Pipeline Control Hazards and Instruction Variations

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H Appendix 4.8

Goals for Today

Recap: Data Hazards

Control Hazards

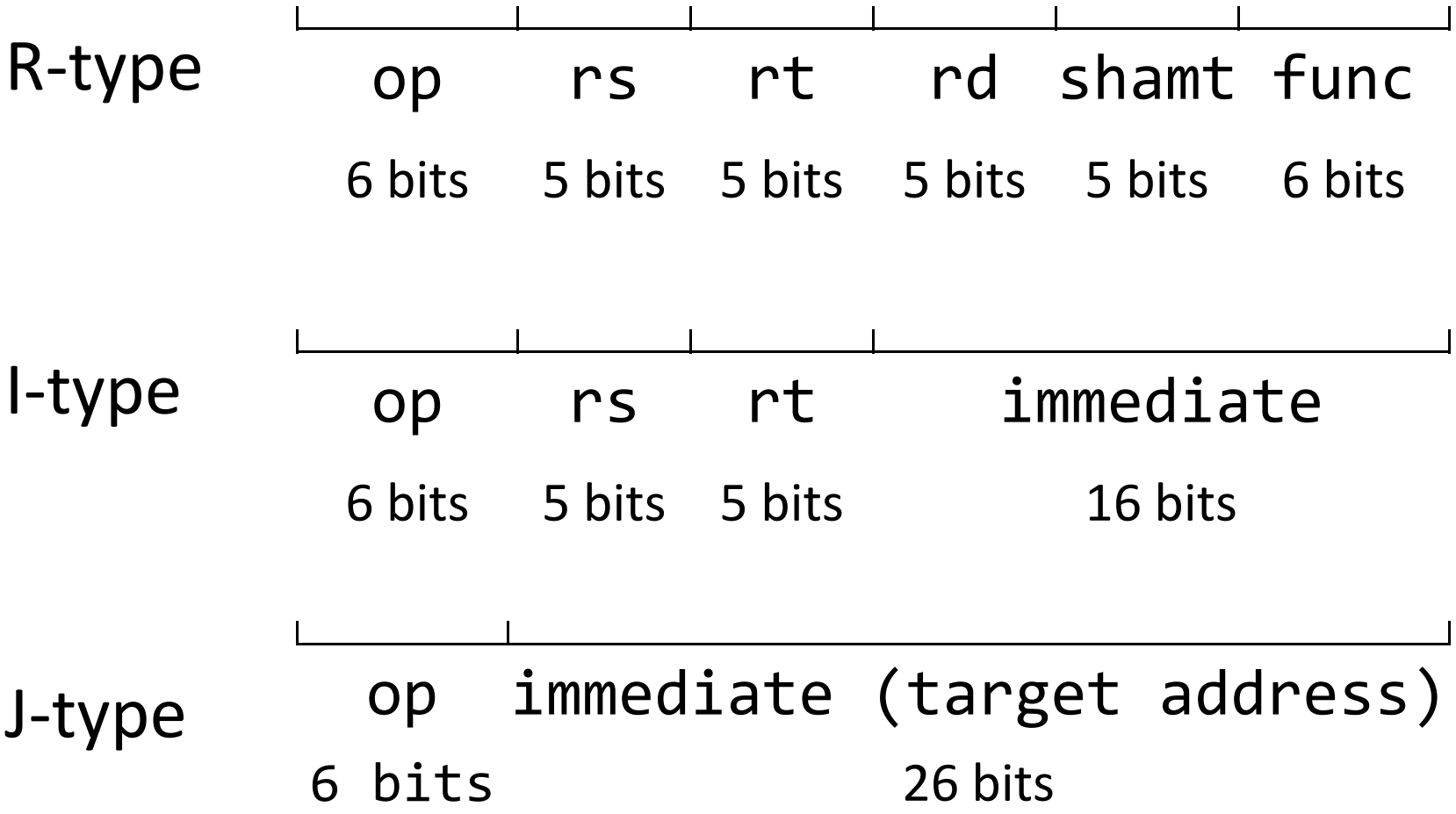
- What is the next instruction to execute if a branch is taken? Not taken?
- How to resolve control hazards
- Optimizations

Next time: Instruction Variations

- Instruction Set Architecture Variations
  - ARM
  - X86
- RISC vs CISC
- The Assembler

Recall: MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats



Recall: MIPS Instruction Types

Arithmetic/Logical

- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

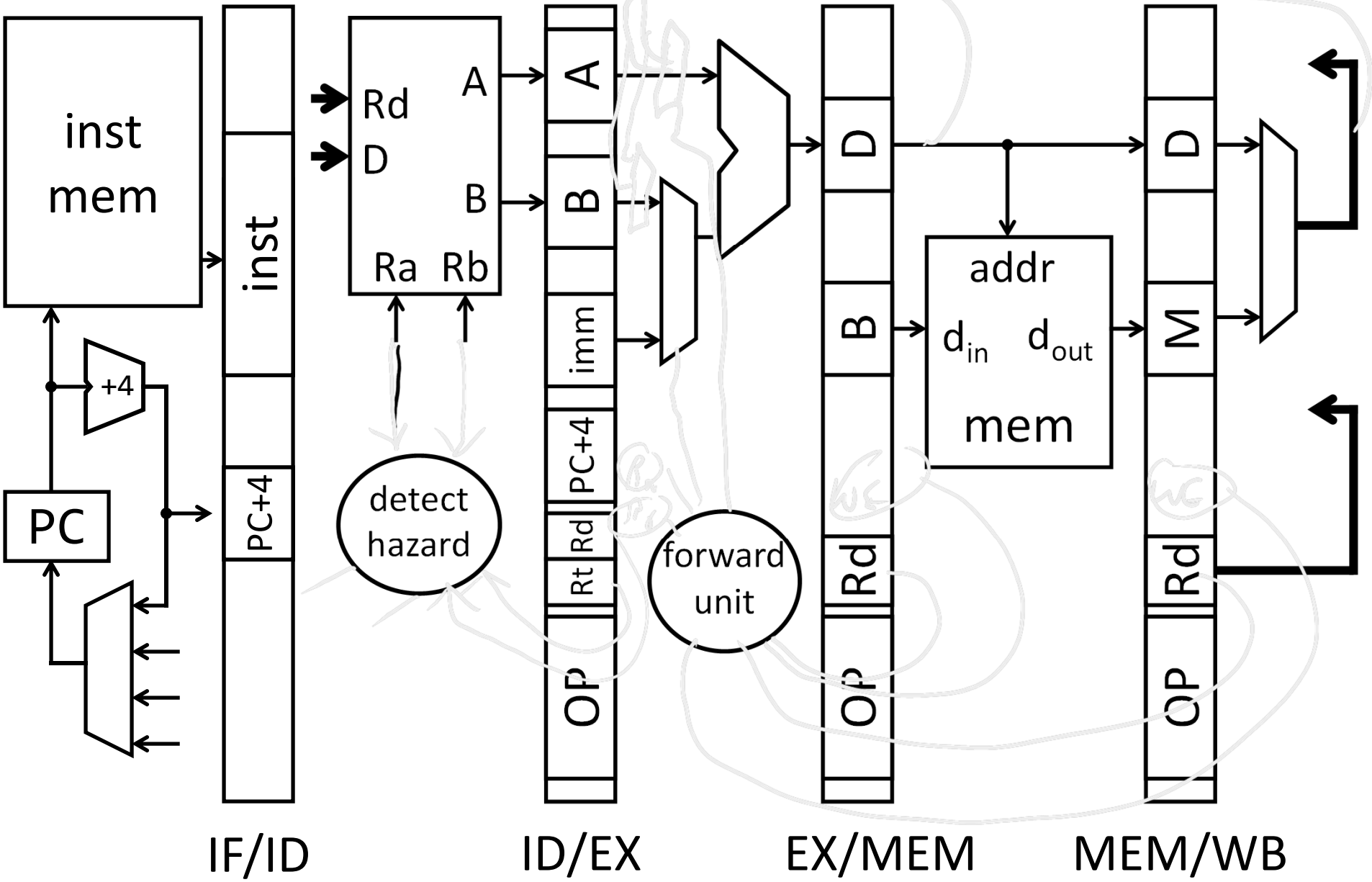
Memory Access

- load/store between registers and memory
- word, half-word and byte operations

Control flow

- conditional branches: pc-relative addresses
- jumps: fixed offsets, register absolute

Data Hazards

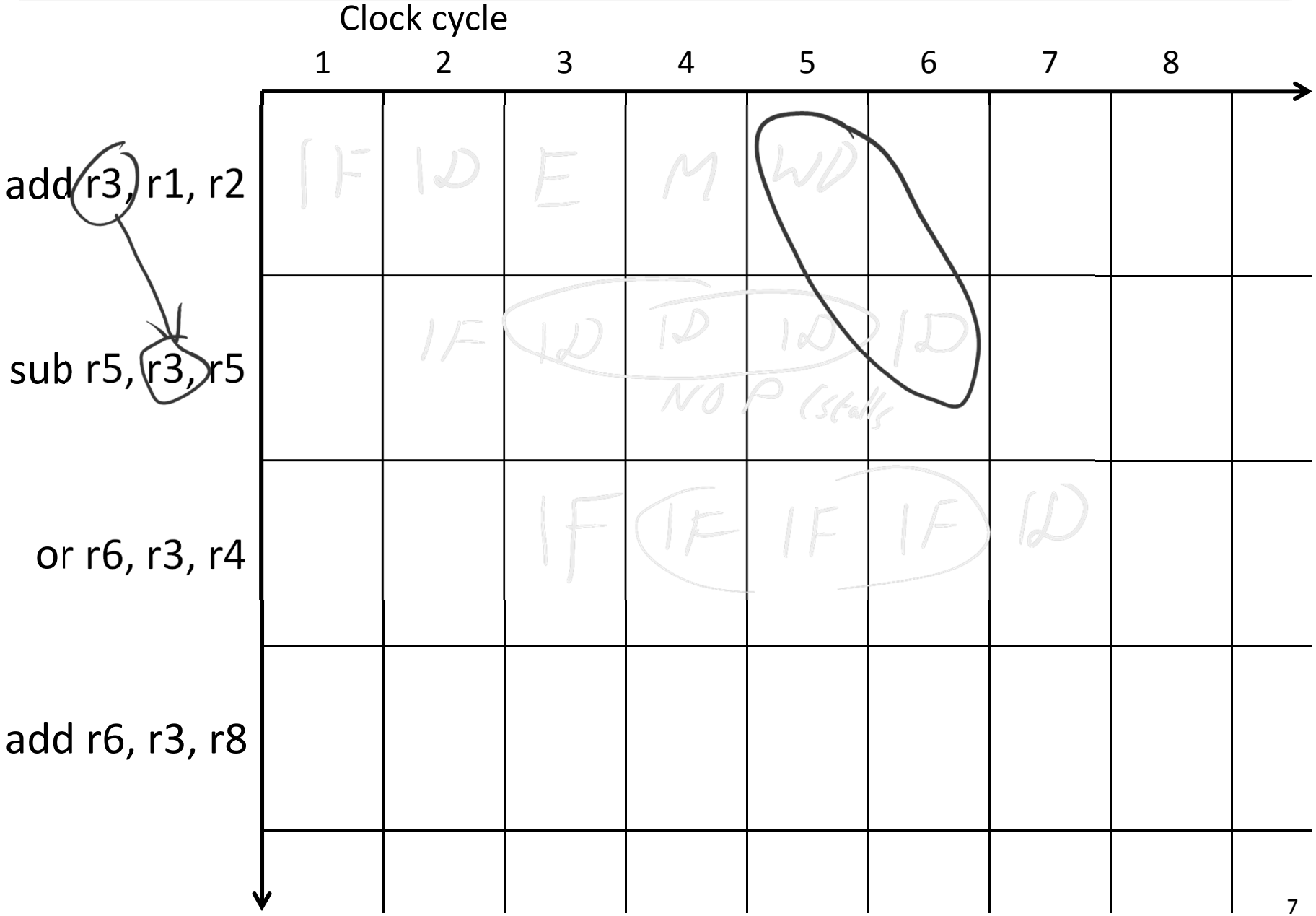


Resolving Data Hazards

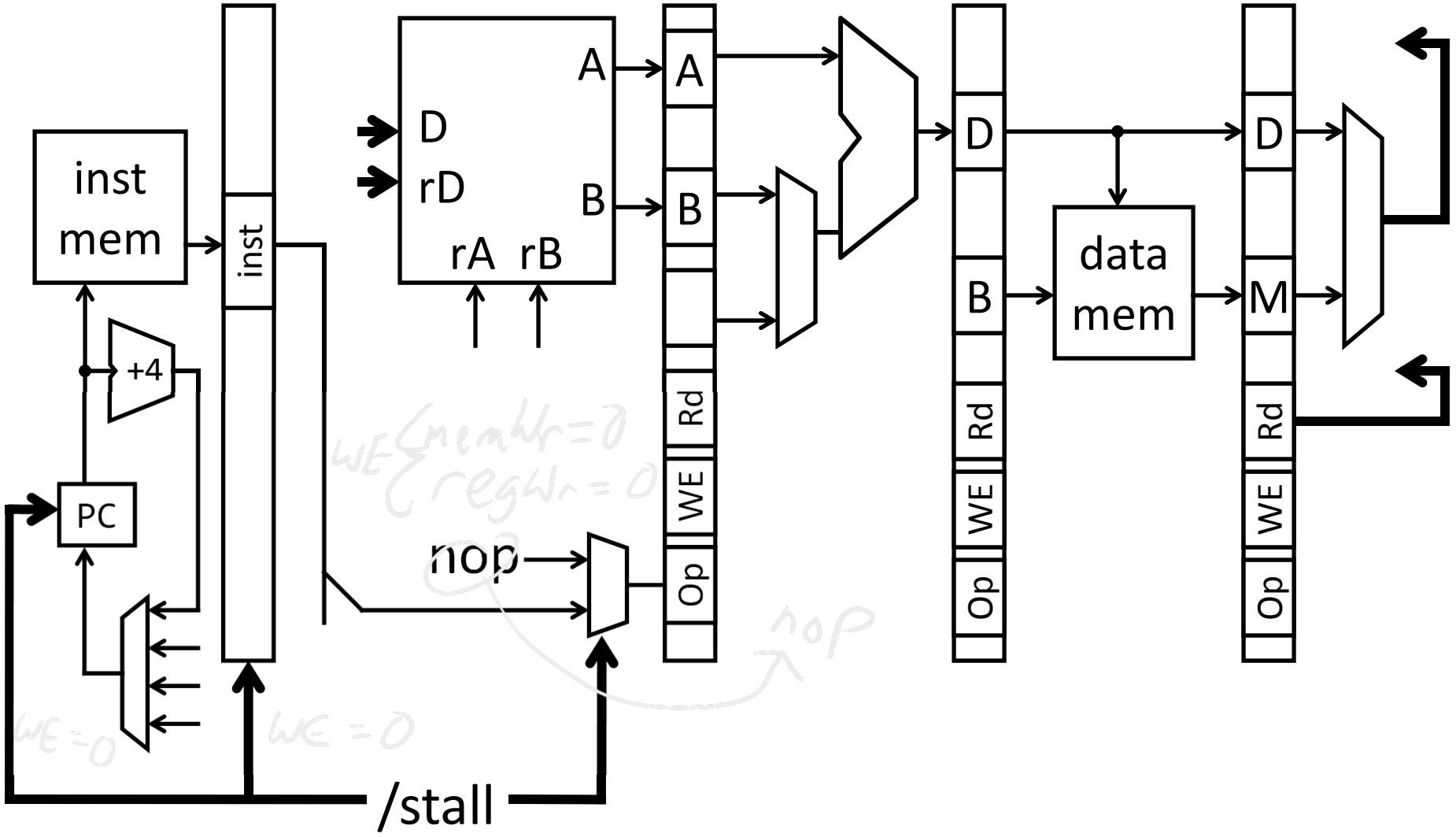
What to do if data hazard detected

- Stall
- Reorder instructions in SW
- Forward/Bypass

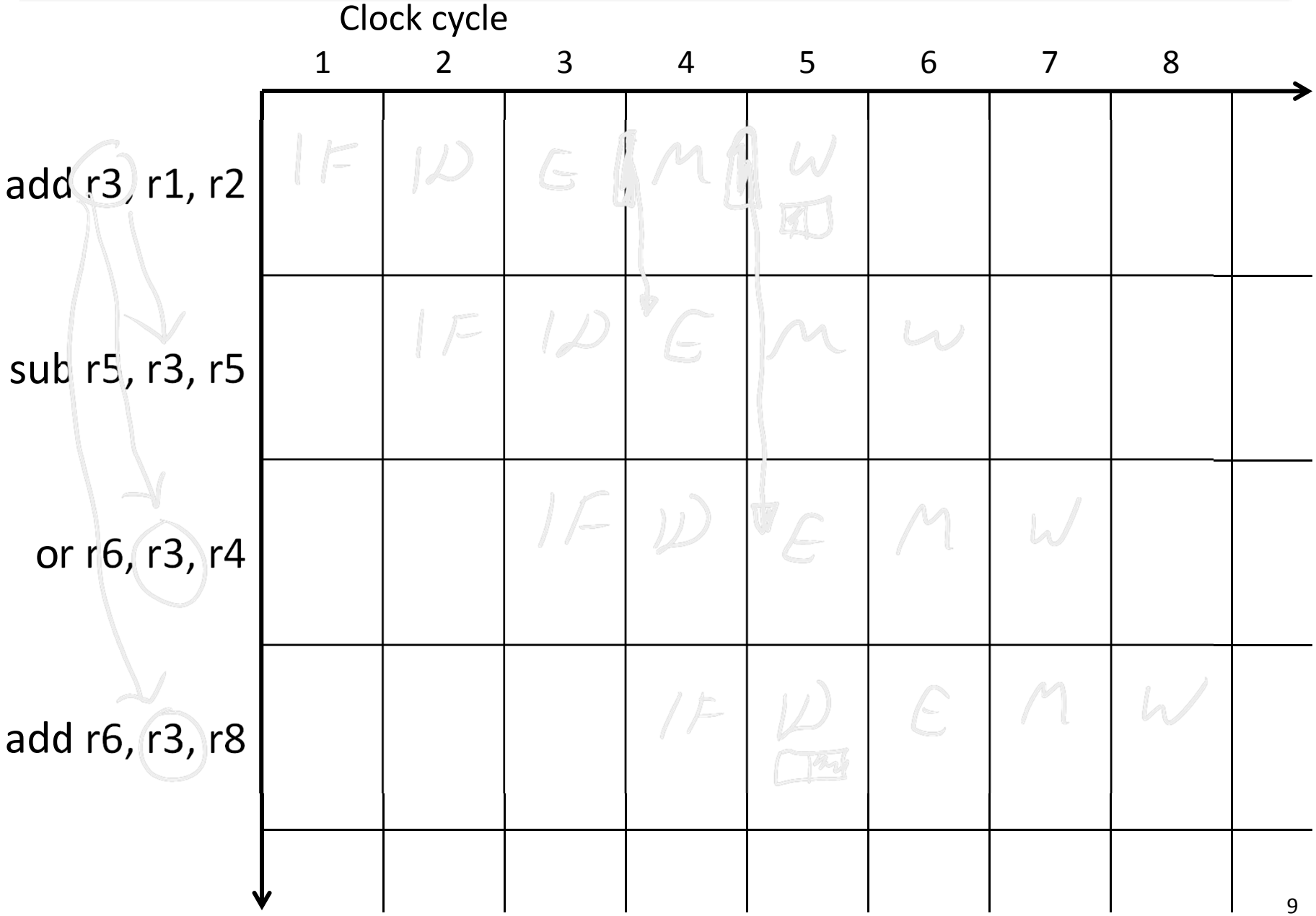
Stalling


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

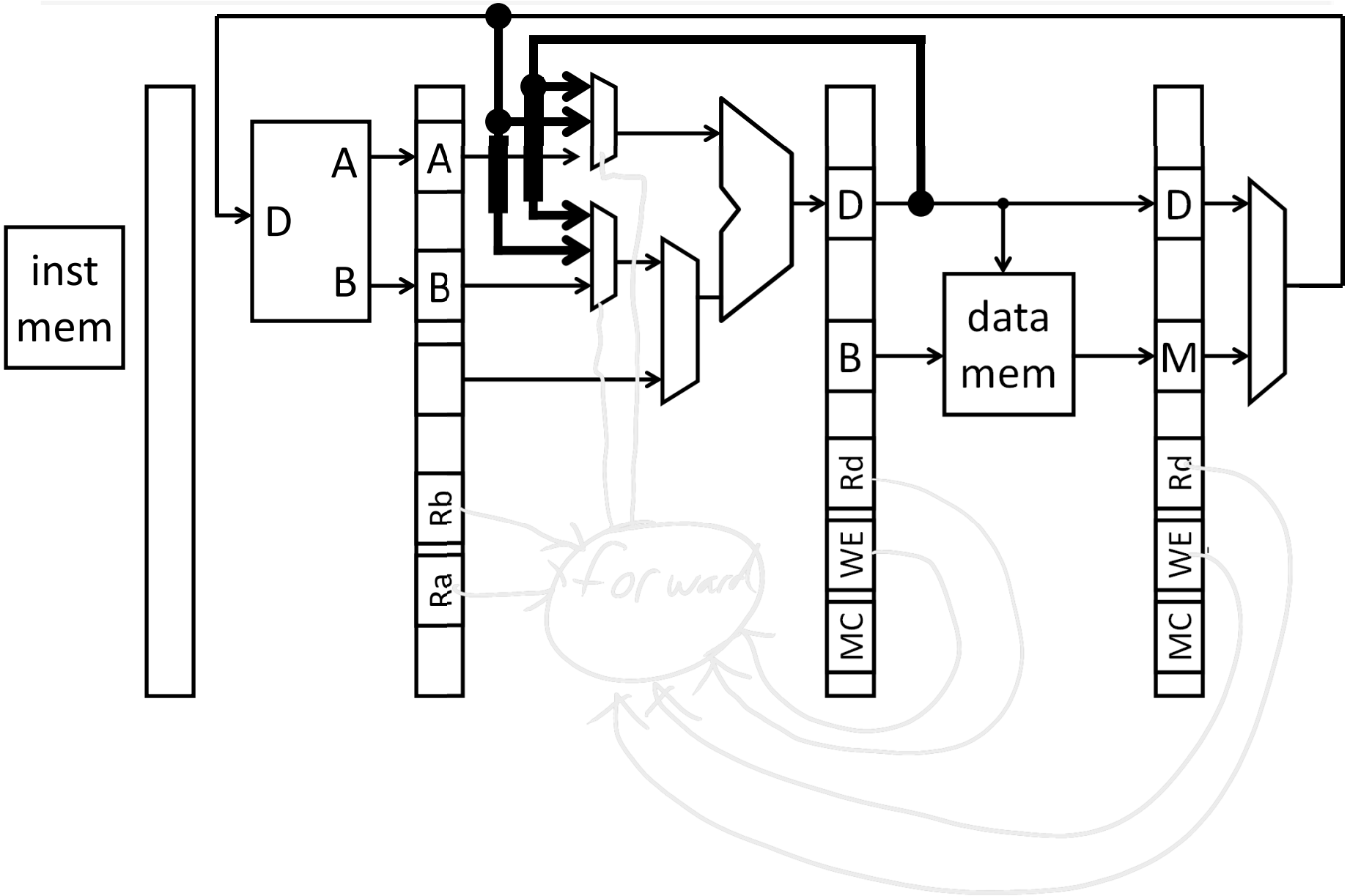
Stalling


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

Forwarding


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

Forwarding Datapath



Forwarding Datapath

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

How to detect? Logic in Ex Stage:

```
forward = (Ex/M.WE && EX/M.Rd != 0 &&
           ID/Ex.Ra == Ex/M.Rd)
        || (same for rB)
```

Forwarding Datapath

WB to EX Bypass

- EX needs value being written by WB
- Resolve:
- Add bypass from WB final value to start of EX

How to detect? Logic in Ex Stage:

```
forward = (M/WB.WE && M/WB.Rd != 0 &&
                ID/Ex.Ra == M/WB.Rd &&
            not (ID/Ex.WE && Ex/M.Rd != 0 &&
                    ID/Ex.Ra == Ex/M.Rd)
        || (same for rB)
```

Forwarding Datapath

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 Soln: (Hack) just negate register file clock

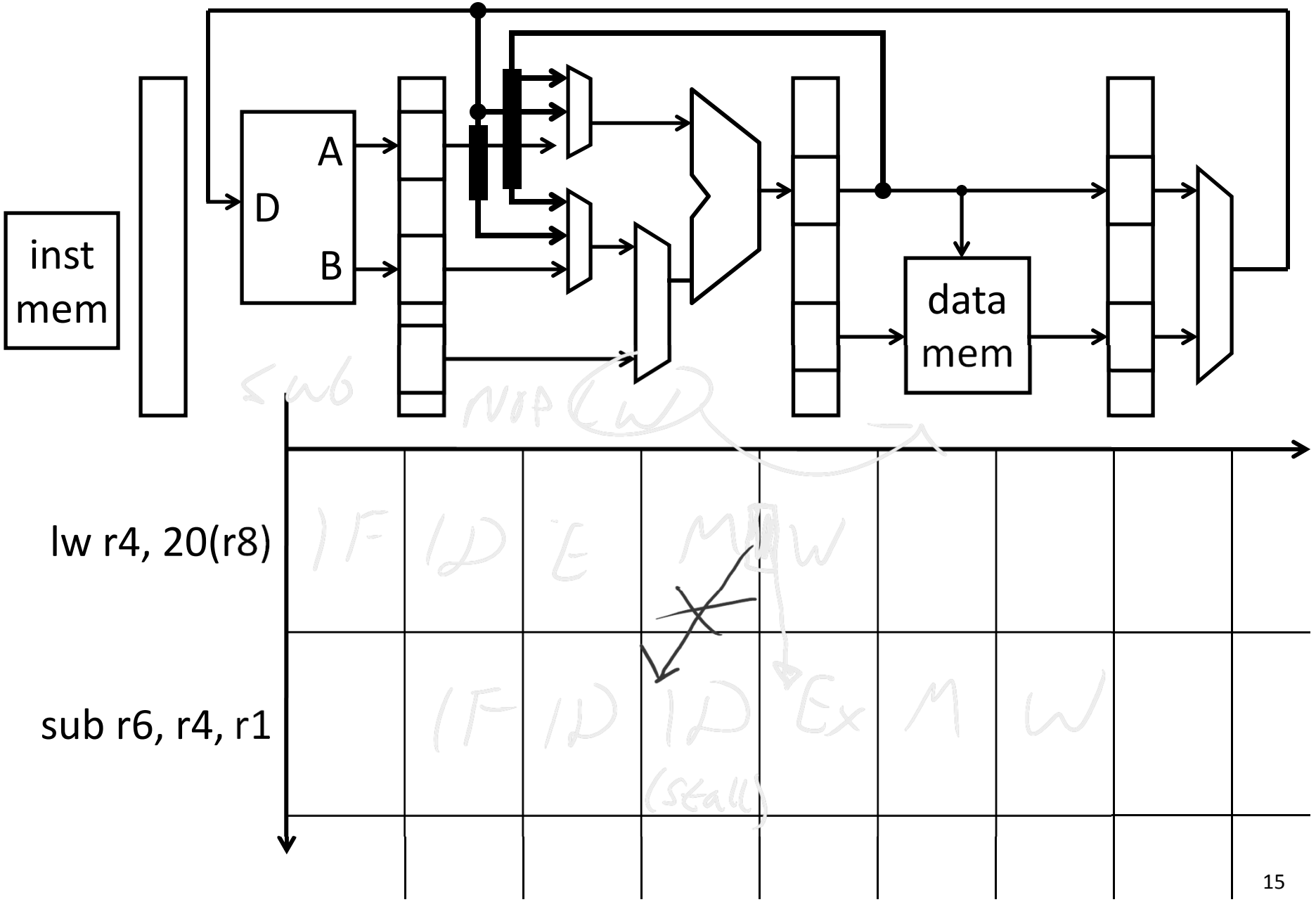
– writes happen at end of first half of each clock cycle

– reads happen during second half of each clock cycle

Quiz 2

```
add     r3, r1, r2
nand    r5, r3, r4
add     r2, r6, r3
lw      r6, 24(r3)
sw      r6, 12(r2)
```

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

For stall, how to detect? Logic in ID Stage

- Stall = ID/Ex.MemRead &&
  (IF/ID.Ra == ID/Ex.Rd || IF/ID.Rb == ID/Ex.Rd)

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

Administrivia

Prelim1: ***today*** Tuesday, February 28th in evening

- Location: GSH132: Goldwin Smith Hall room 132
- Time: We will start at 7:30pm sharp, so come early

- Closed Book: ***NO NOTES, BOOK, CALCULATOR, CELL PHONE***
  - Cannot use electronic device or outside material
- Practice prelims are online in CMS
- Material covered everything up to end of ***last*** 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

Online Survey results

- More chairs in lab sections
- Better synchronization between lecture and homework
- Lab and lecture may be a bit out of sync at times

Project1 (PA1) due next Monday, March 5th

- 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

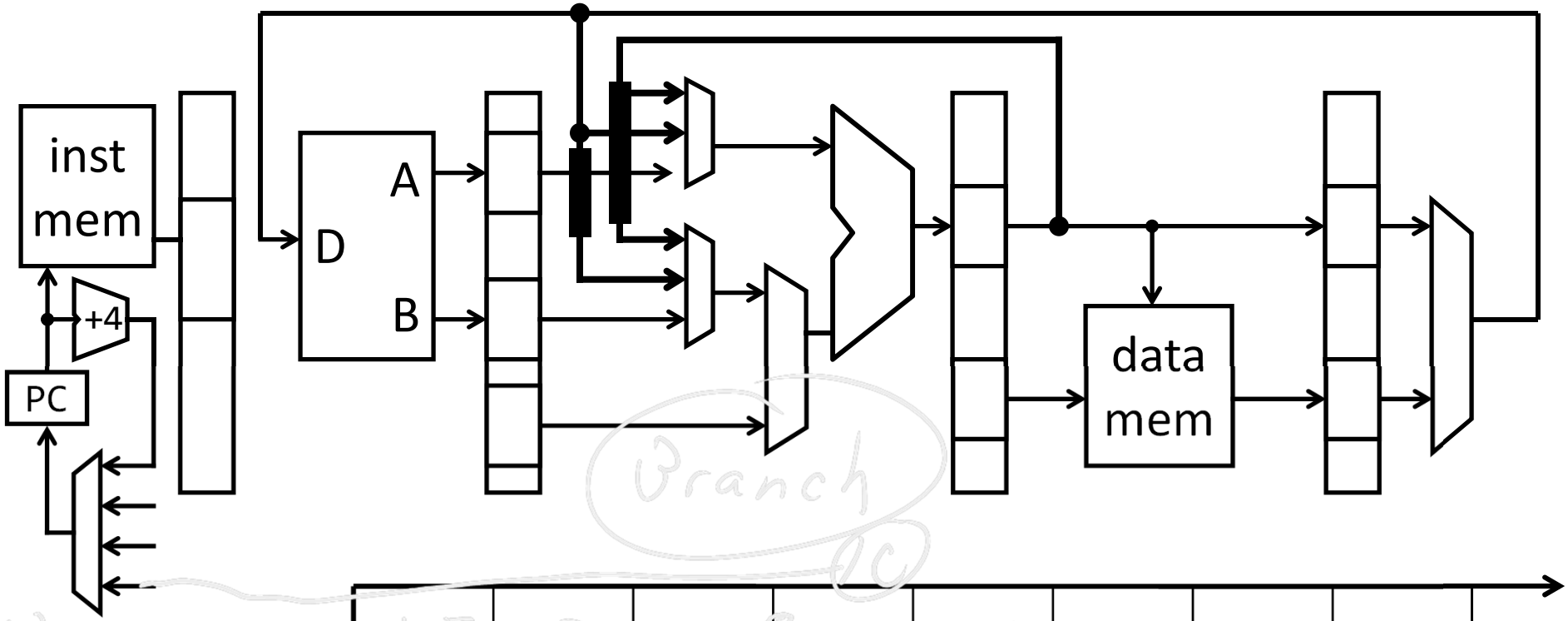
- Lab Section, Piazza.com, Office Hours, Homework Help Session,
- Class notes, book, Sections, CSUGLab

Control Hazards

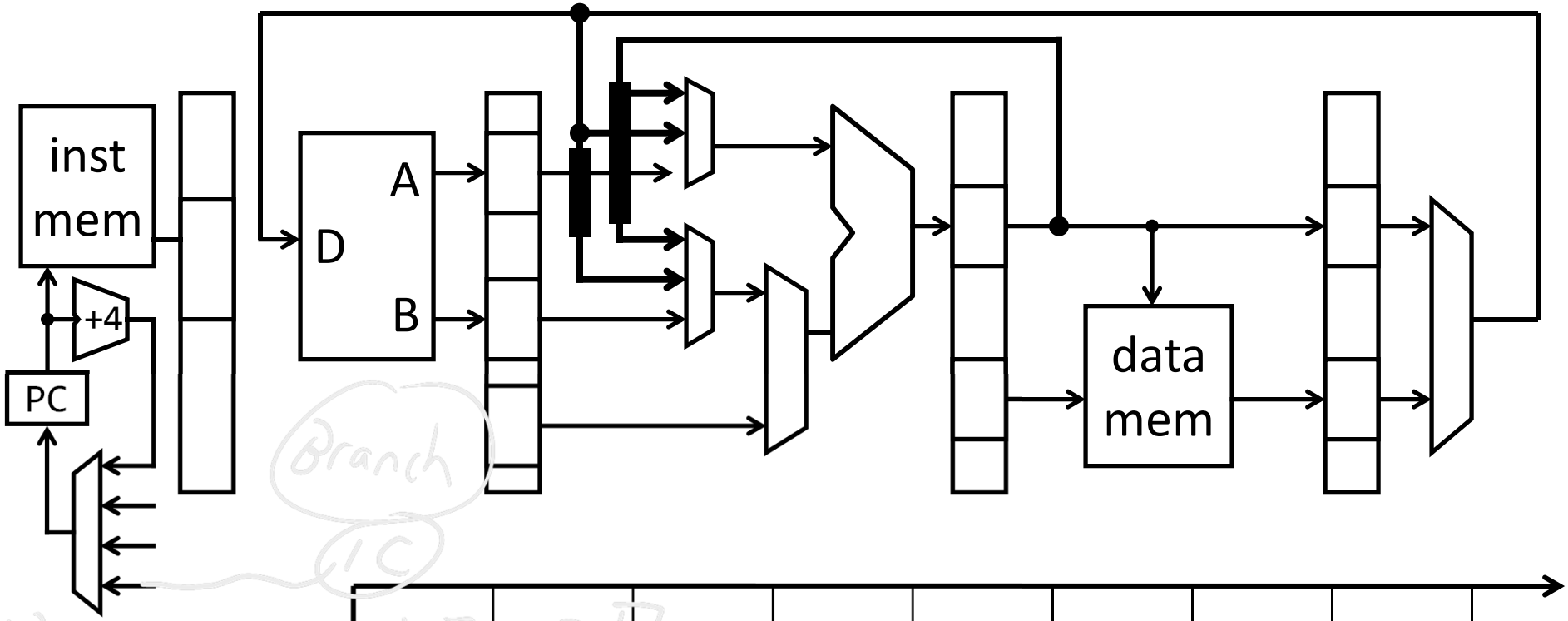
What about branches?

- Can we forward/bypass values for branches?
  - We can move branch calc from EX to ID
  - will require new bypasses into ID stage; or can just zap the second instruction
- What happens to instructions following a branch, if branch taken?
  - Need to zap/flush instructions
- Is there still a performance penalty for branches
  - Yes, need to stall, then may need to zap (flush) subsequent instuctions that have already been fetched.

Control Hazards



Control 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

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

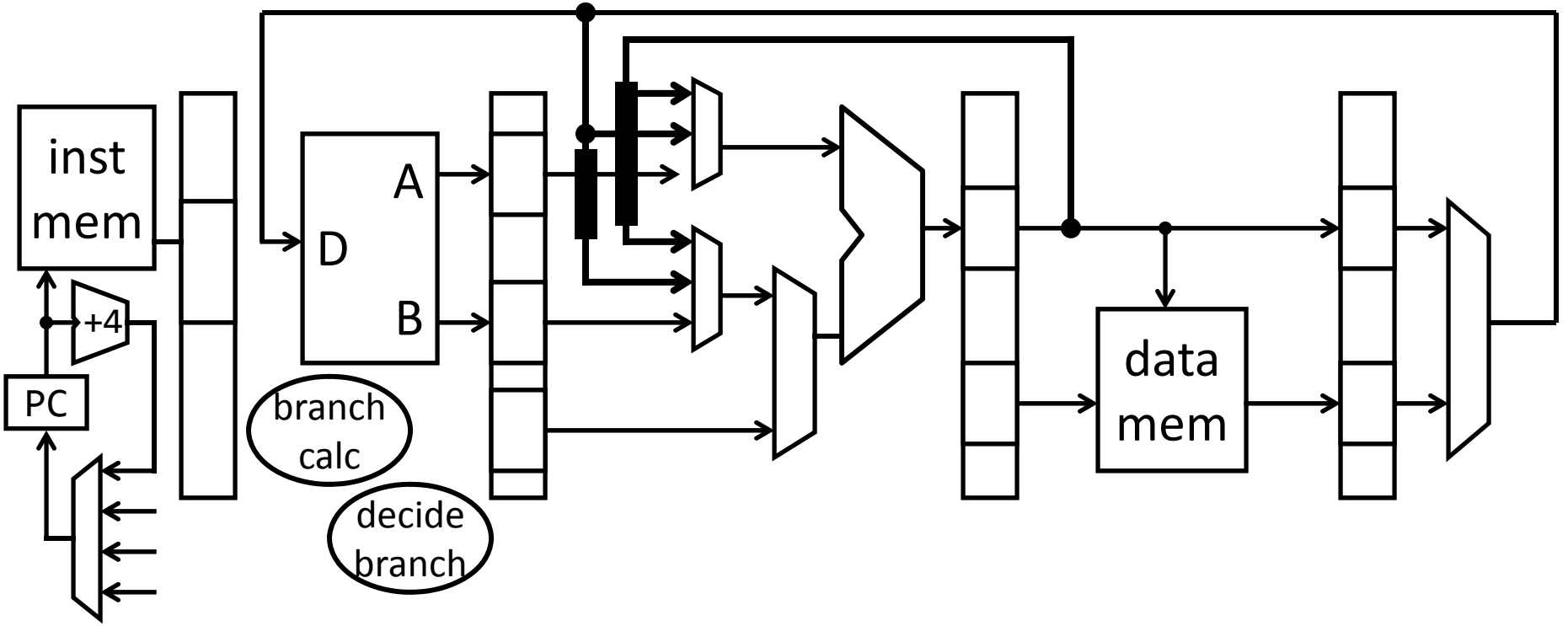
Delay Slot

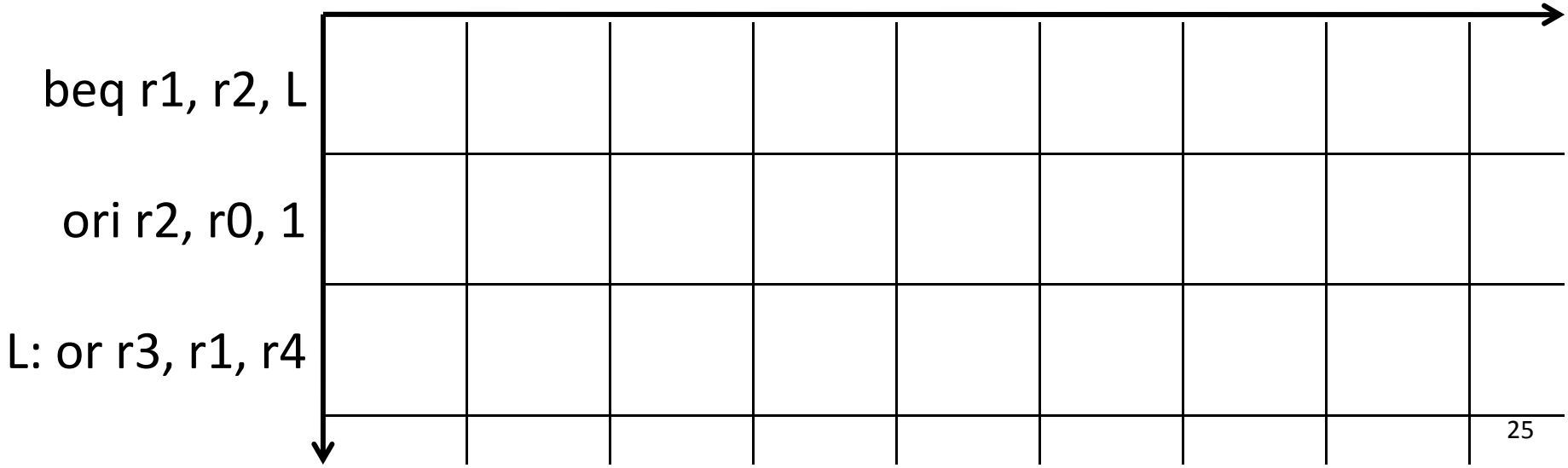
- ISA says N instructions after branch/jump ***always*** executed
  - MIPS has 1 branch delay slot

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

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

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