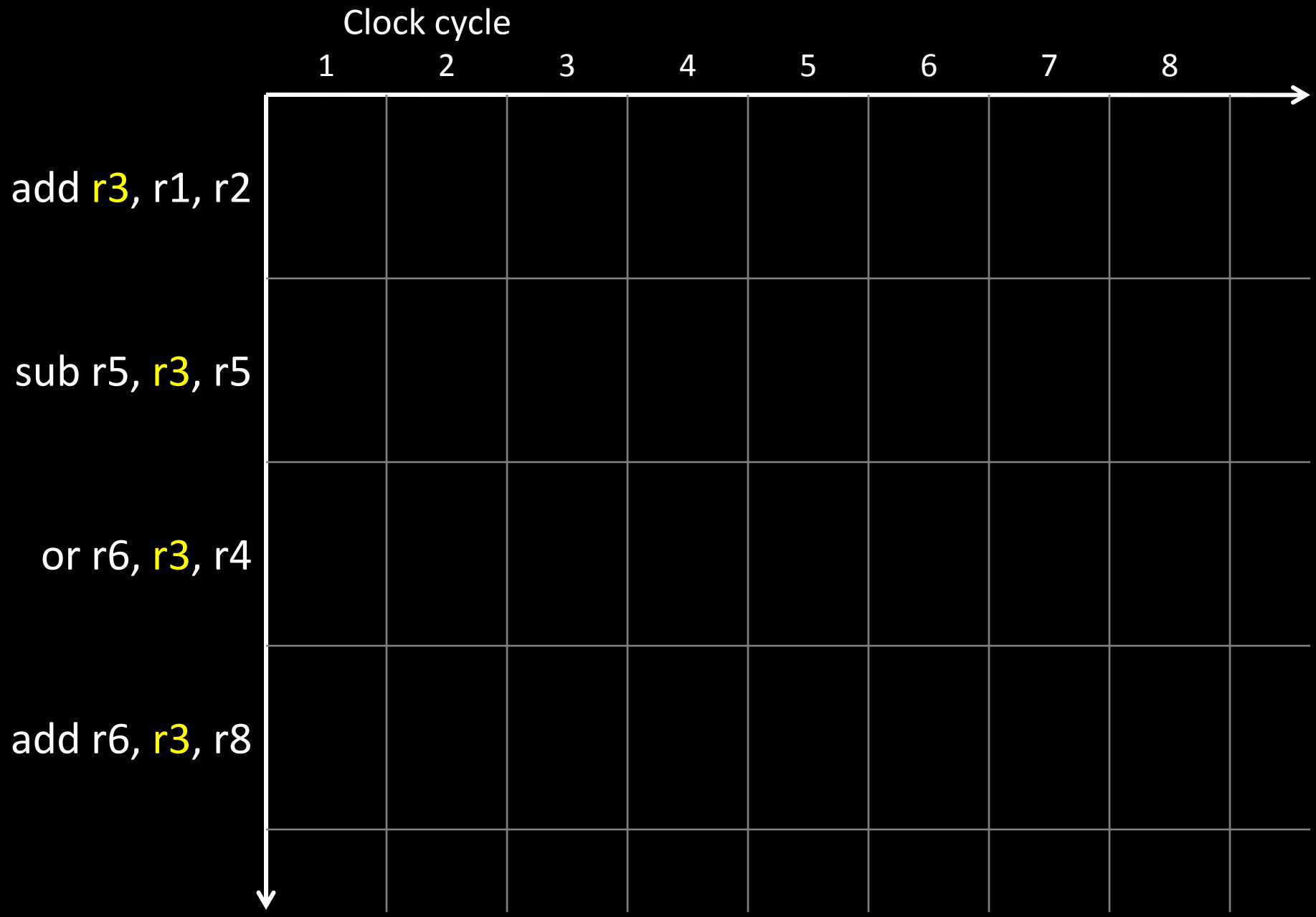
What to do if data hazard detected?

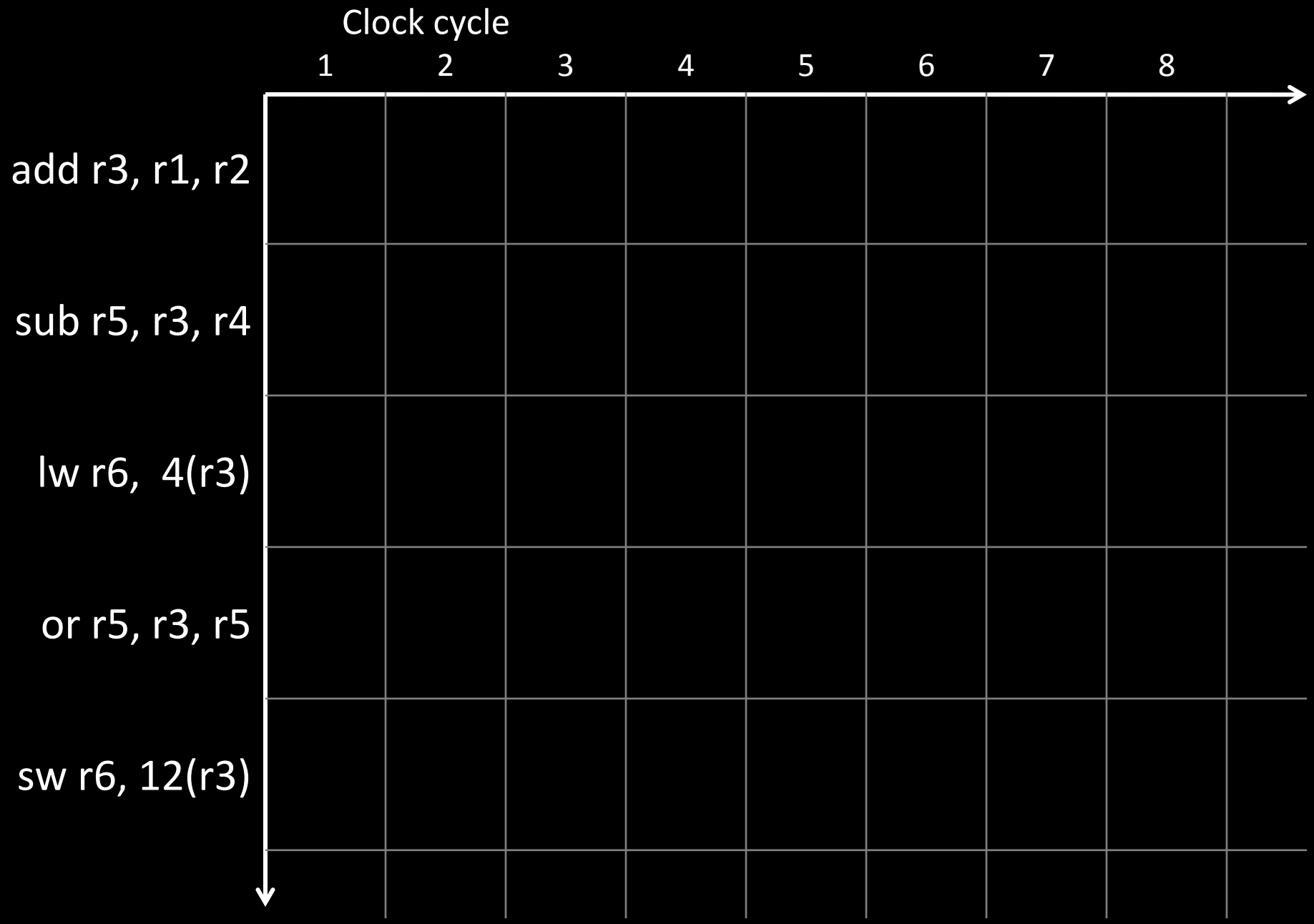

Clock cycle
1
2
3
4
5
6
7
8
add r3, r1, r2
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8

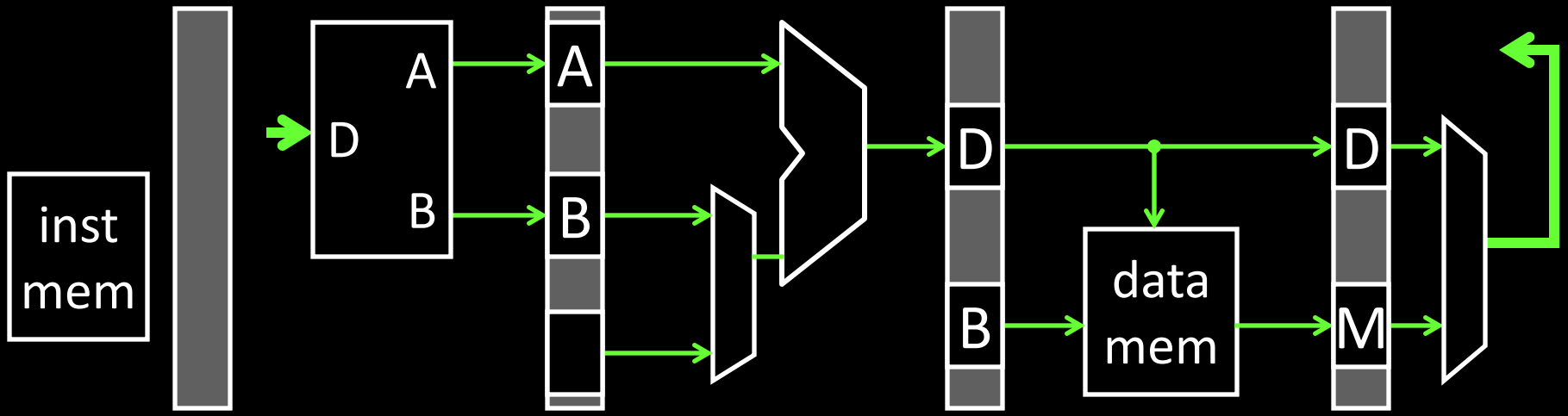

inst
mem
inst
+4
PC
D
rD
A
B
rA
rB
A
B
Rd
WE
Op
nop
/stall
D
B
Rd
WE
Op
data
mem
D
M
Rd
WE
Op

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
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8


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)



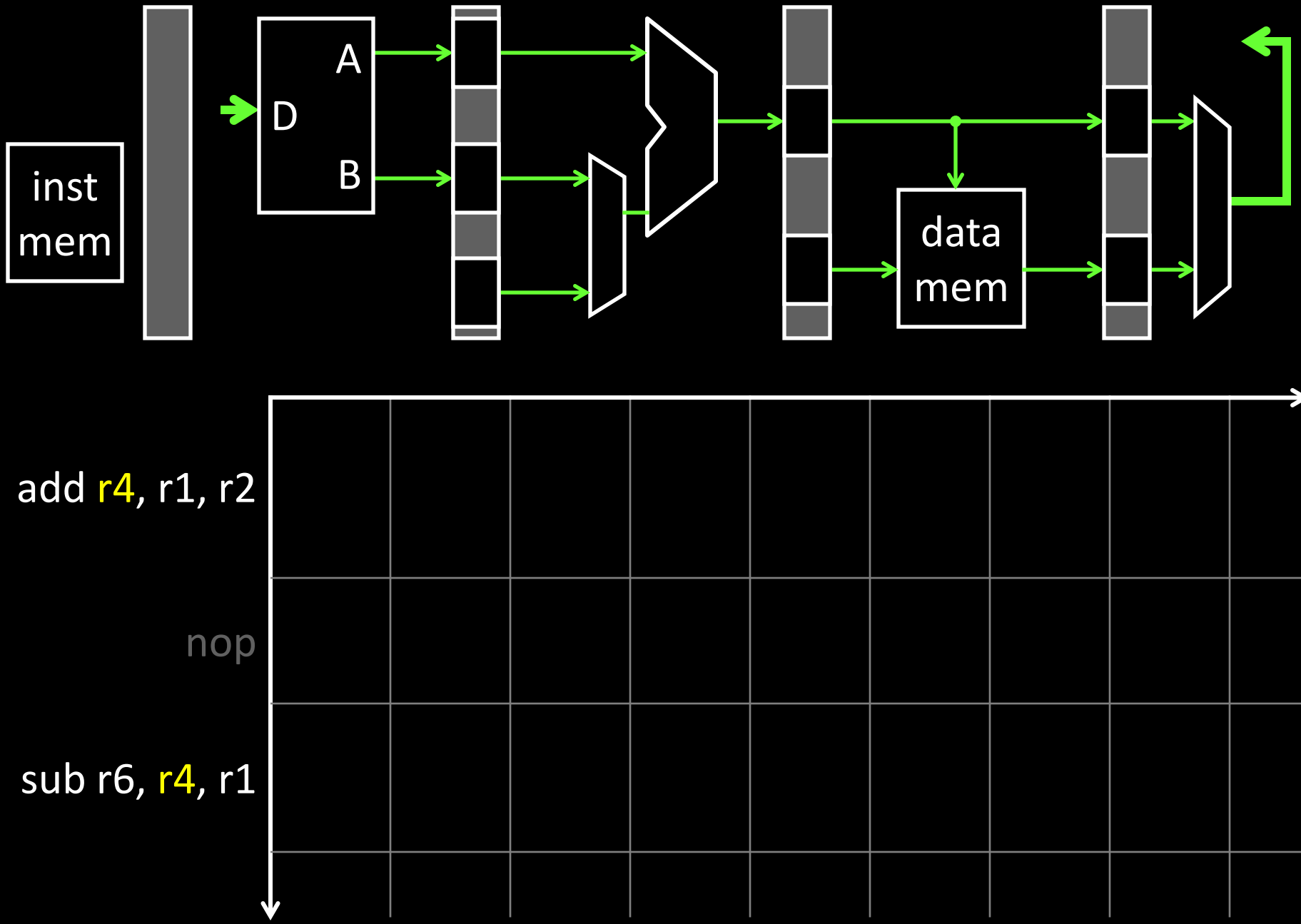
Forward correct value from? to?

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


inst
mem
A
D
B
data
mem
add r4, r1, r2
nop
sub r6, r4, r1

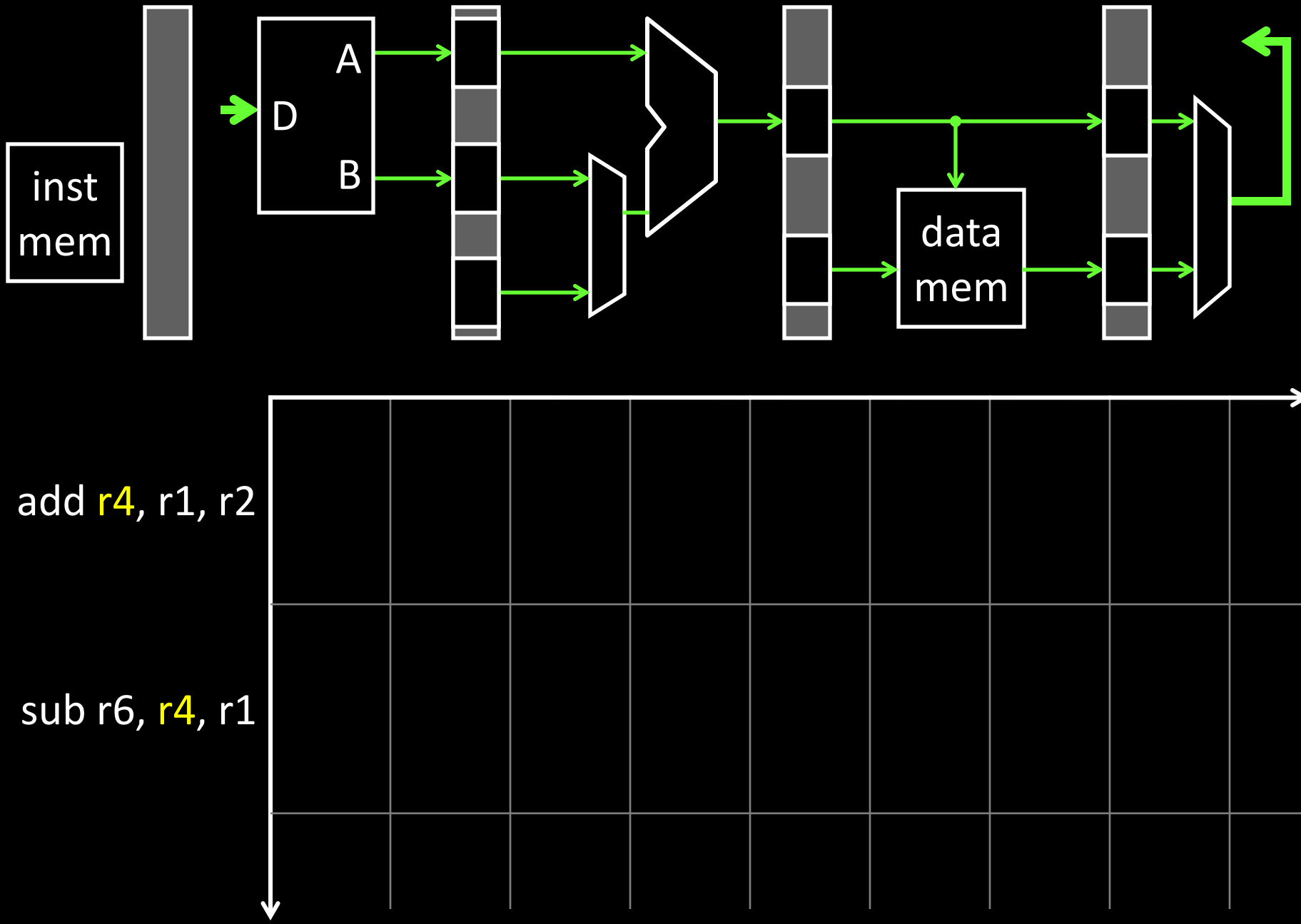
WB to EX Bypass

- EX needs value being written by WB

Resolve:

Add bypass from WB final value to start of EX

Detect:


inst
mem
A
D
B
data
mem
add r4, r1, r2
sub r6, r4, r1

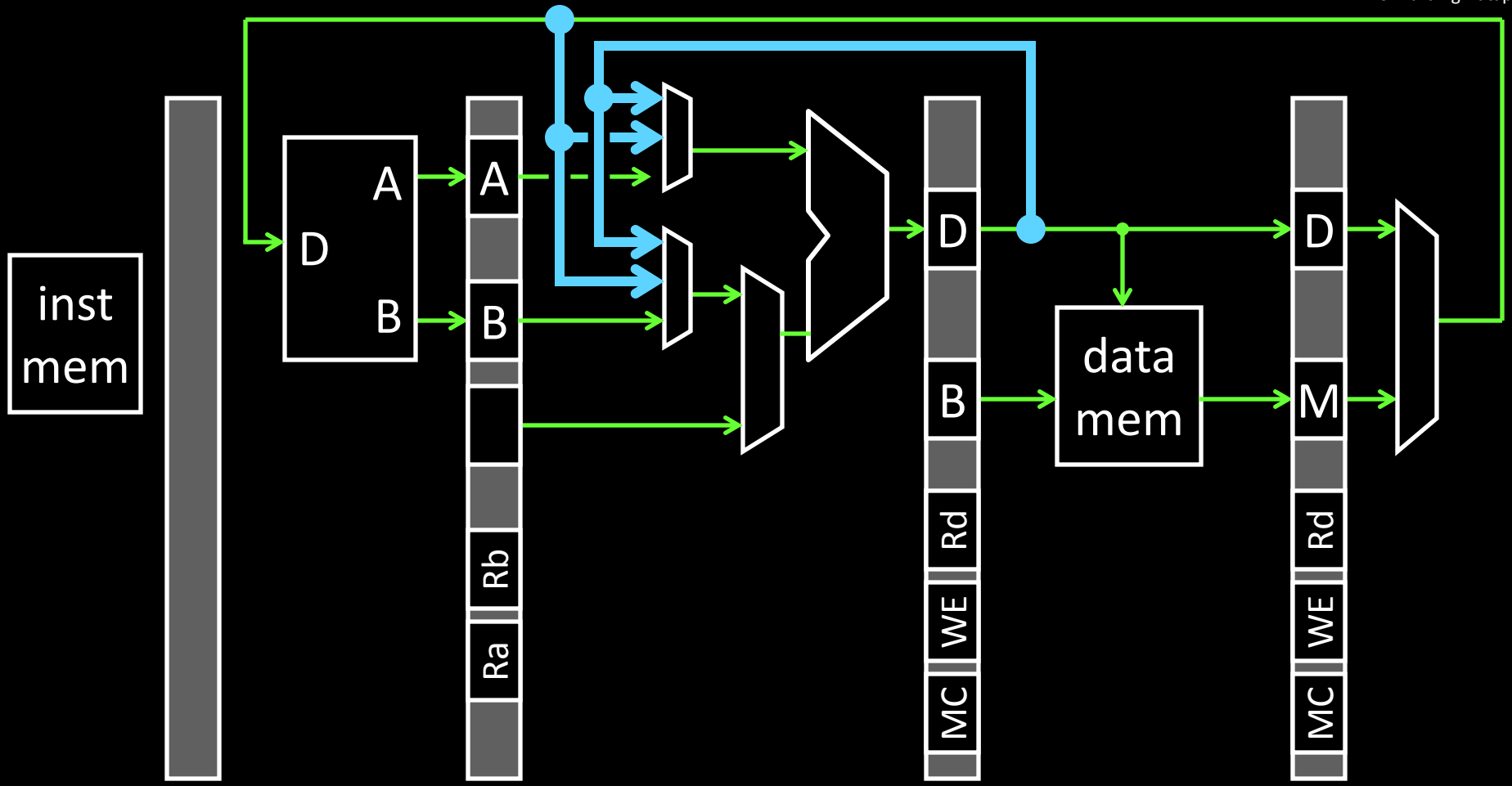
MEM to EX Bypass

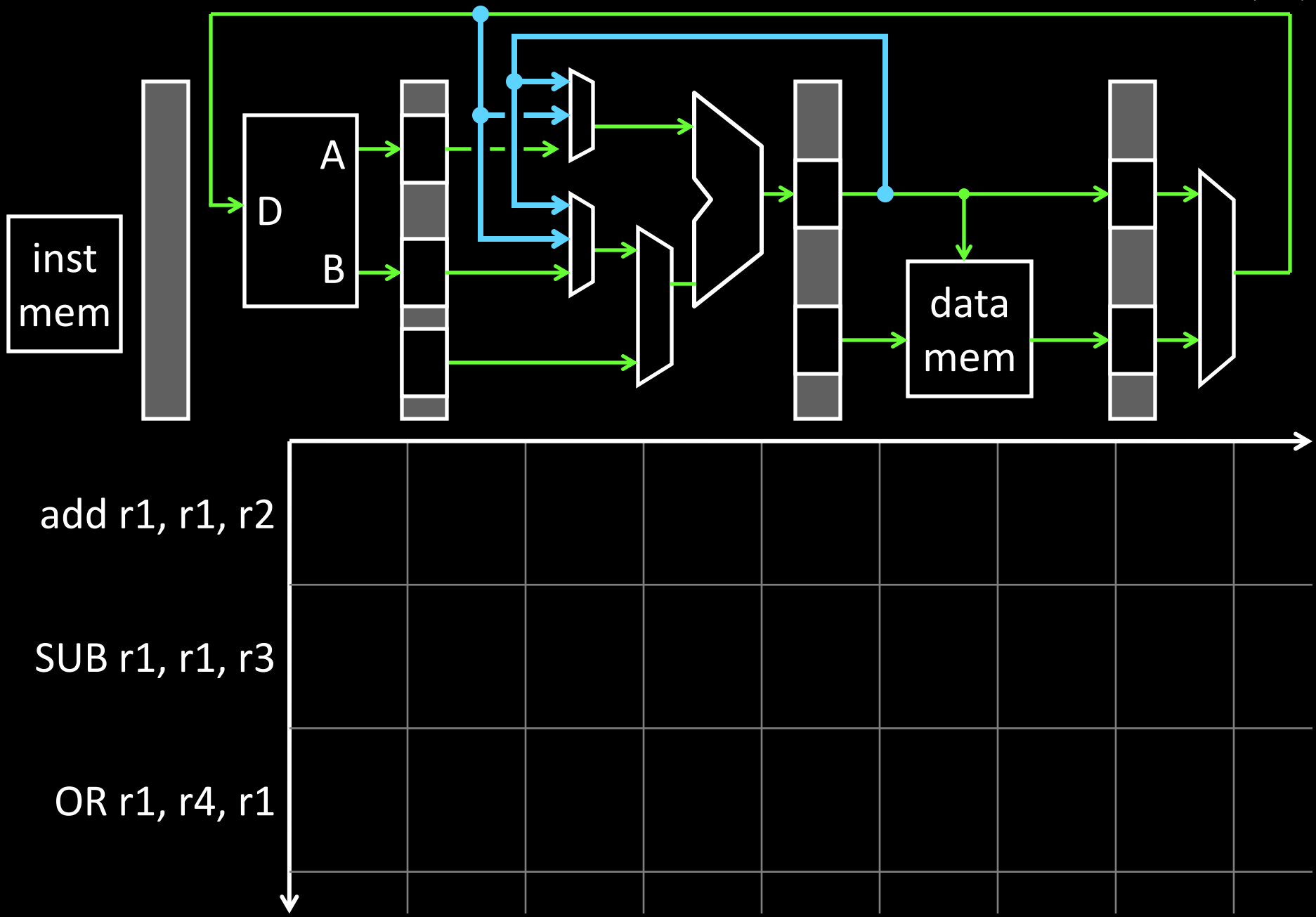
- EX needs ALU result that is still in MEM stage

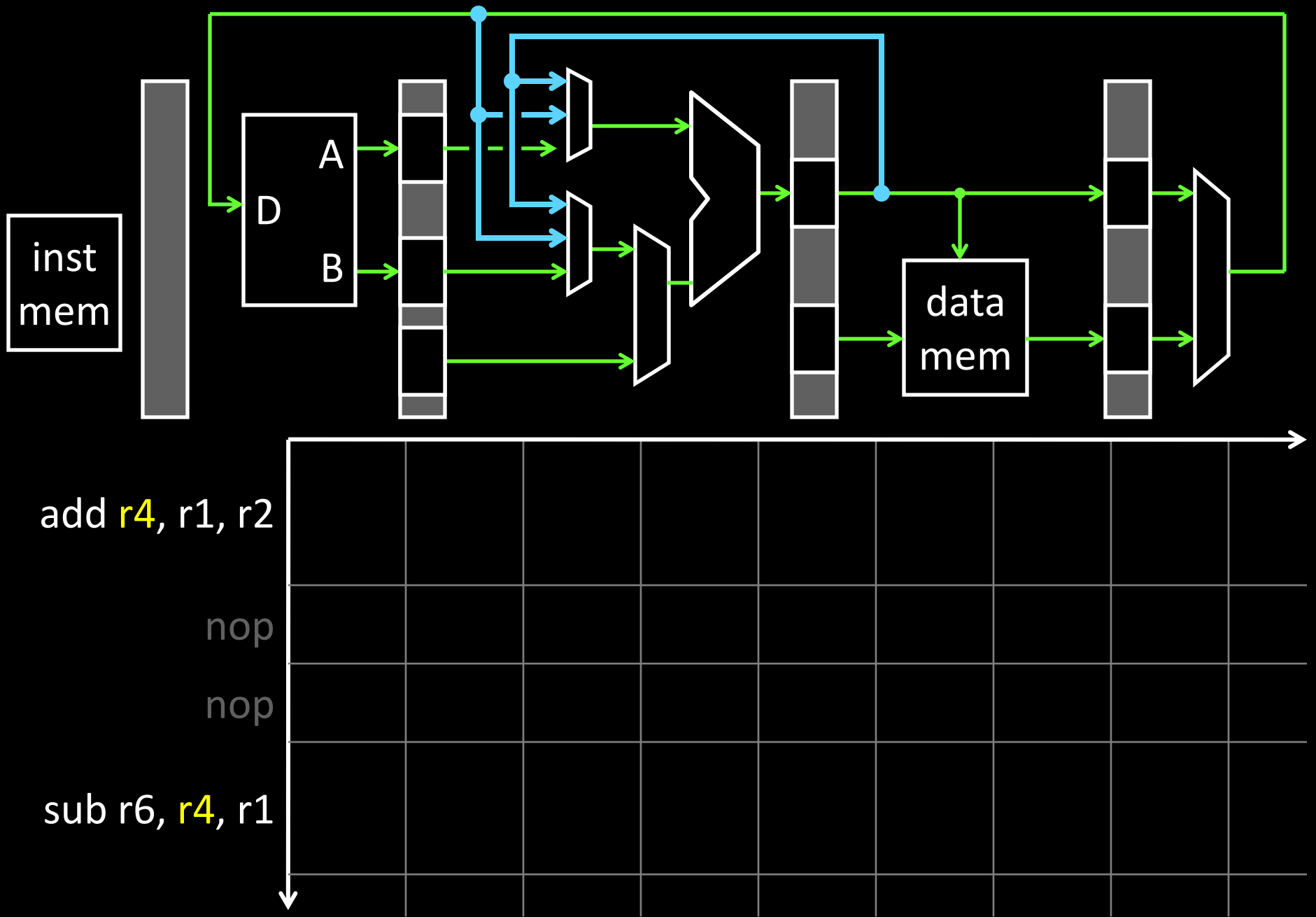
Resolve:

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

Detect:


inst
mem
A
D
B
A
B
Ra
Rb
D
B
Rd
WE
MC
data
mem
D
M
Rd
WE
MC


inst
mem
A
D
B
data
mem
add r1, r1, r2
SUB r1, r1, r3
OR r1, r4, r1


inst
mem
A
D
B
data
mem
add r4, r1, r2
nop
nop
sub r6, r4, r1

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:

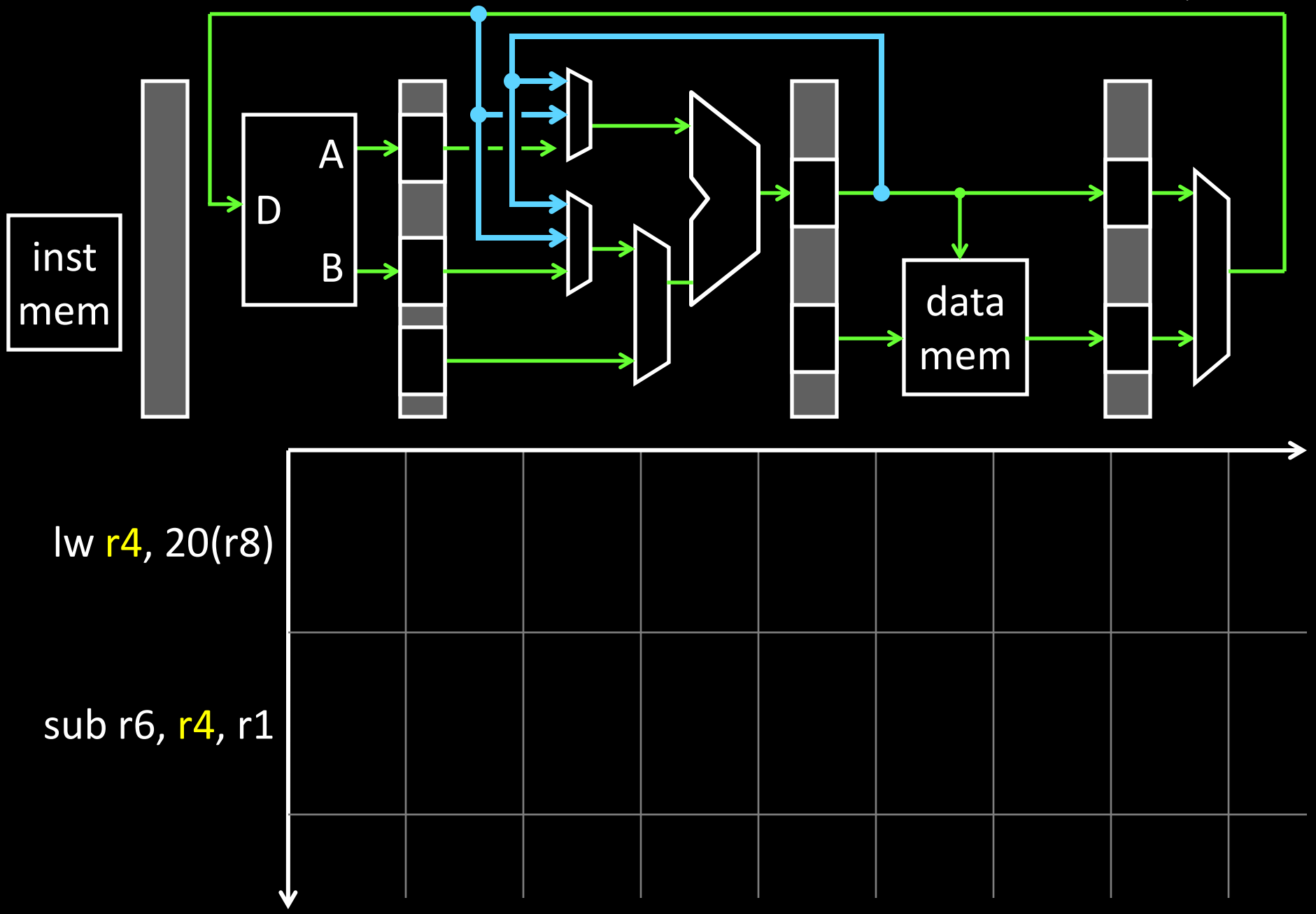
Add a bypass around register file (WB to ID)

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)
```


inst
mem
A
D
B
data
mem
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
add    r2, r6, r3
lw     r6, 24(r3)
sw     r6, 12(r2)
```

Delay Slot(s)

- Modify ISA to match implementation

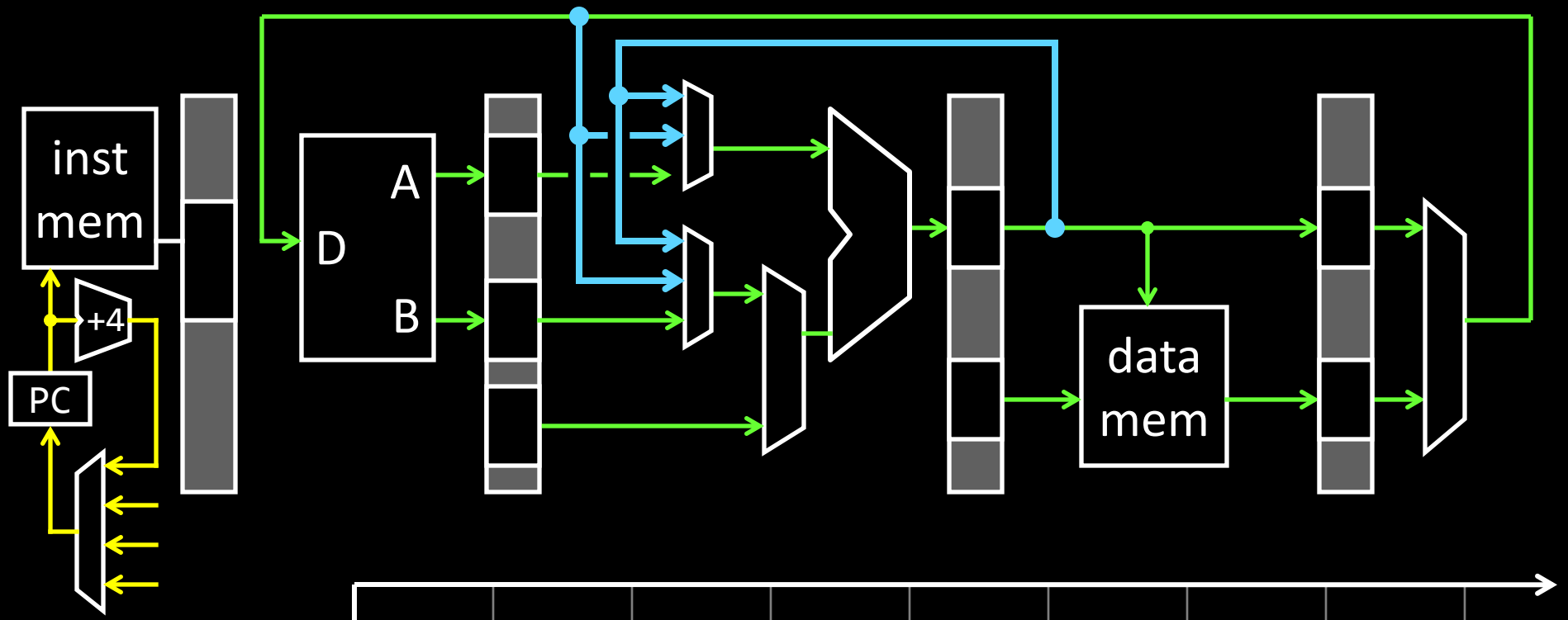
Stall

- Pause current and all subsequent instructions

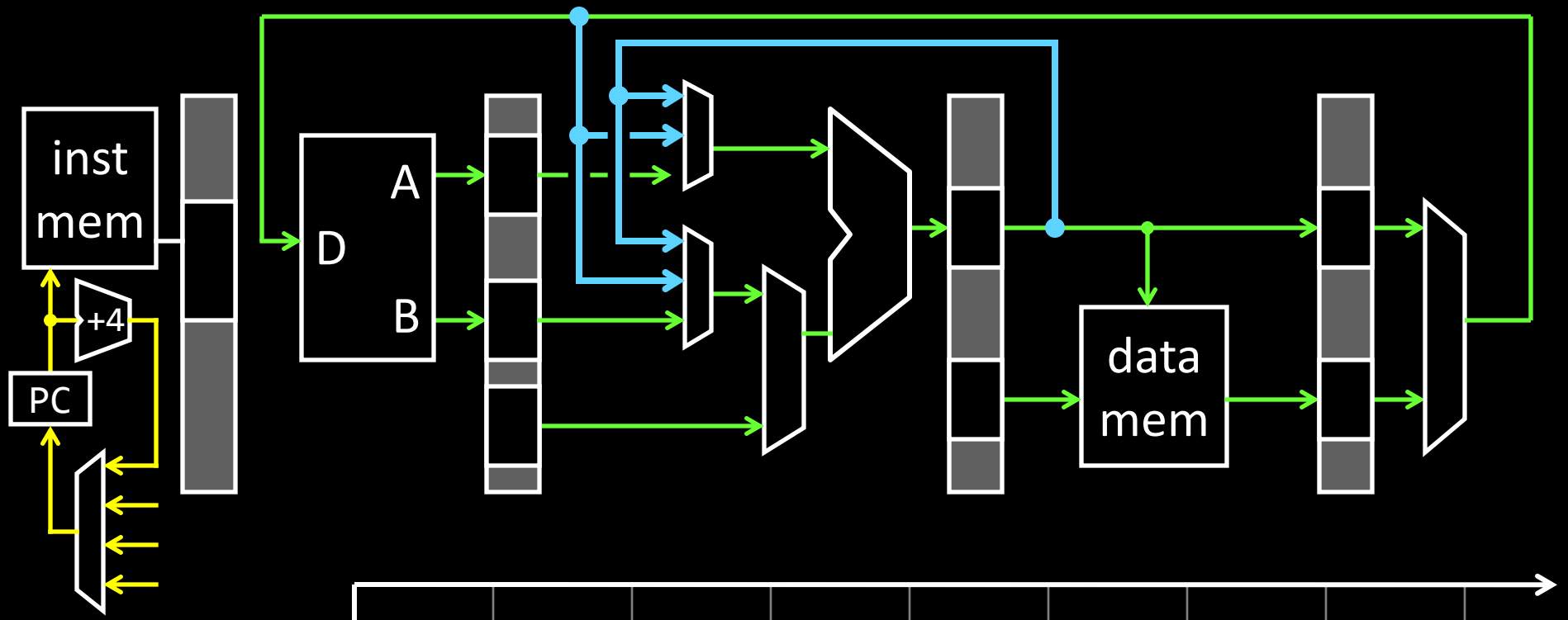
Forward/Bypass

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

Tradeoffs?


inst
mem
+4
PC
A
D
B
data
mem


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


inst
mem
+4
PC
A
D
B
data
mem


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

Control Hazards

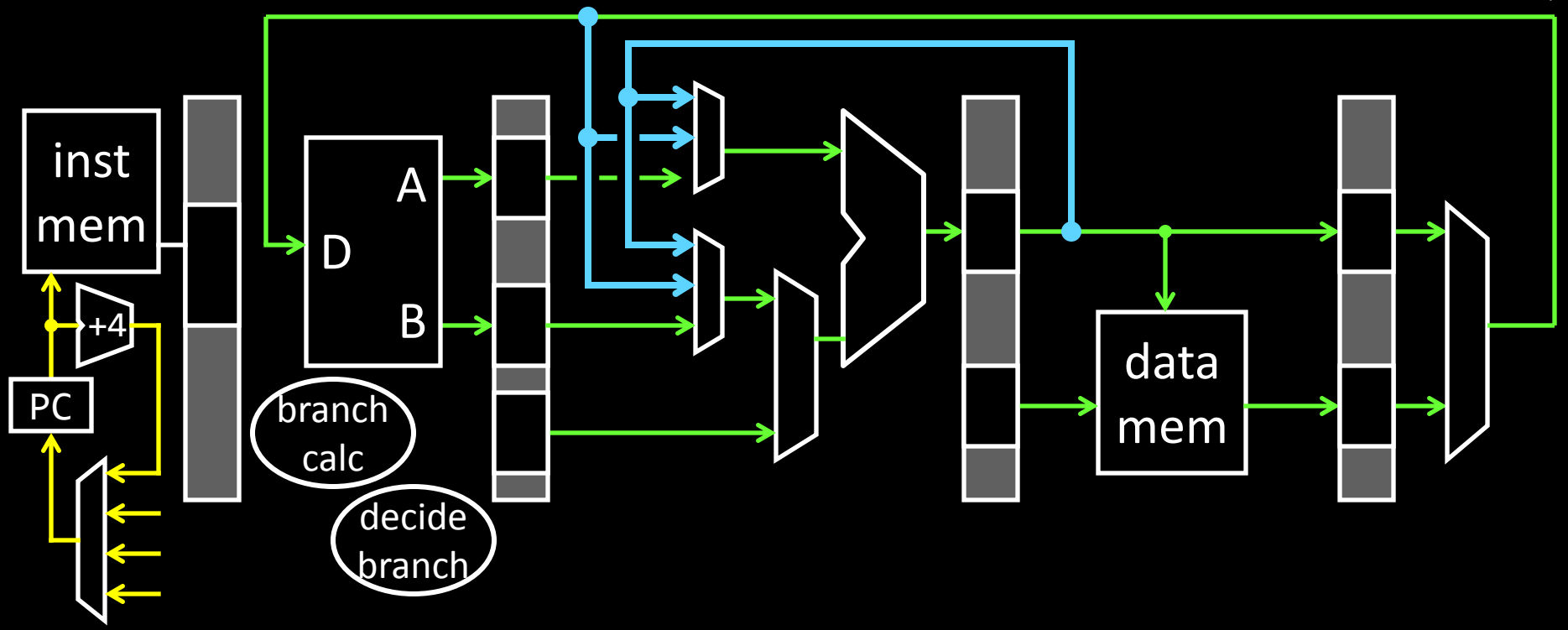
- instructions are fetched in stage 1 (IF)
- branch and jump decisions occur in stage 3 (EX)
- i.e. next PC is not known until 2 cycles after branch/jump

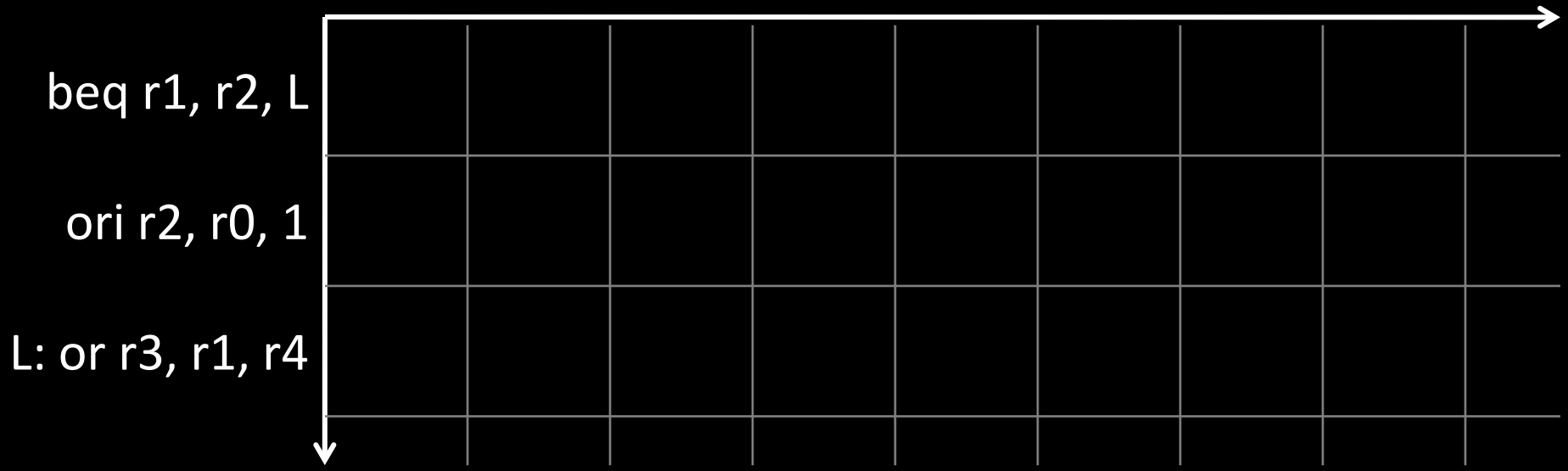
Delay Slot

- 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


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

- 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

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