Pipeline Control Hazards

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

MIPS Design Principles

Simplicity favors regularity

- 32 bit instructions

Smaller is faster

- Small register file

Make the common case fast

- Include support for constants

Good design demands good compromises

- Support for different type of interpretations/classes

Recall: MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats

R-type

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-type

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

J-type

op	immediate (target address)
6 bits	26 bits

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

Recall: MIPS Instruction Types

Arithmetic/Logical

- ADD, ADDU, SUB, SUBU, AND, OR, XOR, NOR, SLT, SLTU
- ADDI, ADDIU, ANDI, ORI, XORI, LUI, SLL, SRL, SLLV, SRLV, SRAV, SLTI, SLTIU
- MULT, DIV, MFLO, MTLO, MFHI, MTHI

Memory Access

- LW, LH, LB, LHU, LBU, LWL, LWR
- SW, SH, SB, SWL, SWR

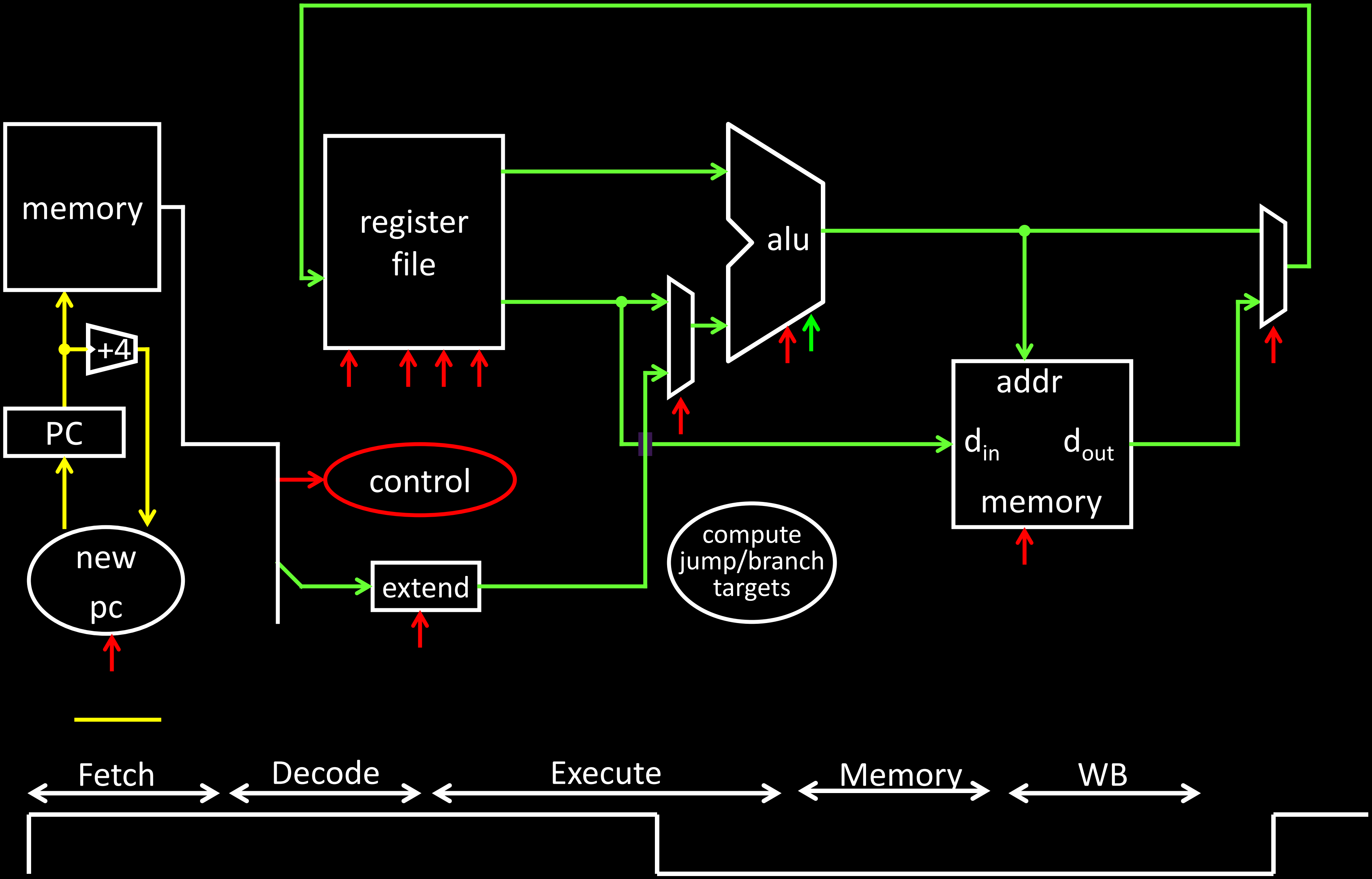
Control flow

- BEQ, BNE, BLEZ, BLTZ, BGEZ, BGTZ
- J, JR, JAL, JALR, BEQL, BNEL, BLEZL, BGTZL

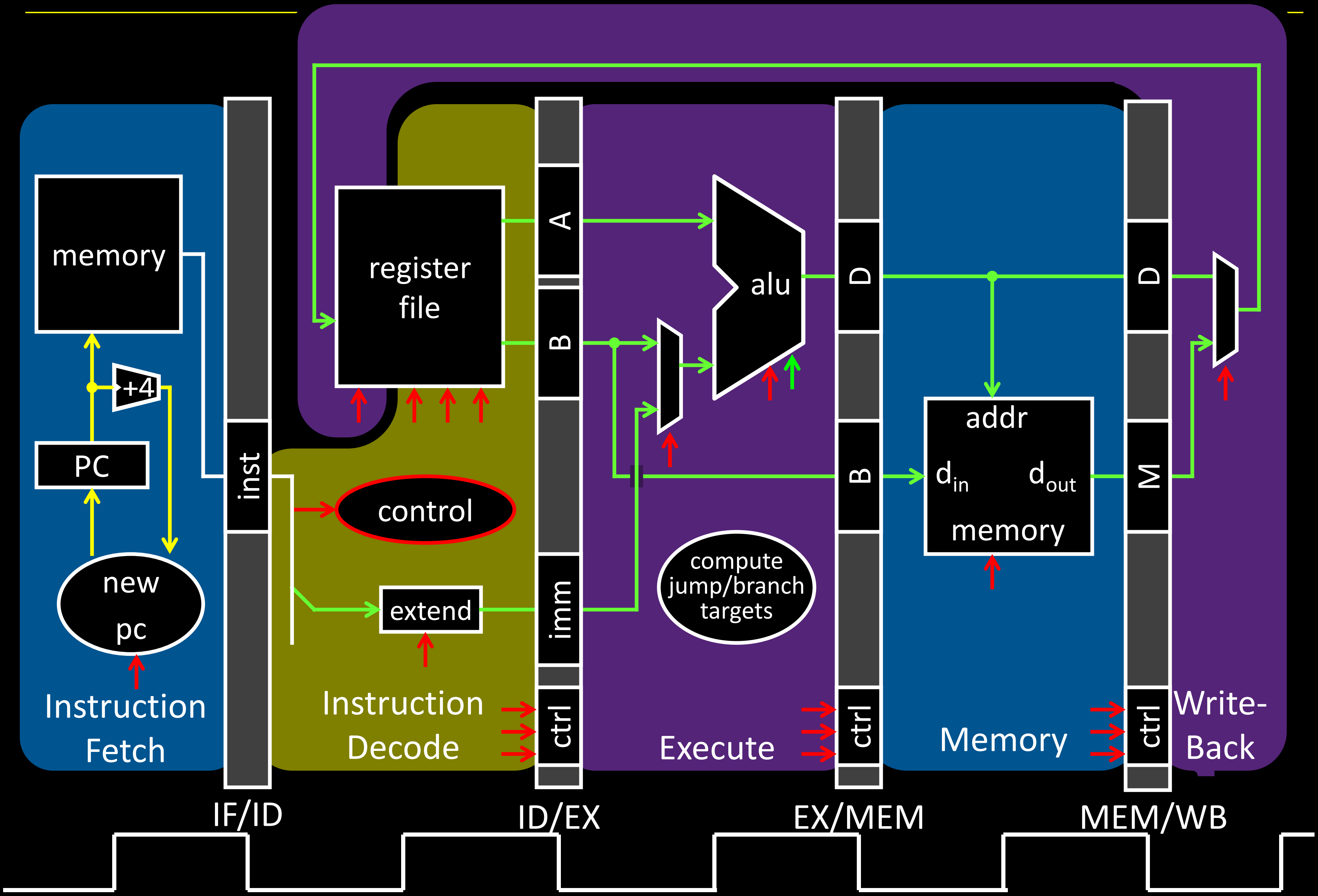
Special

- LL, SC, SYSCALL, BREAK, SYNC, COPROC

Pipelined Processor



Pipelined Processor



Time Graphs

Clock cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
nand		IF	ID	EX	MEM	WB			
lw			IF	ID	EX	MEM	WB		
add				IF	ID	EX	MEM	WB	
sw					IF	ID	EX	MEM	WB

Latency:

Throughput:

Concurrency:

CPI =

Next Goal

What about data dependencies (also known as a data hazard in a pipelined processor)?

$$\begin{tikzcd}[row sep=small, column sep=tiny]
\text{i.e. add} & \text{r3,} \arrow[dr] & \text{r1, r2} \\
\text{sub r5,} & & \text{r3, r4}
\end{tikzcd}$$

Data Hazards

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

Data Hazards

Stall

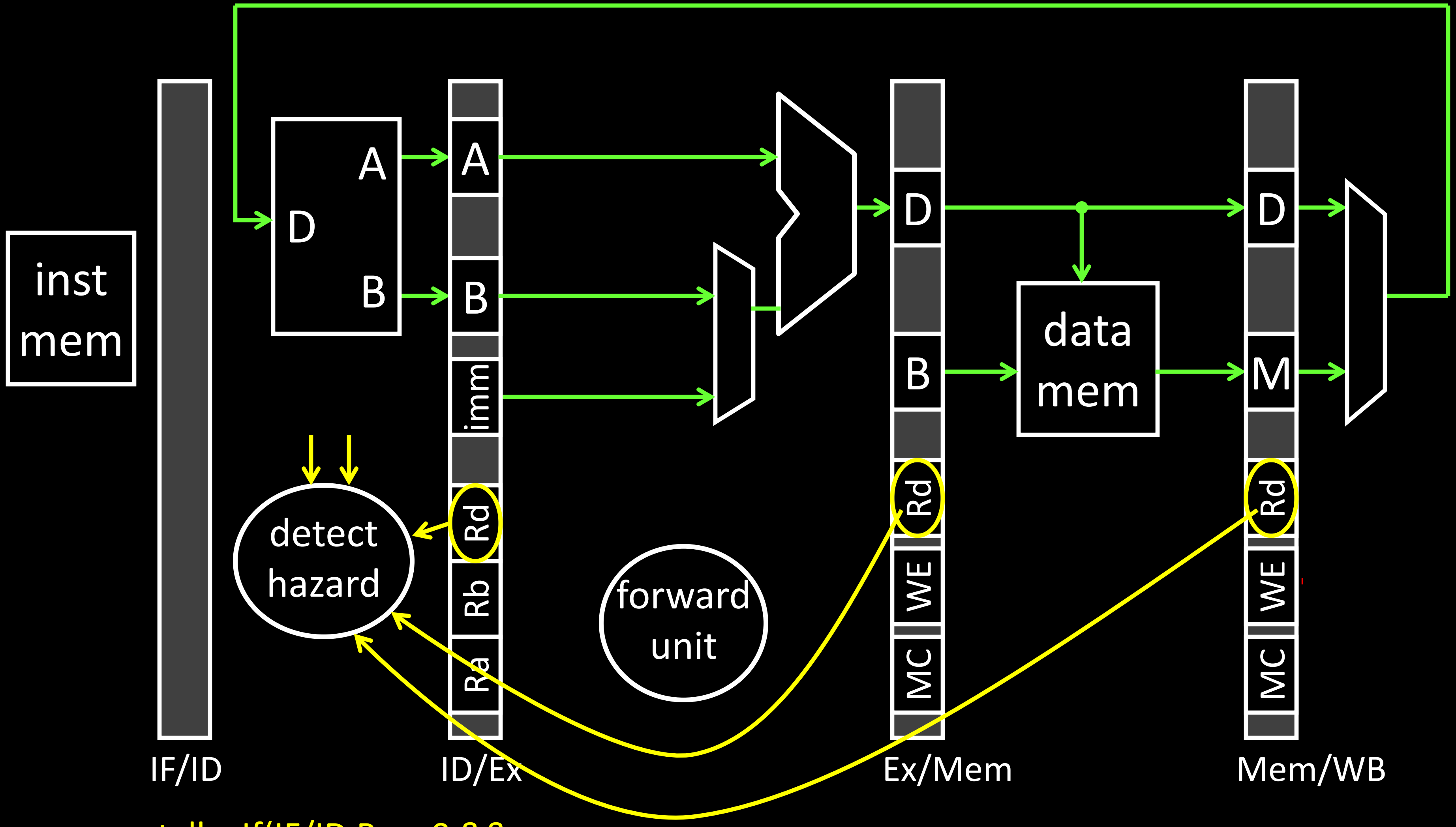
- Pause current and all subsequent instructions

Forward/Bypass

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

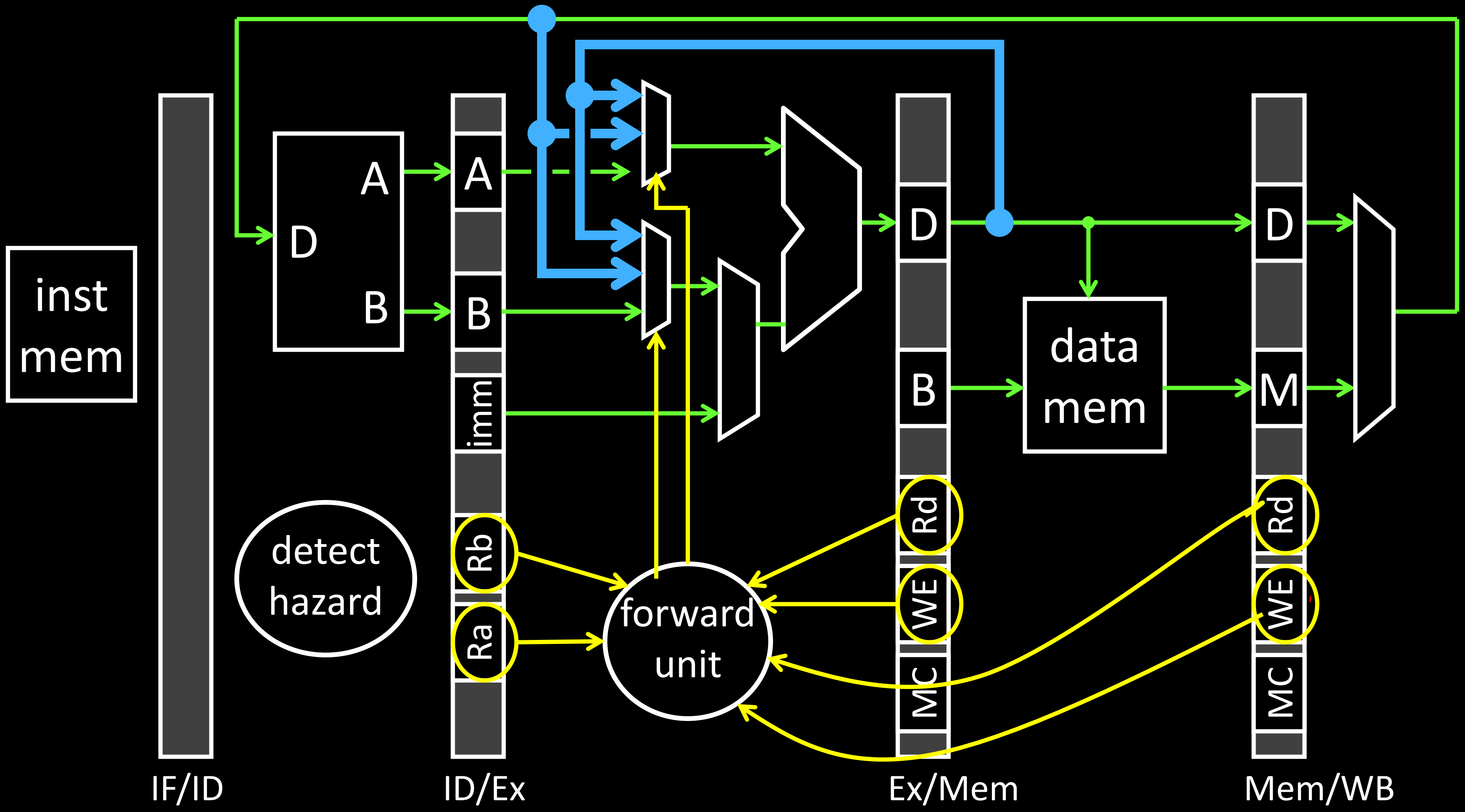
Tradeoffs?

Data Hazards



stall = If(IF/ID.Ra ≠ 0 &&
(IF/ID.Ra == ID/Ex.Rd
IF/ID.Ra == Ex/M.Rd
IF/ID.Ra == M/W.Rd))

Data Hazards



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M→Ex)
- Forwarding from Mem/WB register to Ex stage (W → Ex)
- RegisterFile Bypass

Stalling

Pause current and all subsequent instructions

“slow down the pipeline”

Stalling

time →

	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

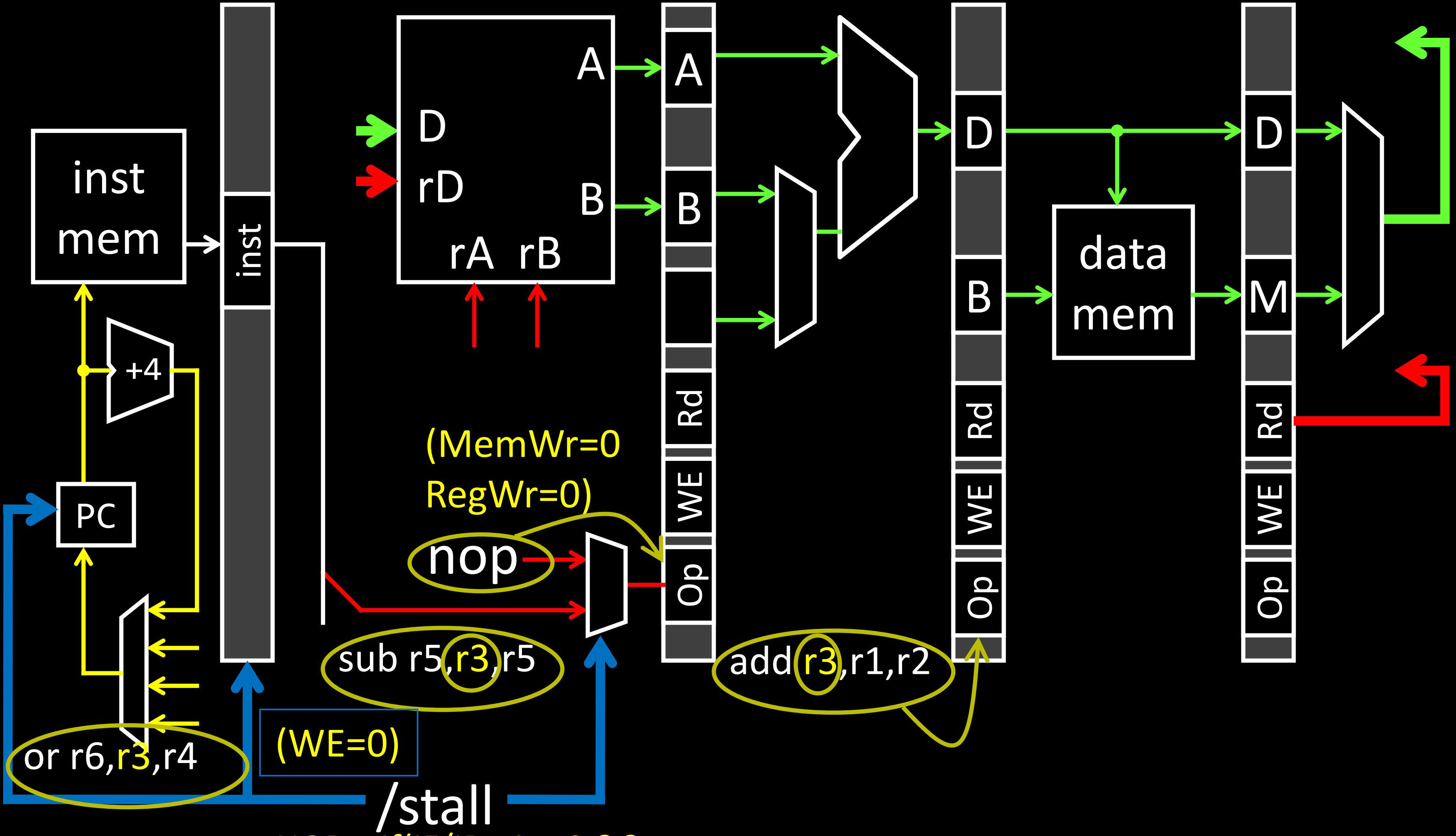
time →

Clock cycle

3 Stalls

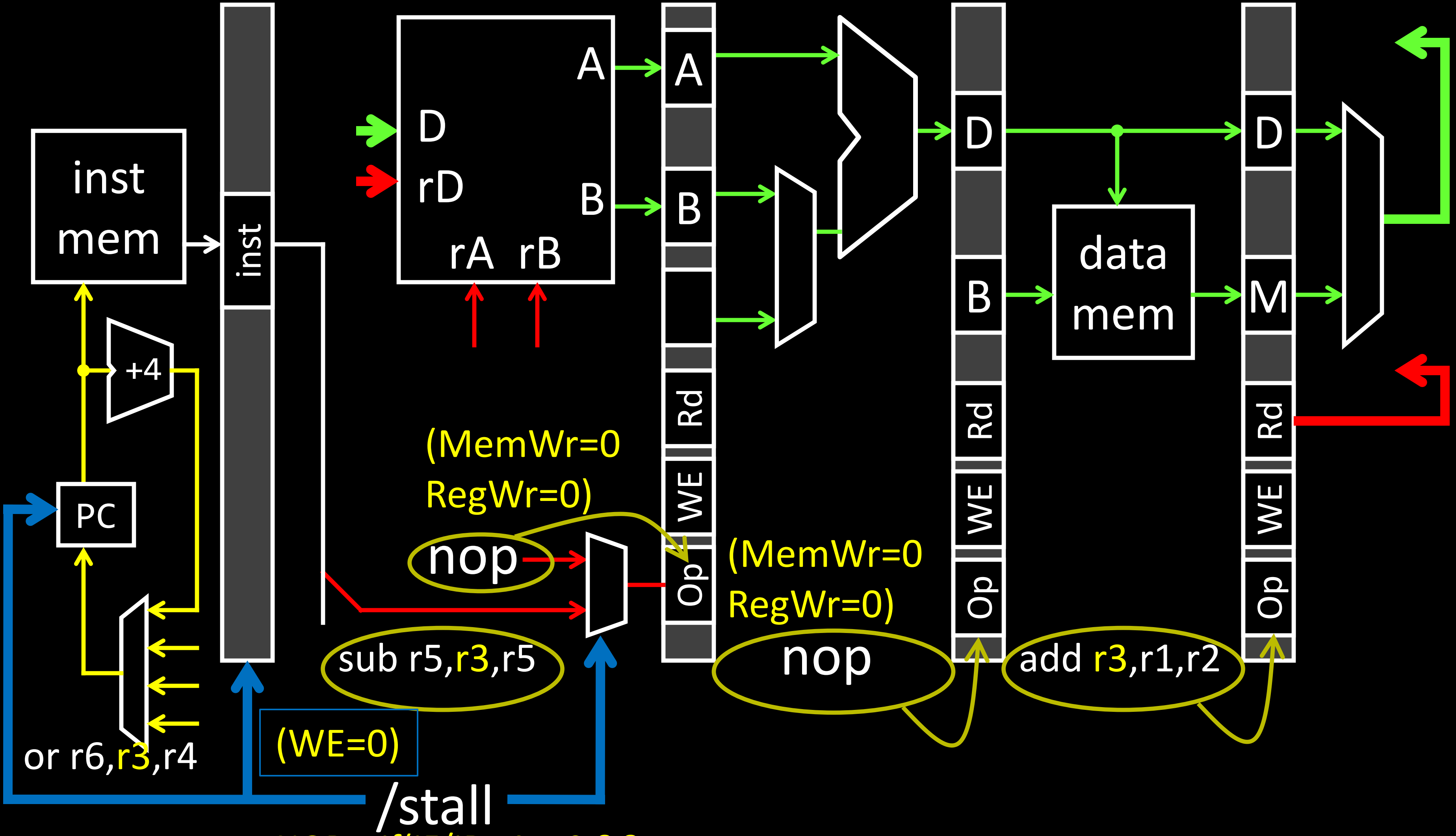
	1	2	3	4	5	6	7	8	
r3 = 10 add r3, r1, r2 r3 = 20	IF	ID	Ex	M	W				
sub r5, r3, r5		IF	ID	ID	ID	ID	Ex	M	W
or r6, r3, r4			IF	IF	IF	IF	ID	Ex	M
add r6, r3, r8							IF	ID	Ex

Stalling



NOP = If(IF/ID.rA ≠ 0 &&
(IF/ID.rA==ID/Ex.Rd
IF/ID.rA==Ex/M.Rd
IF/ID.rA==M/W.Rd))

Stalling



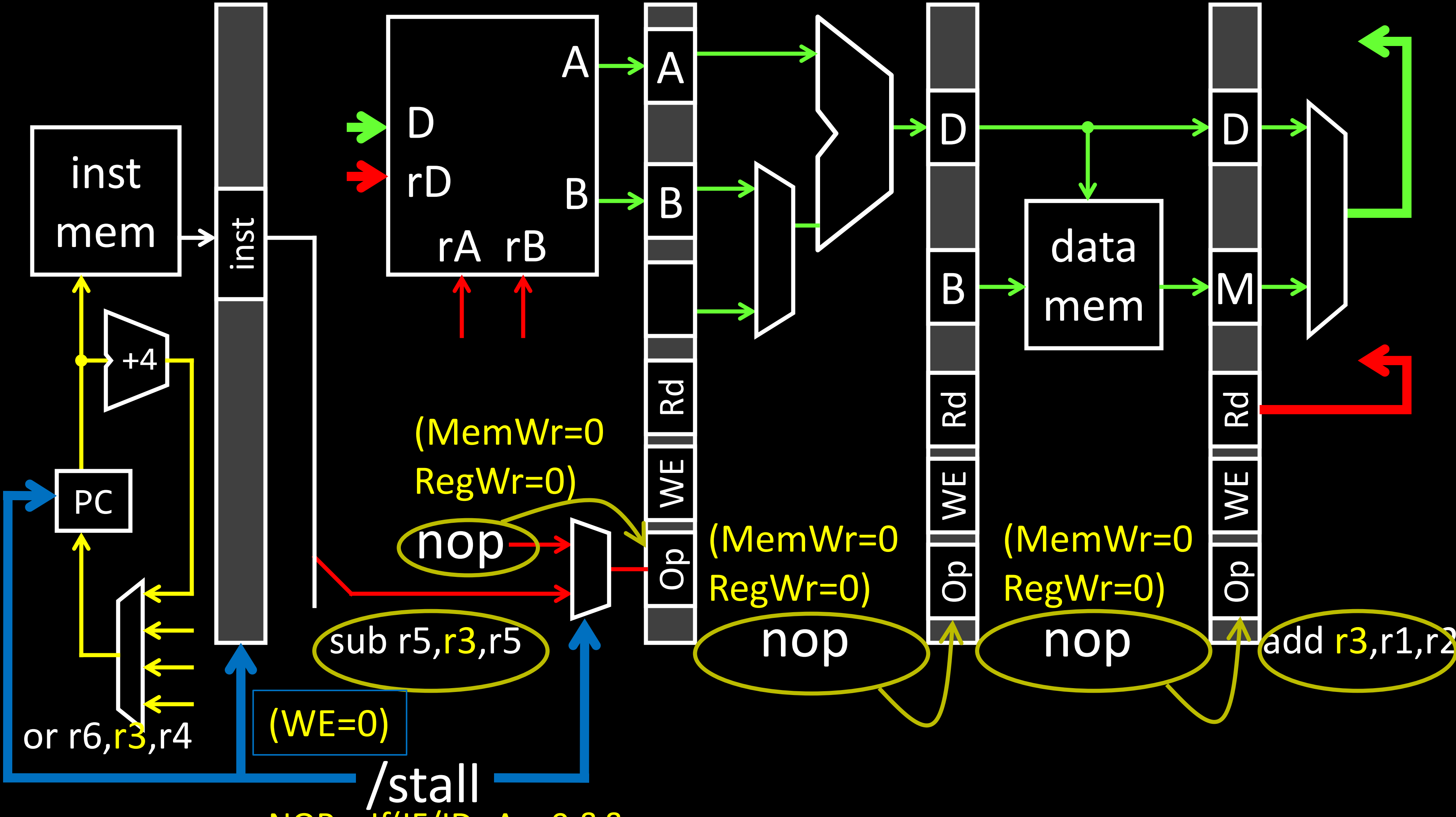
NOP = If(IF/ID.rA ≠ 0 &&

(IF/ID.rA==ID/Ex.Rd

IF/ID.rA==Ex/M.Rd

IF/ID.rA==M/W.Rd))

Stalling



NOP = If(IF/ID.rA ≠ 0 &&

(IF/ID.rA==ID/Ex.Rd

IF/ID.rA==Ex/M.Rd

IF/ID.rA==M/W.Rd))

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

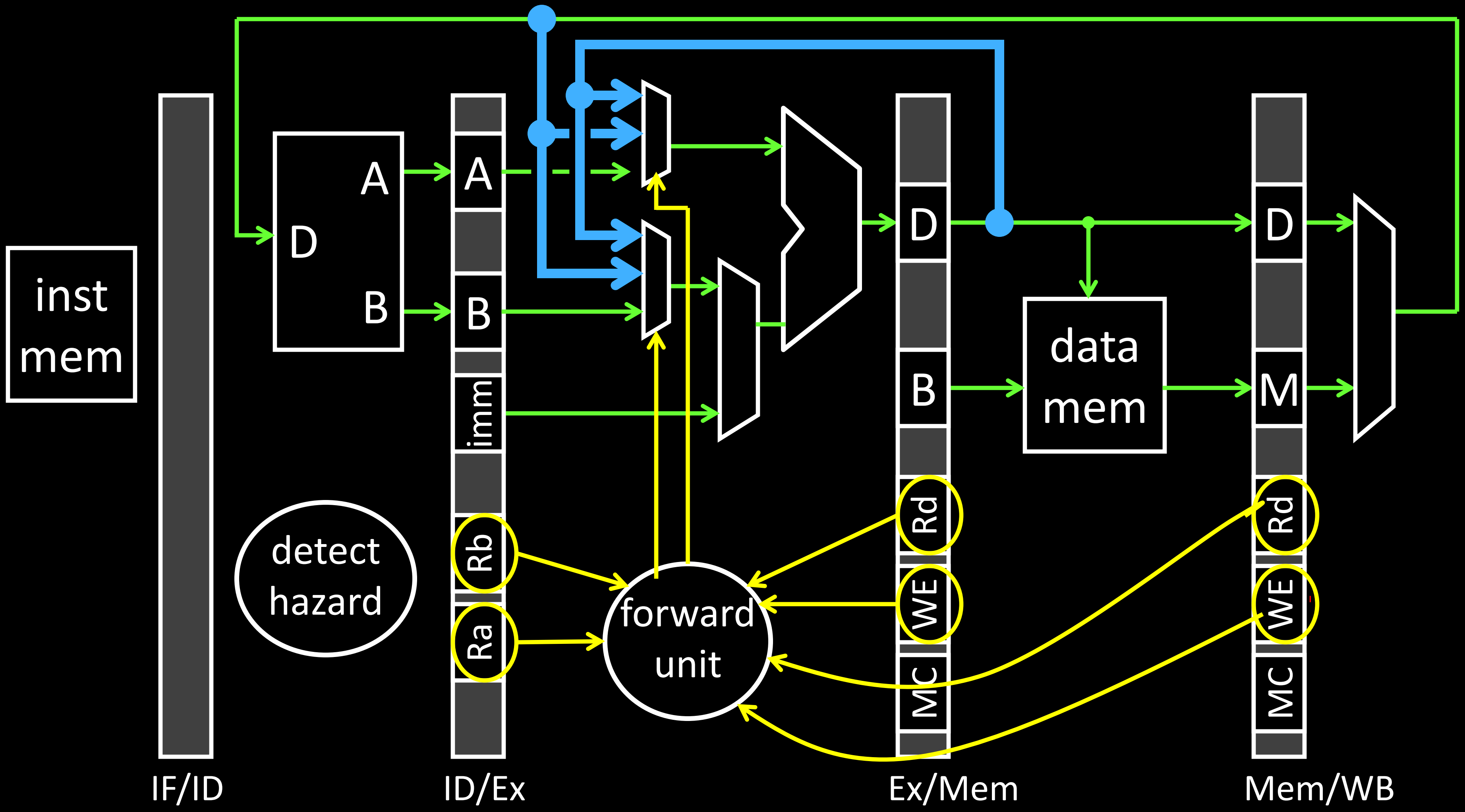
Forwarding

Forwarding bypasses some pipelined stages forwarding a result to a dependent instruction operand (register).

Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M→Ex)
- Forwarding from Mem/WB register to Ex stage (W→Ex)
- RegisterFile Bypass

Forwarding Datapath



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M→Ex)
- Forwarding from Mem/WB register to Ex stage (W → Ex)
- RegisterFile Bypass

Forwarding Datapath

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

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

Mem/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)

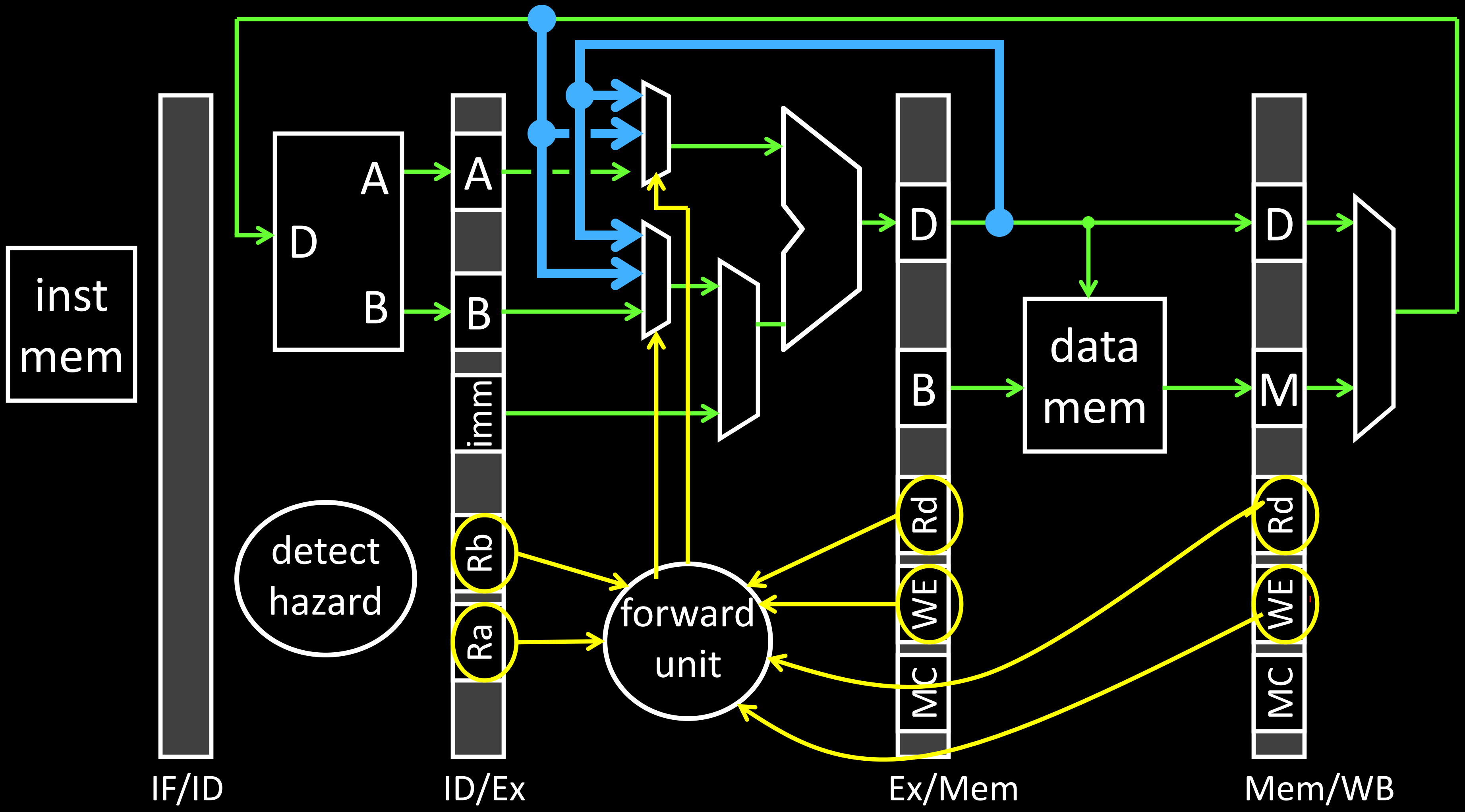
- Resolve:

  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

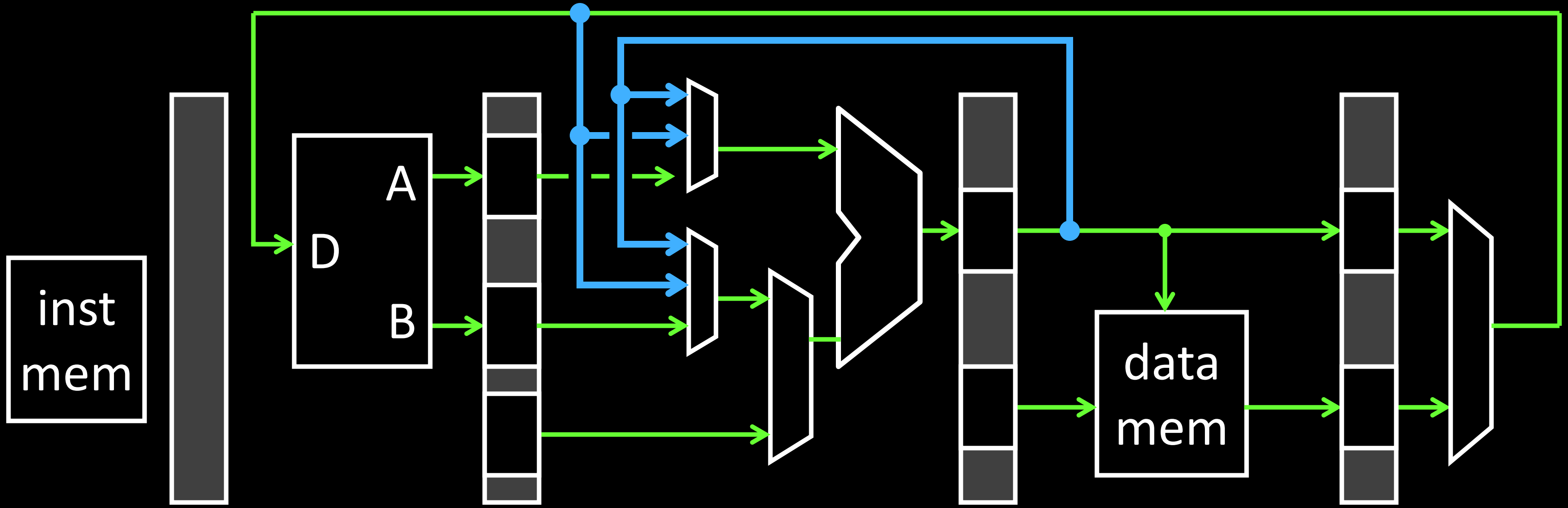
Forwarding Datapath



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M→Ex)
- Forwarding from Mem/WB register to Ex stage (W → Ex)
- RegisterFile Bypass

Forwarding Datapath



add r3, r1, r2	IF	ID	Ex	M	W				
sub r5, r3, r1		IF	ID	Ex	M	W			
or r6, r3, r4			IF	ID	Ex	M	W		
add r6, r3, r8				IF	ID	Ex	M	W	

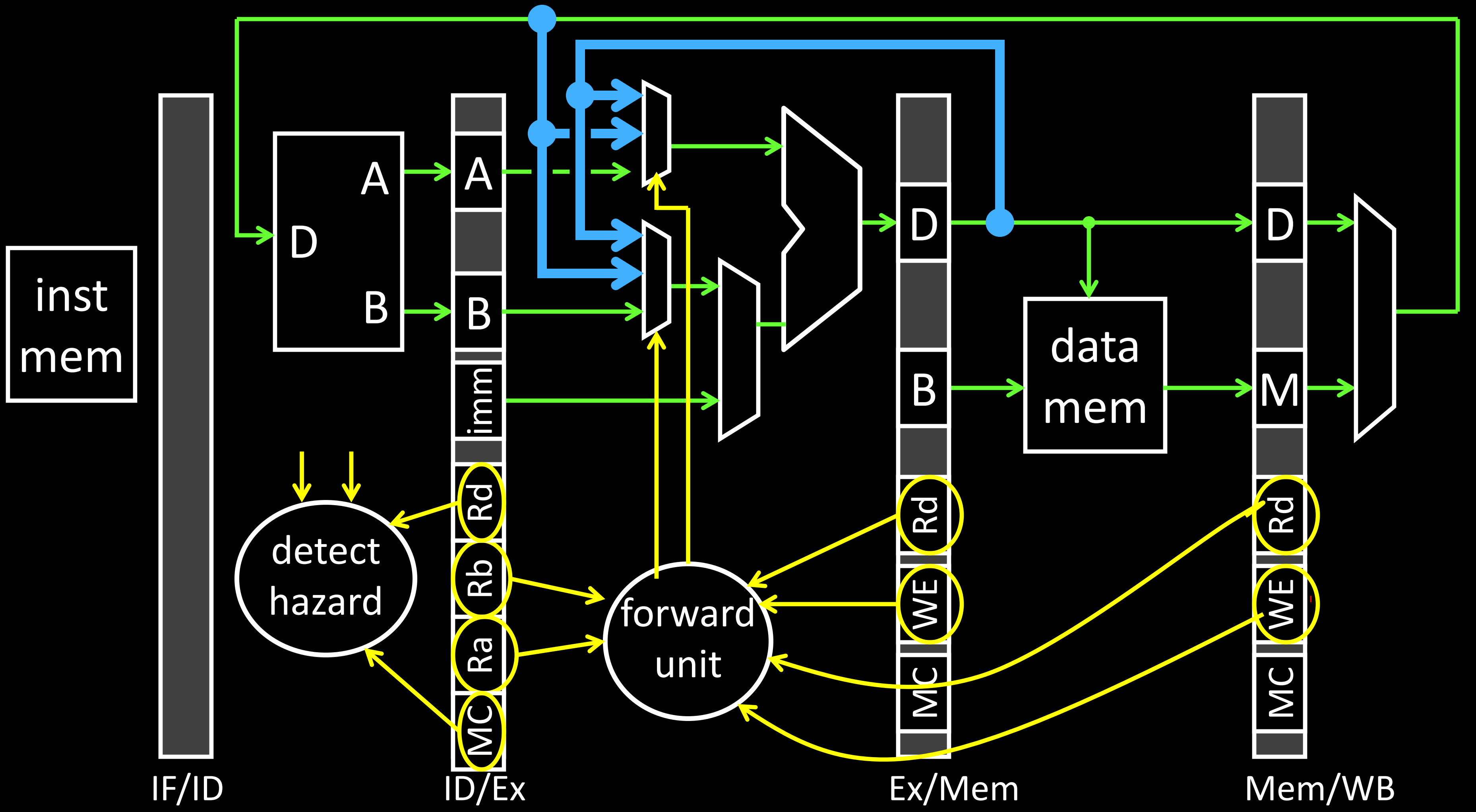
Memory Load Data Hazard

What happens if data dependency after a load word instruction?

Memory Load Data Hazard

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

Memory Load Data Hazard

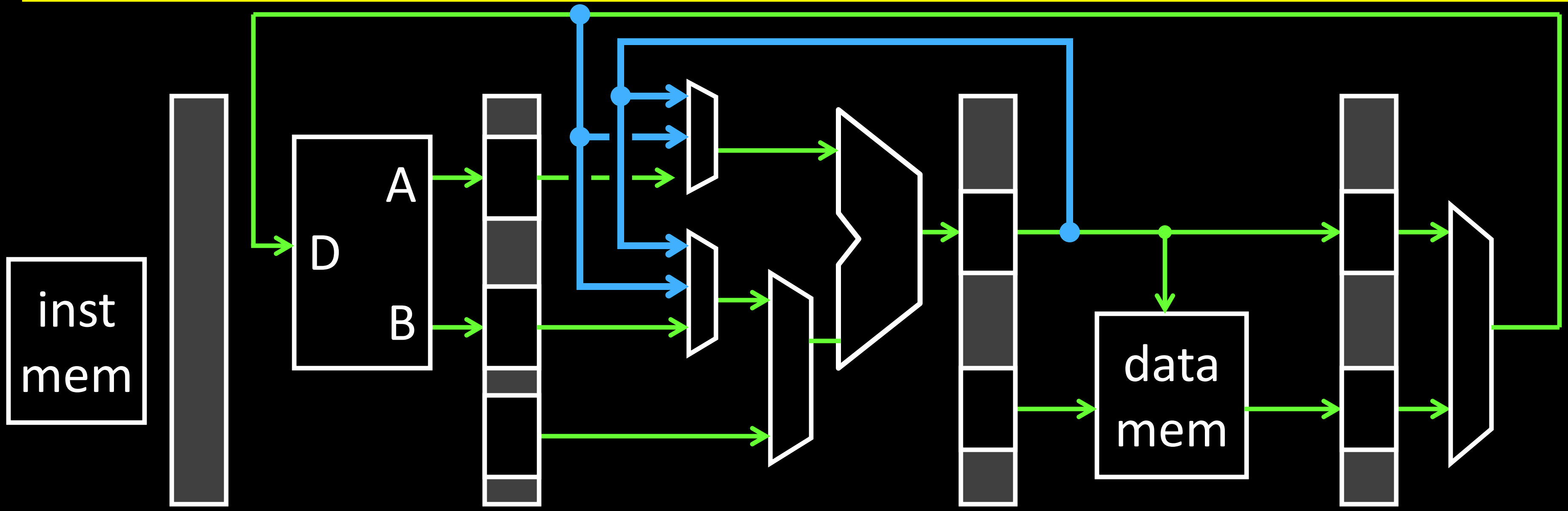


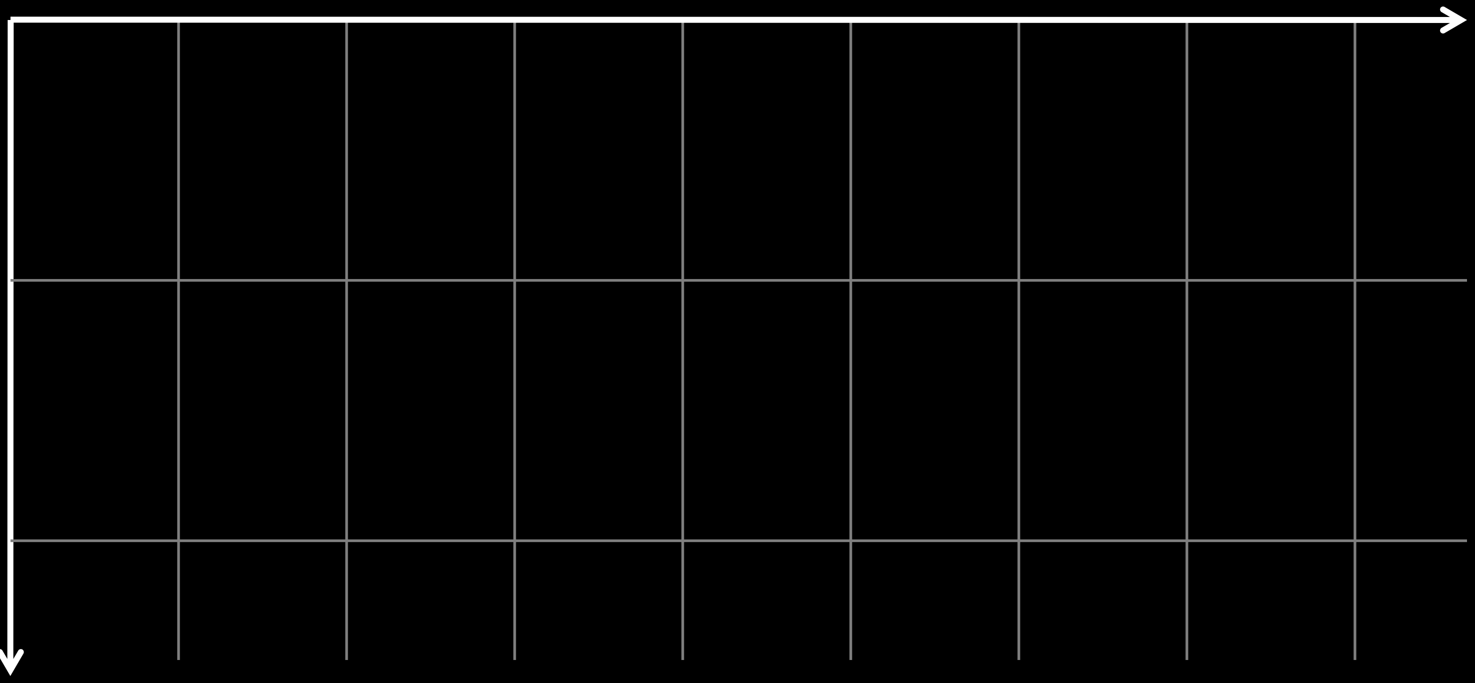
Stall =
If(ID/Ex.MemRead &&
IF/ID.Ra == ID/Ex.Rd

Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M→Ex)
- Forwarding from Mem/WB register to Ex stage (W → Ex
- RegisterFile Bypass

Memory Load Data Hazard





Memory Load Data 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 26th in evening

- **Location: GSHG76: Goldwin Smith Hall room G76**
- Time: We will start at 7:30pm sharp, so come early
- Prelim Review: Today, Thur 6-8pm in Upson B14 and Fri, 5-7pm in Phillips 203

- 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

HW2 was due *yesterday*

- Last day to submit tomorrow night, Friday 11:59pm
- HW2 solutions released on Saturday

Project1 (PA1) due next Monday, March 4th

- 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

Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2013sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

- Tuesday, February 26th
- Thursday, March 28th
- Thursday, April 25th

Schedule is subject to change

Collaboration, Late, Re-grading Policies

"Black Board" Collaboration Policy

- Can discuss approach together on a "black board"
- Leave and write up solution independently
- Do not copy solutions

Late Policy

- Each person has a total of ***four*** "slip days"
- Max of ***two*** slip days for any individual assignment
- Slip days deducted first for *any* late assignment,
  cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- 25% deducted per day late after slip days are exhausted

Regrade policy

- Submit written request to lead TA,
  and 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.

Next Goal

What about branches?

A control hazard occurs if there is a control instruction (e.g. BEQ) because the program counter (PC) following the control instruction is not known until the control instruction computes if the branch should be taken or not.

e.g.

```
0x10:      beq r1, r2, L
0x14:      add r3, r0, r3
0x18:      sub r5, r4, r6
0x1C: L:   or   r3, r2, r4
```

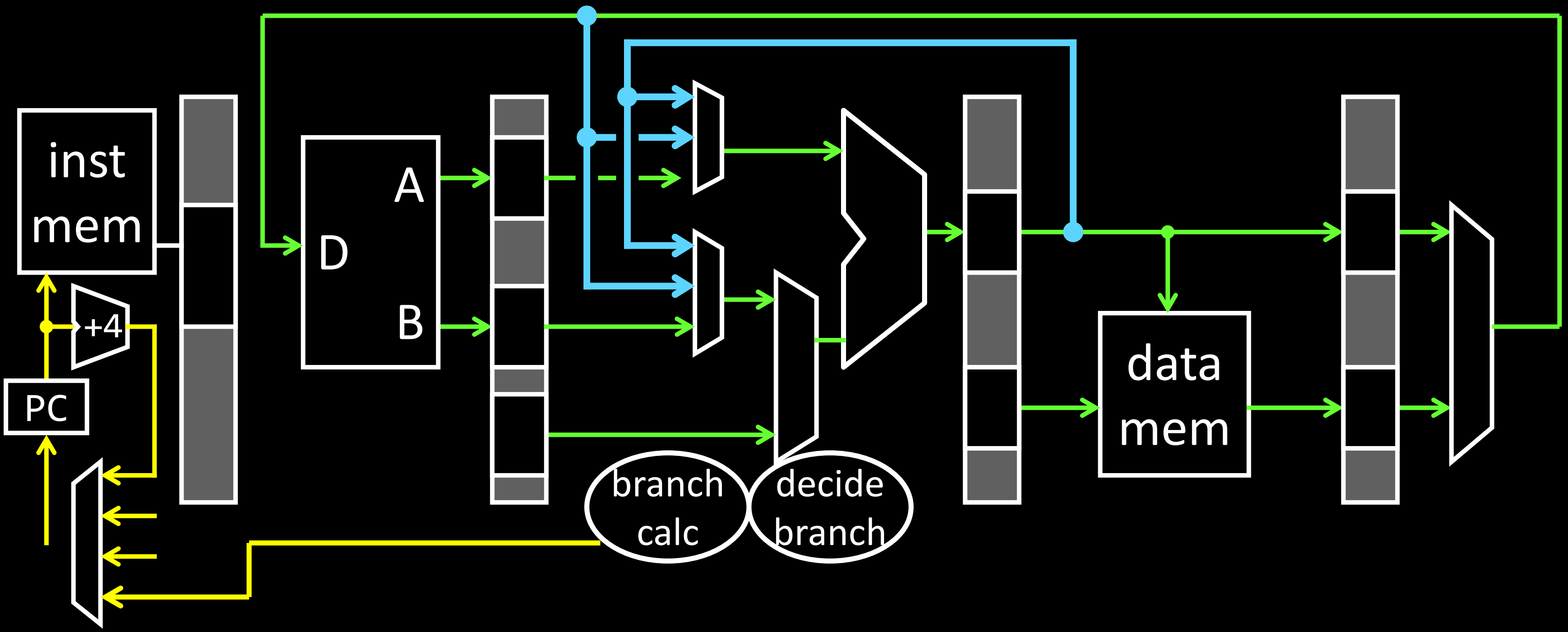
Control Hazards

Control Hazards

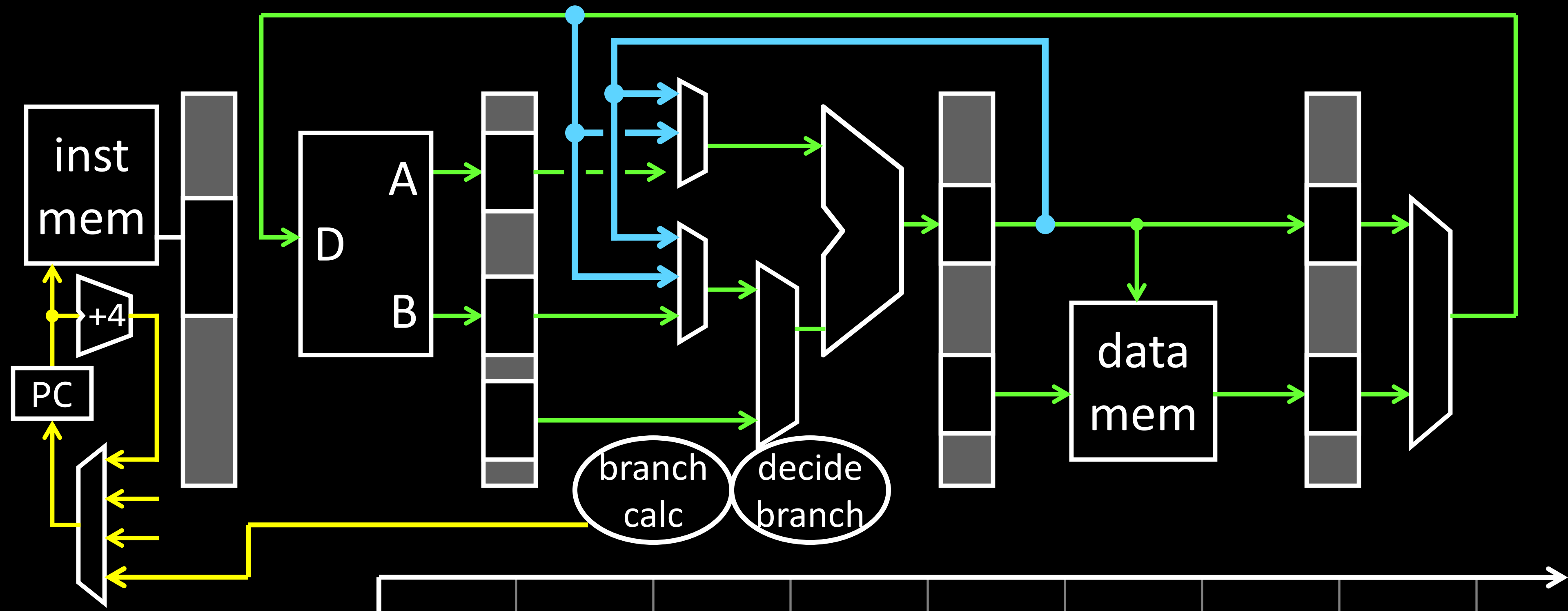
- instructions are fetched in stage 1 (IF)
- branch and jump decisions occur in stage 3 (EX)
- i.e. next PC is not known until ***2 cycles*** *after* branch/jump

What happens to instr following a branch, if branch taken?

Control Hazards



Control Hazards



10: beq r1, r2, L								
14: add r3, r0, r3								
18: sub r5, r4, r6								
1C: L: or r3, r2, r4								

Takeaway

Control hazards occur because the PC following a control instruction is not known until control instruction computes if branch should be taken or not.

If branch taken, then need to zap/flush instructions.

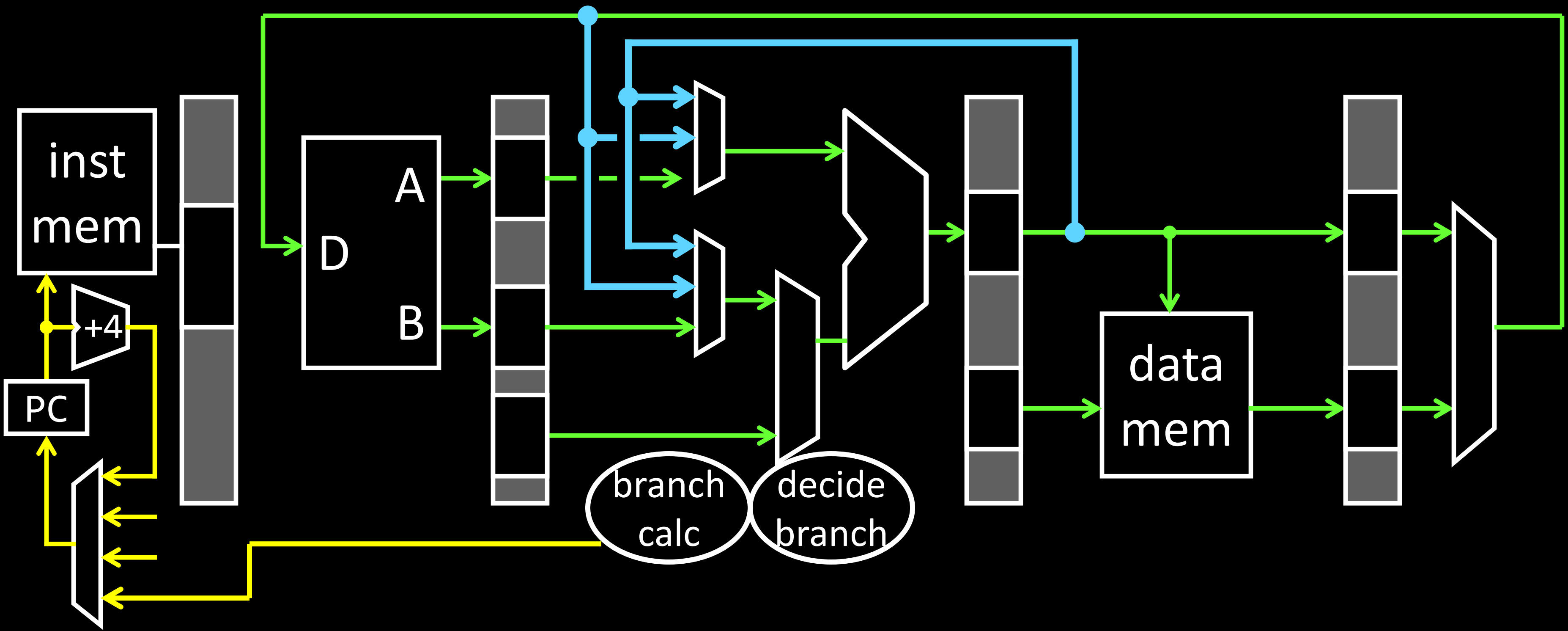
There is a performance penalty for branches:

Need to stall, then may need to zap (flush) subsequent instructions that have already been fetched.

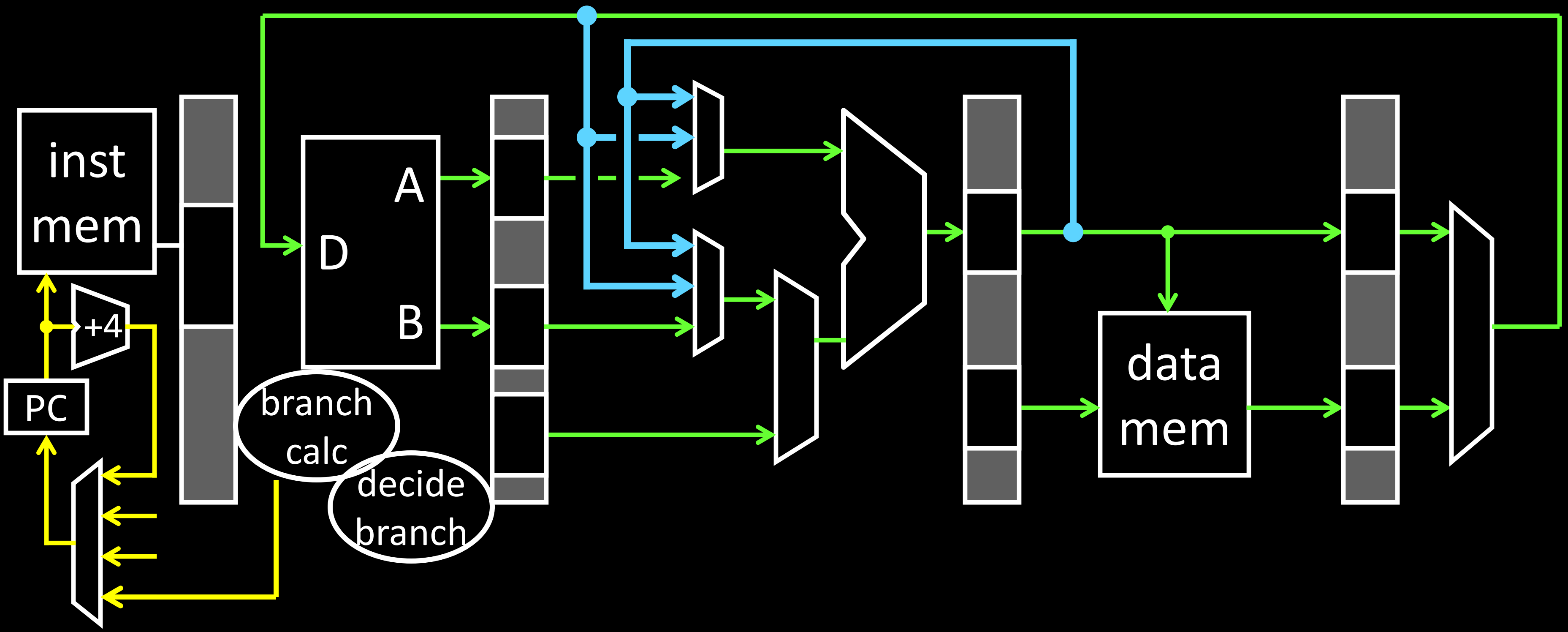
Next Goal

Can we reduce the cost of a control hazard?

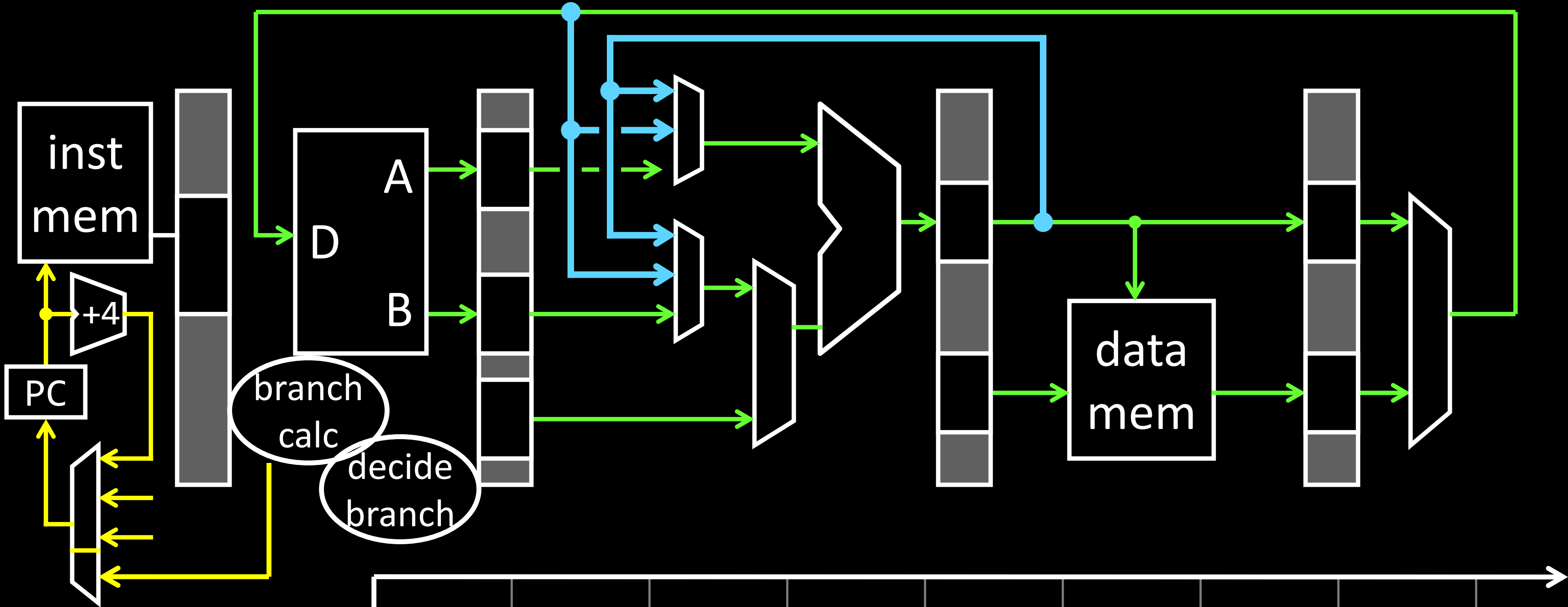
Control Hazards



Control Hazards



Control Hazards



10: beq r1, r2, L

14: add r3, r0, r3

18: sub r5, r4, r6

1C: 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
- Can optimize and move branch and jump decision to stage 2 (ID)

  i.e. next PC is not known until ***1 cycles*** *after* branch/jump

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

Takeaway

Control hazards occur because the PC following a control instruction is not known until control instruction computes if branch should be taken or not. If branch taken, then need to zap/flush instructions. There still a performance penalty for branches: Need to stall, then may need to zap (flush) subsequent instructions that have already been fetched.

We can reduce cost of a control hazard by moving branch decision and calculation from Ex stage to ID stage. This reduces the cost from flushing two instructions to only flushing one.

Takeaway

Control hazards occur because the PC following a control instruction is not known until control instruction computes if branch should be taken or not. If branch taken, then need to zap/flush instructions. There still a performance penalty for branches: Need to stall, then may need to zap (flush) subsequent instructions that have already been fetched.

We can reduce cost of a control hazard by moving branch decision and calculation from Ex stage to ID stage. This reduces the cost from flushing two instructions to only flushing one.

Next Goal

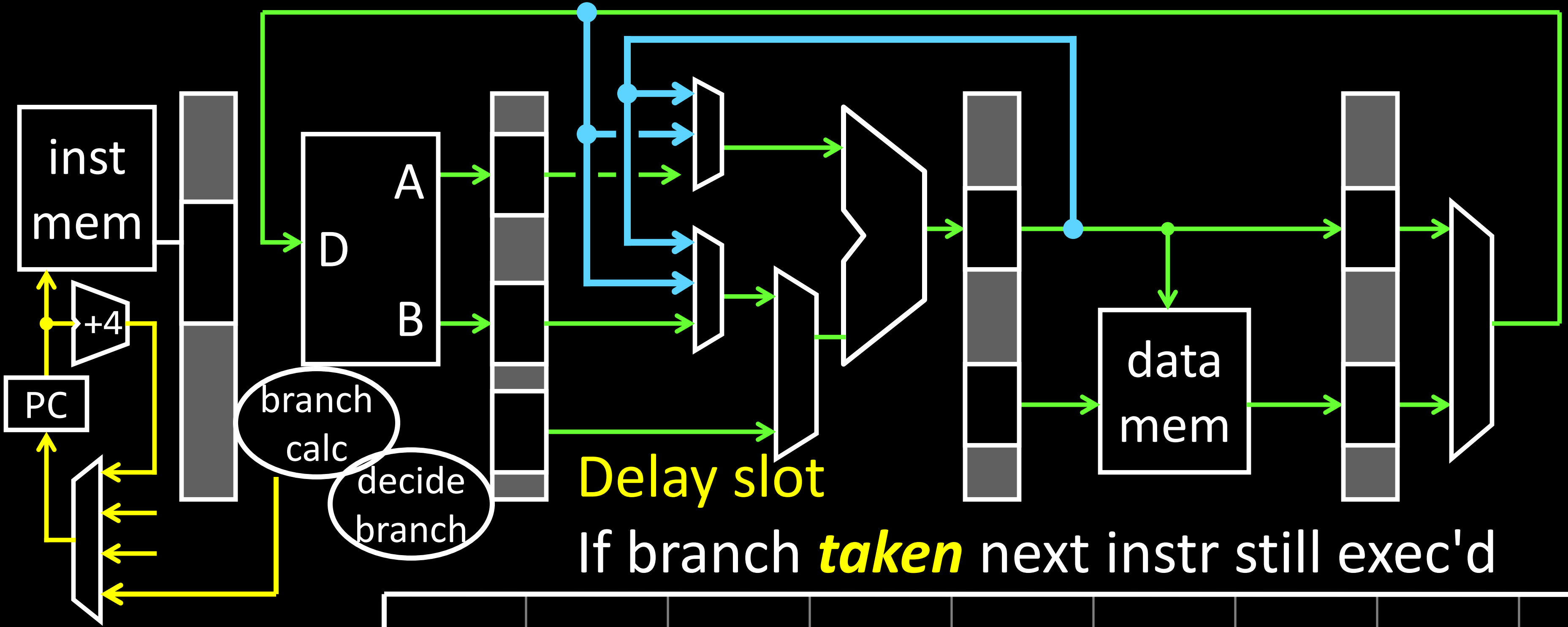
Can we reduce the cost of a control hazard further?

Delay Slot

Delay Slot

- ISA says N instructions after branch/jump ***always*** executed
  - MIPS has 1 branch delay slot
  - i.e. Whether branch taken or not, instruction following branch is ***always*** executed

Delay Slot

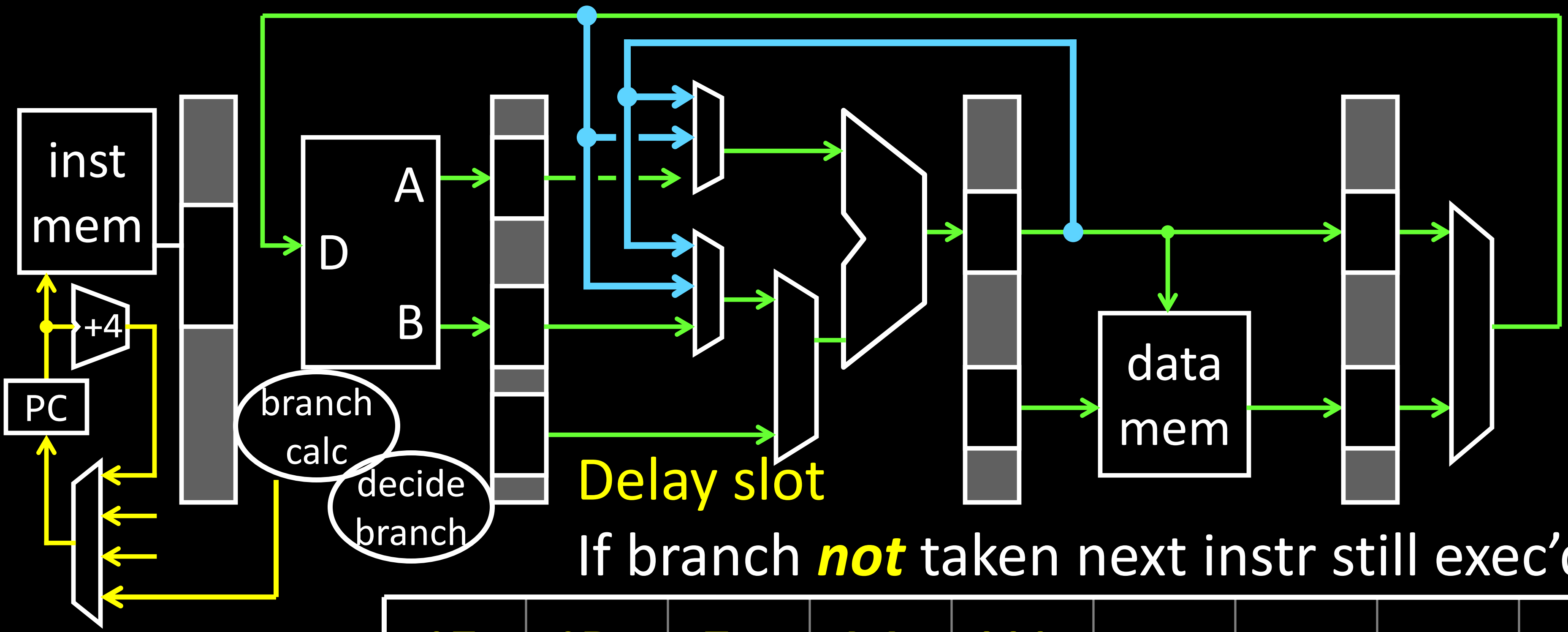


Delay slot

If branch ***taken*** next instr still exec'd

10: beq r1, r2, L	IF	ID	Ex	M	W				
14: add r3, r0, r3		IF	ID	Ex	M	W			
18: sub r5, r4, r6									
1C: L: or r3, r2, r4			IF	ID	Ex	M	W		

Delay Slot



Delay slot

If branch ***not*** taken next instr still exec'd

10: beq r1, r2, L	IF	ID	Ex	M	W				
14: add r3, r0, r3		IF	ID	Ex	M	W			
18: sub r5, r4, r6			IF	ID	Ex	M	W		
1C: L: or r3, r2, r4				IF	ID	Ex	M	W	

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
- Can optimize and move branch and jump decision to stage 2 (ID)
  i.e. next PC is not known until ***1 cycles*** *after* branch/jump

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

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

Takeaway

Control hazards occur because the PC following a control instruction is not known until control instruction computes if branch should be taken or not. If branch taken, then need to zap/flush instructions. There still a performance penalty for branches: Need to stall, then may need to zap (flush) subsequent instructions that have already been fetched.

We can reduce cost of a control hazard by moving branch decision and calculation from Ex stage to ID stage. This reduces the cost from flushing two instructions to only flushing one.

Delay Slots can potentially increase performance due to control hazards by putting a useful instruction in the delay slot since the instruction in the delay slot will ***always*** be executed. Requires software (compiler) to make use of delay slot. Put nop in delay slot if not able to put useful instruction in delay slot.

Next Goal

Can we reduce the cost of a control hazard ***even further***?

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

Speculative Execution: Loops

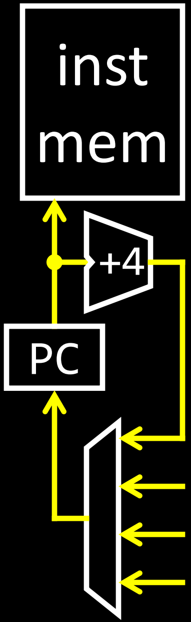
Pipeline so far

- "Guess" (predict) branch not taken

We can do better!

- Make prediction based on last branch:
- Predict "take branch" if last branch "taken"
- Or Predict "do not take branch" if last branch "not taken"

- Need one bit to keep track of last branch

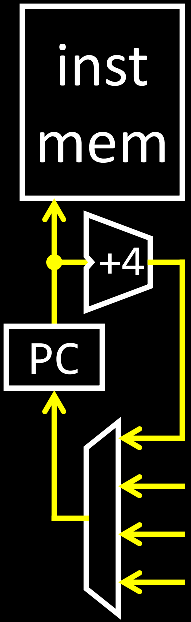


Speculative Execution: Loops

While (r3 ≠ 0)

```
Top:    BEQZ r3, End
        ~~~~~~~~
        ~~~~~~~~
        ~~~~~~~~
        J Top
End:

Top2:   BEQZ r3, End
        ~~~~~~~~
        ~~~~~~~~
        ~~~~~~~~
        J Top2
End2:
```



Takeaway

Control hazards occur because the PC following a control instruction is not known until control instruction computes if branch should be taken or not. If branch taken, then need to zap/flush instructions. There still a performance penalty for branches: Need to stall, then may need to zap (flush) subsequent instructions that have already been fetched.

We can reduce cost of a control hazard by moving branch decision and calculation from Ex stage to ID stage. This reduces the cost from flushing two instructions to only flushing one.

Delay Slots can potentially increase performance due to control hazards by putting a useful instruction in the delay slot since the instruction in the delay slot will ***always*** be executed. Requires software (compiler) to make use of delay slot.

Speculative execution (guessing/predicting) can reduce costs of control hazards due to branches. If guess correct, no cost to branch. If guess wrong, need to flush pipeline.

Hazards Summary

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